Evolving Genotype to Phenotype Mappings with a Multiple-Chromosome Genetic Algorithm

Rick Chow

Division of Mathematics and Computer Science
University of South Carolina Spartanburg
800 University Way, Spartanburg, SC, 29303, U.S.A.
rchow@uscs.edu
http://faculty.uscs.edu/rchow

Abstract. This paper presents an evolutionary coding method that maps genotype to phenotype in a genetic algorithm. Unlike traditional genetic algorithms, the proposed algorithm involves mating and reproduction of cells that have multiple chromosomes instead of single chromosomes. The algorithm also evolves the mapping from genotype to phenotype rather than using a fixed mapping that is associated with one particular encoding method. The genotype-to-phenotype mapping is conjectured to explicitly capture important schema information. Some empirical results are presented to demonstrate the efficacy of the algorithm with some GA-Hard problems.

1 Background

Genetic Algorithms (GAs) are a class of powerful algorithms for solving optimization problems using evolution as a search strategy. Although GAs can successfully solve many difficult optimization problems, some problems are still hard for GAs to solve and such problems are called *GA-Hard problems*. Particularly, there is a class of problems called deceptive problems that exploit the weaknesses in the encodings of chromosomes [1], [2]. This paper purposes a GA based method to solve GA-Hard problems by evolving a genotype-to-phenotype mapping that also capture schema information.

Let's understand how an optimization problem can be deceptive by understanding basic schema theory, which attempts to explain the ability of GAs to perform global search in a large and usually high dimensional problem space [3]. The theory hypothesizes the existence of building blocks that are called *schemata* in the chromosomes. A schema is a chromosome pattern that consists of three symbols, 0, 1, and *, where 0 and 1 are the fixed *defining bits* and the symbol * is a wildcard symbol that matches a 0 or 1. While an 8-bit binary chromosome 10010011 presents a single point in the solution space, the schema 10*1***1 represents a *hyper-plane* in the solution space that consists of 16 data points because each * symbol could be a 0 or 1. A schema is said to *sample* its hyper-plane because as it survives from generation to generation, data points in its hyper-plane are tested repeatedly to be as good solutions in the solu-

K. Deb et al. (Eds.): GECCO 2004, LNCS 3102, pp. 1006–1017, 2004.

tion space. Through genetic recombination, mutation, and selection, promising hyperplanes in the solution space are given higher sampling rates by an exponential increase in quantities of the schemata in the chromosome population.

A binary chromosome may consist of chunks of binary sequences each of which is called a *gene* that corresponds to a parameter of the optimization function. Usually, a particular binary number encoding scheme is chosen to translate such a binary sequence into a numeric value for the parameter. For example, 00001111 can be translated to 15 (decimal) using two's complement coding method. The binary chromosome represents the *genotype* whereas the decoded parameter list represents the *phenotype*. Note that a parameter may not be storing numeric information; it could also represent structural information of a phenotype, e.g., architectural information of a neural network.

1.1 Binary Encoding Methods

This paper focuses on binary encodings where a parameter in the phenotype is encoded as a binary number. The simplest encoding method is the traditional two's complement binary encoding method. Unfortunately, such an encoding method may cause discontinuity in the search process because a single step in the phenotypic space from, say, 15 to 16, would require 5 steps in the genotypic space from 00001111 to 00010000. In this example, the Hamming distance between 00001111 and 00010000 is 5. There were many attempts to use alternative encoding methods such as Gray coding to alleviate this problem [4], [5]. The encoding and decoding steps of Gray code are detailed in [6]. The significance of Gray code is that consecutive binary numbers differ by only one digit, that is, the Hamming distance between two consecutive numbers is one. For example, numbers 0 to 8 in Gray code are as follows: 0000, 0001, 0011, 0110, 0111, 0101, 0100. However, even with Gray coding, a problem space can still be difficult for GAs to search in.

1.2 Deceptive Problems

Some of the problems that are hard for GAs are called deceptive problems. Before defining what a deceptive problem is, we need to understand what kind of schemata could lead to deception. Detailed discussions of deception problems can be found in [1], [2], [7]. A schema's fitness depends on the fitness values of its sampling points. For instance, the schema 10*1***1 would probably have a high fitness value if most (if not all) of the 16 data points carry high fitness values. Schemata with high fitness values have a higher chance to survive into the next generation. A schema with N defining bits is called an order-N schema. For example, the schemata *0**0, *0**1, *1**0, and *1**1 are all order-2 schemata. Moreover, schemata that are of the same order and have defining bits at the same locations are essentially *primary competitors* in the selection process because such schemata compete as orthogonal planes in the hyperspace. A lower order hyper-plane also contains a collection of higher order hyper-planes that share common defining bits with the lower order hyper-plane. For

example, the order-2 hyper-plane 0***0 contains some order-3 hyper-planes such as 00**0 and 01**0 whereas 1***0 contains another set of order-3 hyper-planes such as 10**0 and 11**0. An order-N schema and the order-K schemata (N<K) that it contains are said to be *relevant* to one another. When the lower order schema competes in a selection process, all of its own relevant higher order schemata also participate in the competition. For example, the competition between the order-2 schemata 0***0, 0***1, 1***0, and 1***1 also involves the competitions between the order-3 schemata 00**0, 01**0, 00**1, 01**1, 10**0, 11**0, 10**1, and 11**1. On the other hand, the competitions between higher hyper-planes also involve competition of their relevant lower order hyper-planes.

A deception occurs when a hyper-plane competition of higher order K leads to a global winner that is radically different, in terms of Hamming distance, from the global winner of the competitions among the relevant order-N hyper-planes, where N<K. For example, consider the fitness values for the following deceptive function f1:

Table 1. A Deceptive Function f1

Function f1 Values	
f1(000) = 28	f1(001) = 26
f1(010) = 22	f1(100) = 14
f1(110) = 0	f1(011) = 0
f1(101) = 0	f1(111) = 30

With function f1, the global winner of an order-3 hyper-plane competition is 111. Consider 111's relevant hyper-plane *11, when it competes with *00, *01, and *10 in order-2 competitions, the average fitness of *11 is 15 ((0+30)/2) versus the average fitness values of *00, *01, and *10 are 21, 13, and 11, respectively. The schemata *00 will most likely win the order-2 hyper-plane competition. The radical difference, or long Hamming distance, between the two global winners 111 and *00 creates deception. We are now ready to define a deception problem:

A *deceptive problem* is any problem of a specific order K that involves deception in one or more relevant lower order N hyper-plane competitions where N<K.

A deceptive problem misleads a GA to converge incorrectly to a region in the problem space called the *deceptive attractor*.

2 Previous Work on Solving Deceptive Problems

Different techniques have been proposed to tackle deceptive problems. Some techniques focus on the encoding methods. According to the work in [1],[8], a tagged bit that specifies the phenotypic bit location is attached to each binary bit in the chromosome. A binary string 11011001 is represented as a list of order pairs of bit value-bit location: ((1 1) (2 1) (3 0) (4 1) (5 1) (6 0) (7 0) (8 1)). The order of the ordered pairs may be rearranged or randomized but bits on two chromosomes are first lined up in the same order before a crossover operation. However, these approaches involve an

additional search space for finding the right permutation of the bit ordering that is as large as the original function optimization space. In addition, one of the bit orderings of the parents is randomly chosen as the ordering of the siblings. As suggested in a related work [12], gene positions, instead of bit positions, could also be rearranged to preserve building blocks.

In [9], a 2-level GA is proposed. Each low level gene in the chromosome is tagged with a high level control gene that turns the gene on or off. This 2-level GA approach is very similar to a redundant gene approach in [10] where multiple genotypic bits would vote to turn on or off a phenotypic bit. In [1], [11], multiple populations were maintained and migration was allowed among different sub-populations. The primary goal was to maintain diversity. So long as at least one subpopulation does not converge to the deceptive attractor, there is still a chance that the global optimum can be found.

3 An Evolutionary Mapping Method

The proposed method in this paper is based on a single population GA model although it could be extended easily to use multiple populations. A population of cells instead of chromosomes is maintained. Each cell has two chromosomes, a binary *data chromosome* that stores the genotype for the optimization function and a value-coded *mapping chromosome* that stores bit locations as integers. For example, an initial cell may have two chromosomes as follow:

1	1	0	1	1	1	0	0

0	1	2	3	4	5	6	7

The chromosome to the left is the data chromosome that stores genotypic bits while the mapping chromosome on the right hand side defines the mapping from a genotype to a phenotype. For example, the above mapping chromosome maps the genotype to a phenotype that is exactly the same as the genotype itself. However, a different mapping chromosome (7,6,0,1,2,3,4,5) will map the above data chromosome to a phenotype (0,0,1,1,0,1,1,1), which may be a parameter of an optimization function. In the evolution process, the data chromosomes and mapping chromosomes undergo separate genetic operations with their counter parts in the other cells. The mapping chromosomes also go through additional permutation operations to alter the bit ordering. While the genetic operations for the data chromosomes are traditional 2-point crossover and mutation, different implementations of the genetic operations for the mapping chromosomes may result in significantly different outcomes. If the bit ordering on the mapping chromosome is constantly maintained as a permutation of the bit locations, then the approach will be very similar to the tagged bit approach as in Genitor [1] and Goldberg's work [8]. However, this permutation method by itself is shown not to be very effective against deceptive problems.

This paper suggests a decoupling of the genotype from the mapping process by allowing the mapping chromosomes to cross and mutate freely using integer genetic operators without maintaining a one-to-one mapping between a genotypic bit and a phenotypic bit.

3.1 Genetic Operators

As mentioned earlier, the data chromosomes undergo the traditional 2-point crossover and mutation operations. On the other hand, the permutation operator rearranges bits on the mapping chromosomes by swapping and shifting bits. The crossover operator for the mapping chromosomes is a traditional 2-point integer crossover operator. The mutation operator is an integer mutation operator that randomly switches a gene to one of the bit locations. In addition, a gene replacement operation is also included in the mutation process. Through this gene replacement operation, a gene on the mapping chromosome can be copied to another gene. The whole genetic operation process will break the one-to-one mapping between a genotypic bit and a phenotypic bit. A genotypic bit on the data chromosome may be mapped to more then one bit on the phenotype. For example, given a data chromosome (1,0,0,1,1,1,0,0) and a mapping chromosome (1,4,0,1,2,7,4,5), a phenotype (0,1,1,0,0,0,1,1) can be constructed. Bit locations 3 and 6 on the data chromosomes are actually not used.

The rationales of this approach are: First, this approach will force the data chromosomes to concentrate on exploring and surviving in the genotypic space where genomic materials constantly undergo construction and destruction of schemata. The mapping chromosomes on the other hand, will concentrate on the effort of obtaining an optimal mapping between a genotype and a phenotype. Second, with cells of multiple chromosomes, different mappings can evolve at the same time because each mapping is associated with a particular genotype. On the contrary, some previous approaches evolved only one mapping for the entire population. Third, perhaps the most important reason is that the introduction of non-one-to-one mapping creates the possibility of capturing useful schemata as described below.

3.2 Capturing Schemata

Suppose an order N schema consists of r 0 bits and s 1 bits. The 0-bits together form a subschema of order r called 0-subschema whereas the 1-bits form a subschema of order s called 1-subschema. For example, the schema 1*0*1*0*1 consists of an order-3 1-subschema 1***1***1 and an order-2 0-subschema **0***0**. As the defining bits of a subschema emerge during evolution, the system will try to maintain the same bit values at those locations.

Because the mapping chromosomes are allowed to evolve freely to form non-one-to-one mapping between a genotype and a phenotype, the selection pressure should map one defining bit location to another defining bit location in the same subschema. The result is an explicit formation of a schema. For example, the phenotypic subschema 1***1***1 can still be reconstructed from the genotypic pattern 1***0***0 if the mapping chromosome is (0,1,2,3,0,5,6,0).

Once a subschema is explicit formed, the change of one genotypic bit may trigger the changes of multiple defining bits on the phenotype. The ability to alter multiple bit values greatly shortens the Hamming distance between two distant hyper-planes in the solution space. For example, the distance between 1***1***1 and 0***0***0 can be

reduced to one. Hence the search may be able to get away from a deceptive attractor or go toward a global optimum much faster. In deceptive problems, the global optimum is frequently the complement of the deceptive attractor. Therefore, the explicit formation of 1-subschema or 0-schschema may help the search to jump from a deceptive attractor straight to the global optimum.

The schema theory proposed that schemata of higher order and with longer length are less likely to survive because they are more likely to be destroyed by genetic operators [3]. On the contrary, as shown in the above section, the higher the order of subschema, the greater in reduction of Hamming distance between two distant hyperplanes.

4 Experiments

Two sets of experiments were setup to test the dynamic mapping approach. The first set of experiments is primarily based on the deceptive functions constructed in [1] while the second set is based on functions used in [14], [15].

4.1 First Set of Experiments

The first step in these experiments was to construct some deceptive functions. First, the following algorithm from [1] was applied to construct an order-4 deceptive function:

Step 1: Select a global optimum pattern.
Step 2: Sort all binary patterns by their Hamming distance to the optimum pattern. The group of patterns that have the same distances can be placed in any order within the same group
Step 3: The first pattern is pattern 1, the optimum pattern and the last pattern N will be the deceptive attractor. Assign fitness values according to:
fitness(pattern X) = fitness(pattern X–1) + C
where C is a constant and fitness(pattern 2) is another constant, say, 0.

Using the above algorithms, two deceptive functions f2 and f3 were defined as shown in Table 2. The function f2 was directly obtained from [1] while the function f3 was newly constructed using the above algorithm. Note that the optimum of f3 1001 has equal numbers of 1 and 0.

Next, an "ugly" 40-bit function could be constructed based on the discussion in [1] by spreading the four bits of the parameter of function f2 to the chromosome bit locations 0, i+10, i+20, i+40, respectively, for 10 times where $0 \leq i \leq 9$. For instance, based on the parameter 0101, the function value of

ugly(0000000000 1111111111 0000000000 1111111111)

is 16. Similarly, another "bad" 40-bit function could be constructed using f3. In the experiments, the 40-bit ugly and bad functions were also extended to 60-bit by repeating the spreading the four parameter bits 15 times.

Table 2. Deceptive Functions f2 and f3

f2				f3			
f2(1111)	30	f2(0110)	14	f3(1001)	30	f3(0101)	14
f2(0111)	0	f2(0101)	16	f3(1011)	0	f3(0011)	16
f2(1011)	2	f2(0011)	18	f3(1101)	2	f3(0000)	18
f2(1101)	4	f2(1000)	20	f3(1000)	4	f3(0111)	20
f2(1110)	6	f2(0100)	22	f3(0001)	6	f3(1110)	22
f2(1100)	8	f2(0010)	24	f3(1111)	8	f3(0100)	24
f2(1010)	10	f2(0001)	26	f3(1100)	10	f3(0010)	26
f2(1001)	12	f2(0000)	28	f3(1010)	12	f3(0110)	28

Two additional algorithms were also included for comparison. The first one was simply a regular GA without any special operators. The second algorithm was similar to the tagged bit approach in which the one-to-one genotype-phenotype mapping was maintained. In all experiments, for the data chromosomes, the crossover rate was set to 0.6 and mutation rate was set to 1/(length of chromosome). For the mapping chromosome, the crossover rate was 0.6 and the mutation was set to 0.005, which was about 1/5 of the mutation rate for the data chromosome. The permutation rate was 0.025. Several tests were set up to test the algorithms using the 40-bit ugly function, 60-bit ugly function, 40-bit bad function, and 60-bit bad function. The results are obtained summarized in Table 3.

Table 3. Experimental Results using the ugly and bad Functions

	Traditional GA		Tagged Bits		Dynamic Mapping	
	Percentage of Optimal Runs	Avg. Gen.	Percentage of Optimal Runs	Avg. Gen.	Percentage of Optimal Runs	Avg. Gen.
ugly 40 bits	0%	n/a	0%	n/a	100%	1387
ugly 60 bits	0%	n/a	0%	n/a	100%	2471
bad 40 bits	0%	n/a	0%	n/a	100%	4240
bad 60 bits	0%	n/a	0%	n/a	80%	8931

Each result is based on 5 independent runs. The dynamic mapping approach proposed in this paper successfully found the global optimum in all of the ugly function

tests and the 40-bit bad function tests and it found the global optimum of the 60-bit bad function 80% of the time. The average numbers of generations for the dynamic mapping approach to reach the optimum were listed in the last column. The averages were calculated for successful runs only. The search missed the optimum two times out of ten runs. In addition, the time to find the optimum almost doubled when the size of the problem was increased by 50% from 40 bits to 60 bits. The other two approaches got stuck in the deceptive attractors and never found the optima. The failures of the other approaches simply verify how deceiving the functions were for traditional GAs.

4.2 Second Set of Experiments

The second set of experiment is based on the some of the most difficult GA-Hard problems listed in [14], [15] although other difficult functions were first proposed in [13]. The chosen functions are listed in Table 4 and they are considered GA-Hard because the search spaces are multimodal with large number of local optima. The Rastrigin's function is particularly difficult to optimize because it contains millions of local optima in the interval of consideration. Many search method failed to converge to the global optimum at $x_i = 0$. The Schwelfel's function has a global optimum at $x_i =$ 420.9687. In implementation, the Schwefel's function was modified as 418.9828872721624–f_7 to avoid negative fitness values. The optima of these optimization functions are indeed needles in haystacks.

Table 4. Additional Optimization Functions

Name	Function	Interval
Rastrigin's	$f_6(\vec{x}) = \sum_{1}^{20}[x_i^2 - 10\cos(2\pi x_i) + 10]$	$-5.12 \le x_i \le 5.12$
Schwefel's	$f_7(\vec{x}) = \sum_{1}^{10} - x_i \sin(\sqrt{\lvert x_i \rvert})$	$-500 \le x_i \le 500$

The experiments were set up to compare the dynamic mapping approach and the traditional GA approach. Because both approaches had problems to converge to an exact optimal point in the solution space, additional local hill climbing was incorporated to help the algorithms to perform local search to zoom into the optimum. Note that the optimization functions are defined in the real number domain with infinite precision. Therefore, if $\mathbf{x}^*$ is the optimum point in the solution space and $\mathbf{x}$ is the current search point, the search will stop as soon as $|f(\mathbf{x}) - f(\mathbf{x}^*)| < \varepsilon$, where ε is a small comparison threshold. In the experiments, the threshold ε was set to 10^{-16}. However, using this small threshold, the Rastrigin's problem was not 100% solved. The traditional GA kept wondering in vicinity of the exact optimal point while the dynamic mapping algorithm keeps converging to the optimal point at an ever decreasing rate.

The results of the experiments are summarized below in Table 5. Each of the results is based on data averaged over 5 separate runs. The maximum number of evolu-

tions for the experiments was set to 10,000 and 20,000 generations for Schwefel's function and Rastrigin's function, respectively. For the Schwefel's function, both approaches found the optimum. The fitness values were in the order of 10^{-17} when the evolution was stopped. Again, the dynamic mapping approach performed better than the traditional version. The Rastrigin's function is the hardest among all the functions presented here. If the comparison threshold ε is set to 10^{-16}, no trial runs were considered successful in finding an optimum. However, if the threshold ε is lowered to 10^{-6}, 40% of the dynamic mapping algorithm found solutions that are close enough to the optimum. The traditional GA did not find any optimum even with the lowered threshold.

Table 5. Experimental Results using some GA-Hard Functions

	Traditional GA		Dynamic Mapping	
	Percentage of Optimal Runs	Avg. Gen.	Percentage of Optimal Runs	Avg. Gen.
Schwefel's	100%	2231	100%	1198
Rastrigin's	0% / 0%	n/a	0% / 40%	n/a / 15826

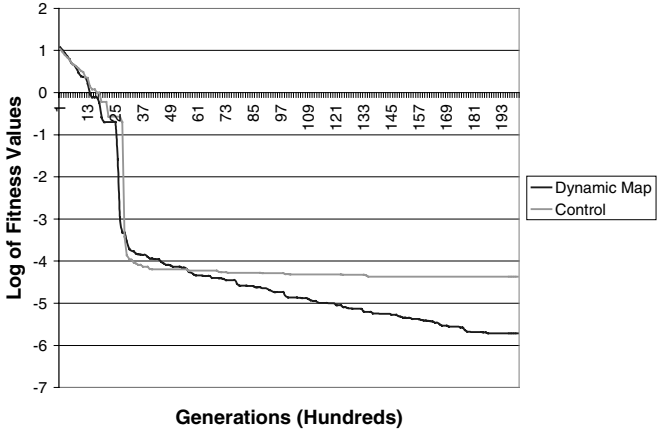

Fig. 1. Fitness Values of the All-Time-Best Cells

The fitness values from the all-time-best cell were averaged and plotted in Figure 1. The traditional GA converged very quickly in the beginning and at times produced better solutions earlier than the dynamic mapping approach; however, it got stuck in some local optima. The search by the dynamic mapping approach continued until the cut-off point at 20,000 generations. Since we did not use any elitist approach in any experiment, the all-time-best cells were not reintroduced into the populations. How-

ever, the current best cells in each generation also behaved similarly as shown in Figure 2.

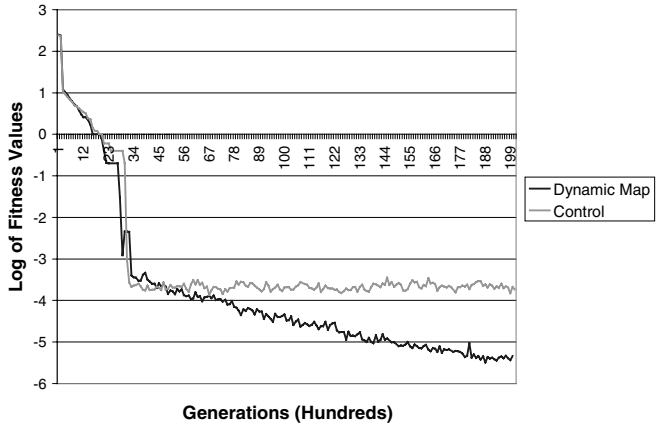

Fig. 2. Fitness Values of the Current Best Chromosomes

4.3 Population Diversity

The final populations of the above experiments were also examined. The following figures, Figures 3 to 5, are population dumps for the 40-bit ugly experiments. The populations are shown as matrices with 256 columns and 40 rows. A column in a matrix is a 40-bit chromosome and there are 256 chromosomes in a population. A 0-bit is shown as a black dot whereas a 1-bit is shown as a white dot.

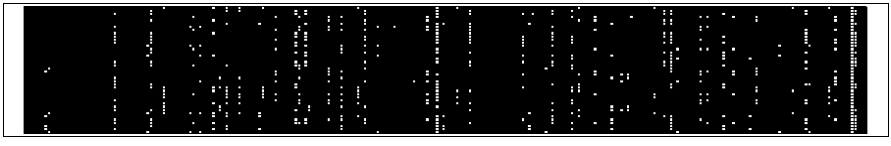

Fig. 3. A Phenotypic Population of the Dynamic Mapping Approach

Figure 3 shows the final phenotypic population for an experiment of the dynamic mapping approach. A perfect solution with all 1's resides near the far right. The rest of the population consists of many local optimal solutions with all 0's since they were attracted to the deceptive local optima.

The corresponding genotypic population as shown in Figure 4 is quite different from the phenotypic population. There are many different individuals and the diversity in the genotypic population prevents the population from converging prematurely. However, some bit locations do consist of predominantly the same bit values across

almost the whole population and such locations may be locations of defining bits of some schemata.

As a comparison, a final genotypic population from the control experiment is shown in Figure 5. Since the control experiments used a canonical GA that got stuck in the deceptive local optima, the population is much less diverse than its counter part from the dynamic mapping experiments.

Fig. 4. A Genotypic Population of the Dynamic Mapping Approach

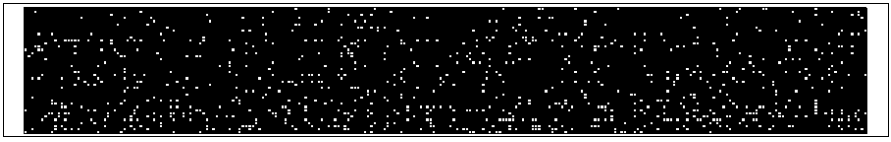

Fig. 5. A Genotypic Population of the Control Experiment

5 Conclusions

The proposed algorithm is every effective in solving some of the deception problems. It is also rather effective in solving some of the GA-Hard problems listed in [14], [15]. The dynamic mapping genetic algorithm is a novel approach that utilizes multiple chromosomes in a single cell for mating with another cell within a single population. The mapping from genotype-to-phenotype is explicitly evolved and maintained. Defining bits of 0-subschema and 1-subschema are supposed to be explicitly captured and stored in the mapping chromosomes. The direct accessibility to such subschema allows fast navigation to complementary hyper-planes and the Hamming distance between two such hyper-planes is greatly shortened to one. As a comparison, the Dual Genetic Algorithm approach in [16], [17], a meta-gene is used to flip gene values to their complements and hence reducing the number of steps needed to transform a chromosome to its genetic opposite.

Although this initial study has shown good potential in solving some GA-Hard problems, more study is needed to better understand and analyze the inner working of the algorithm. The 0/1-subschemata are also limited because they consist of defining bits of the same values. More research is needed to define a structure to capture the whole schema effectively.

References

1. Whitley, L. D.: Fundamental principles of deception in genetic search. In: Rawlins, G., (ed.): Foundations of Genetic Algorithms. Morgan Kaufmann, San Mateo, CA, (1991) 221-241
2. Goldberg, D.: Genetic Algorithms and Walsh Functions: Part II, Deception and its Analysis. Complex Systems 3 (1989) 153-171
3. Holland, J.H.: Adaptation in Natural and Artificial Systems. Ann Arbor, MI: University of Michigan Press (1975).
4. Mathias K., Whitley, L.D.: Transforming the search space with gray coding. In Proc. IEEE Int'l Conference on Evolutionary Computation (1994) 513-518
5. Yokose, Y., Cingoski, V., Kaneda K., and Yamashita, H.: Performance Comparison Between Gray Coded and Binary Coded Genetic Algorithms for Inverse Shape Optimization of Magnetic Devices, Applied Electromagnetics, (2000) 115-120
6. Wolfram, Gray Code, http://mathworld.wolfram.com/GrayCode.html
7. Goldberg, D.: Genetic Algorithms and Walsh Functions: Part I, A Gentle Introduction. Complex Systems 3 (1989) 129-152
8. Goldberg, D., Bridges, C. An Analysis of a Reordering Operator on a GA-Hard Problem. Biological Cybernetics 62 (1990) 397-405
9. Dasgupta, D.: Handling deceptive problems using a different genetic search. In Proceedings of the First IEEE Conference on Evolutionary Computation (1994) 807-811
10. Shackleton M., Shipman, R., and Ebner, M.: An investigation of redundant genotype-phenotype mappings and their role in evolutionary search. In Proceedings of Congress on Evolutionary Computation (2000) 493-500
11. Whitley, D., Rana, S., Heckendorn, R.B.: Exploiting Separability in Search: The Island Model Genetic Algorithm. Journal of Computing and Information Technology, v. 7, n. 1, (1999) 33-47 (Special Issue on Evolutionary Computing)
12. Sehitoglu, O.T., Ucoluk, G.: A building block favoring reordering method for gene positions in genetic algorithms. In Spector, L., Goodman, E.D., Wu, A. et.al., (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (2001) 571-575
13. De Jong, K.A.: An Analysis of the Behavior of a Class of Genetic Adaptive Systems, Ph.D. Thesis, University of Michigan (1975)
14. Digalakis J., Konstantinos, M.: An Experimental study of Benchmarking Functions for Evolutionary Algorithms. International Journal of Computer Mathemathics, Vol. 79, (2002) 403-416
15. Salomon. R.: Reevaluating Genetic Algorithm Performance under Coordinate Rotation of Benchmark Functions. BioSystems vol. 39, Elsevier Science (1995) 263-278
16. Collard P., Aurand J.-P.: DGA: an efficient Genetic Algorithm. In Proceedings of the 11th European Conference on Artificial Intelligence (ECAI'94) (1994) 487-492
17. Collard P., Clergue M., Defoin P.M.: Artificial Evolution: In Proceedings of the Fourth European Conference (AE'99), Lecture Notes in Computer Sciences 1829, Springer-Verlag Ed. (2000) 254-265