Coevolutionary Genetic Algorithms for Solving Dynamic Constraint Satisfaction Problems

Hisashi Handa[+] **Osamu Katai**[*] **Tadataka Konishi**[+] **Mitsuru Baba**[+]

+Department of Information Technology
Fuculty of Engineering
Okayama University
Tsushima-naka 3-1-1, Okayama 700-8530, JAPAN
{handa, konishi, baba}@sdc.it.okayama-u.ac.jp
+81-86-251-8250

∗Department of System Science
Graduate School of Informatics
Kyoto University
Yoshida-Honmachi, Kyoto 606-8501, JAPAN
katai@i.kyoto-u.ac.jp
+81-75-753-5201

Abstract

In this paper, we discuss the adaptability of Coevolutionary Genetic Algorithms on dynamic environments. Our CGA consists of two populations: solution-level one and schema-level one. The solution-level population searches for the good solution in a given problem. The schema-level population searches for the good schemata in the former population. Our CGA performs effectively by exchanging genetic information between these populations. Also, we define Dynamic Constraint Satisfaction Problems as such dynamic environments. General CSPs are defined by two stochastic parameters: density and tightness, then, Dynamic CSPs are defined as a sequence of static constraint networks of General CSPs. Computational results on DCSPs confirm us the effectiveness of our approach.

1 Introduction

In this paper, we investigate the behavior of Genetic Algorithms on dynamic environments. We expect that GAs with rich diversity of genetic information in the GA population perform well in such environment. To keep the diversity effectively, we adopt Coevolutionary Genetic Algorithm proposed in (Handa *et al.*, 1997). Our Coevolutionary Genetic Algorithm (CGA) consists of two GA populations, i.e., H-GA and P-GA: The H-GA searches for good solution by means of an ordinal Genetic Algorithm, while the P-GA searches for good schemata contained in genetic information in the H-GA's population. Two genetic operators between these populations, called superposition and transcription, serve as a way of the propagation of genetic information. CGA outperforms standard Genetic Algorithms resulting from their symbiotic coevolution (Handa *et al.*, 1998, 1997, 1997b).

Also, we adopt the notions of Constraint Satisfaction Problems (CSPs) (Tsang, 1992) to construct dynamic environments. The notions of CSPs have been used to solve many practical problems, and have been studied by many researchers. We employ General CSPs, which is defined by two stochastic parameters, to make up instances of CSPs. Moreover, we introduce Dynamic CSPs as a sequence of static instances of General CSPs. By using a framework of DCSPs, there are several benefits for various fields, e.g., decision making problems in politics, economics, game playing and so on. If such problems are only formulated in static CSPs, we have to formulate all about simulation environment including internal states in the other agents or players. By using the framework, however, we only formulate a DCSP of own status and define effects from the other agents' policy or action. Also, some huge-scale problems may be easily solved by using the framework of DCSPs. The huge-scale problems are divided into several partial problems. Each of several partial problems is formulated as sub-DCSPs, and then defines constraints against adjacency but external nodes. Like as the islands-model (Goldberg, 1989), the best solutions in the previous interval are used as candidates of the external nodes. That is, we only concentrate on solving the part of a huge-scale problem at each of sub-DCSPs.

Related works are described as follows: Dynamic Constraint Satisfaction Problems in believe maintenance are discussed by Dechter and Dechter (Dechter, 1989). We adopt almost same formulation of the DC-

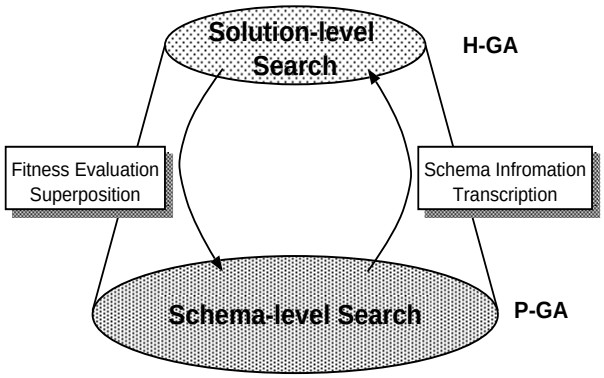


Figure 1: Process of Coevolutionary Genetic Algorithm

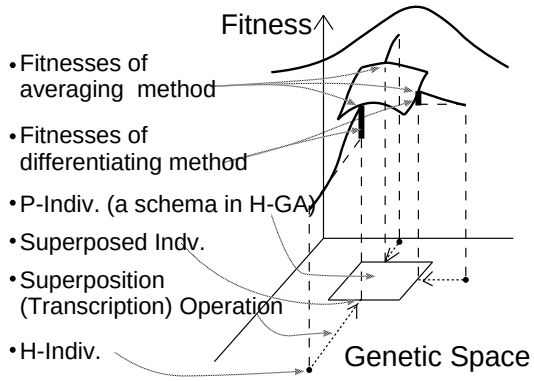


Figure 2: Fitness Evaluation of P-GA in Genetic Space of H-GA

SPs as Dechter's. Coevolutionary approaches have been studied by many researchers (Hills, 1992; Potter, 1995; Barbosa 1997). Especially, coevolutionary approach for solving Constraint Satisfaction Problems is proposed by Paredis (Paredis 1993, 1996). He used two populations where an inverse fitness interaction, more precisely, the predator-prey relationship, between these populations is set. Also, schemata-oriented search methods in evolutionary computation have been adopted in several problem solving methods such as Cultural Algorithms and Stochastic Schemata Exploiter (Reynolds, 1996; Aizawa, 1994). In Cultural Algorithm, usual GA model is associated with a belief space that is similar to the schema space and is used to promote directed evolution of individuals in the GA model. Our method is similar to Cultural Algorithm in the sense that both methods use additional mechanisms to promote the evolution of usual GA. However, the specificity of our approach is to use a coevolutionary mechanism.

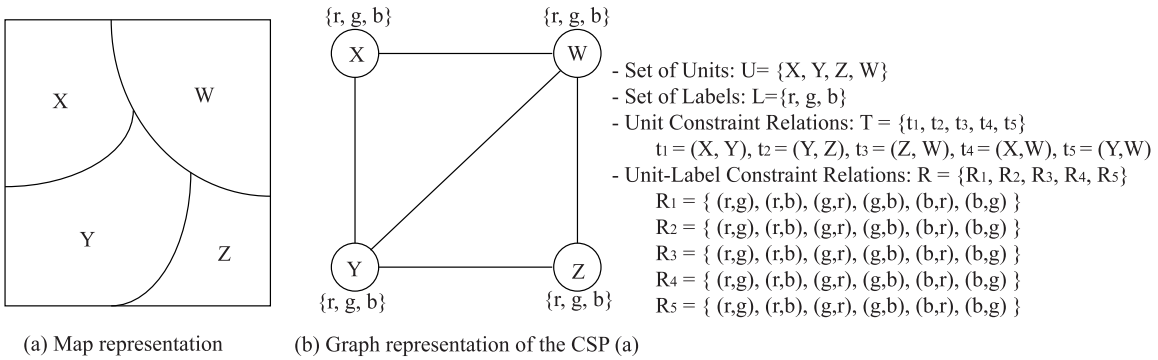
In next section, we explain our Coevolutionary Genetic Algorithms. Then, we introduce static and dynamic Constraint Satisfaction Problems in Section 3. In Section 4, several computer simulations are examined and confirm us effectiveness of our approaches, and finally, this paper is concluded.

2 Coevolutionary Genetic Algorithm

We adopt Coevolutionary Genetic Algorithm to solve Dynamic CSPs. As depicted in Figure 1, we have two GA populations: **H-GA** and **P-GA**. The H-GA is a traditional GA, in other words, it searches for good solutions in the given problem. In this paper, as the traditional GA, we use SGA including tournament selection, two point crossover and normal mutation. The P-GA searches for the good schemata in the H-GA. Each individual in P-GA consists of alleles of H-indiv. and "$*$", which are representing a schema in the H-GA. As depicted in the figure, two genetic operators, i.e., **superposition** and **transcription**, play the role to communicate (propagate) genetic information between the H-GA and the P-GA. The superposition operator copies the genetic information of a P-indiv., except for "don't care symbol" (denoted by "$*$" in the figure), onto one of H-Indiv.'s in order to calculate the fitness of the P-Indiv, where **H-Indiv.** and **P-Indiv.** denote an individual of H-GA population and an individual of P-GA population, respectively. The transcription operator serves as a mean of transmitting effective genetic information searched by the P-GA to the H-GA. For further details see also (Handa, 1997). Following subsection describes some explanation of an important part of our algorithm, i.e., fitness evaluation of P-Indiv.

2.1 Fitness Evaluation of P-GA

Generally speaking, the fitness value of a schema is calculated as the average fitness value of all individuals belonging to the schema. It is difficult, however, to calculate the fitness values of all individuals when the order of schema is small. So, the average value of schema is set to the average value of "sampled" individuals belonging to the schema. Also, P-GA searches for *useful* schemata in H-GA. Here, the useful schemata in H-GA may be defined as follows: (1) undiscovered useful schemata or simply (2) useful schemata, i.e., those with high average fitness values. Thus, we introduce two manner of fitness evaluation: Differentiating method and Averaging method.



(a) Map representation (b) Graph representation of the CSP (a)

Figure 3: An example of CSP: graph coloring problem

Differentiating method
If a schema information discovered by P-GA is already discovered by H-GA, the H-GA will receive no effective information from this "discovery" by P- GA in this method. Hence, we let P-GA to search for "undiscovered" useful schemata in H-GA, and the fitness evaluation of a P-Indiv. is given as follows: First, the fitness value F_j of a P-indiv., say, j is calculated in the following way: The superposing (superposition) operation of each P-indiv. onto H-indiv.'s is carried out n times.

(1) First, n H-indiv.'s to be superposed by P-indiv. j are randomly selected.

(2) These selected H-indiv.'s are denoted as i_1, ..., i_n, and the resultant superposed H-indiv.'s are denoted as $\tilde{i}_1$, ..., $\tilde{i}_n$.

(3) Then, to calculate the fitness value of P-indiv. j, the effect of each of the superposition operations is evaluated the contribution of the superposition operation to each H-indiv. defined as follows:

$$F_j = \sum_{k=1}^{n} \max(0, f_{\tilde{i}_k} - f_{i_k}) \qquad (k = 1, \ldots, n) \ .$$

Thick lines in Figure 5 denote the difference between the fitness values of the original H-indiv.'s and those of the superposed H-indiv.'s, that is, "positive contribution" of this superposition operation. If the difference is negative, then the contribution of this operation is regard to be 0.

Averaging method
In our CGA, the P-GA evaluates a subspace in the genetic space of H-GA. Step **(1)** and **(2)** of the differentiating method also apply to the averaging method. However, step **(3)** is altered as follows:

(3') Similar to **(3)**, to calculate the fitness of P-Indiv., the fitness function for the superposition operation is defined by the "result" of the operation, i.e., we set

$$F_j = \sum_{k=1}^{n} f_{\tilde{i}_k} \qquad (k = 1, \ldots, n) \ .$$

In this method, the average evaluations of he H-GA's genetic subspace are carried out.

3 Constraint Satisfaction Problems

3.1 Formalization of Constraint Satisfaction Problems

Constraint Satisfaction Problems (CSPs) are a class of problems consisted of variables and constraints on the variables (Tsang, 1993; Marchiorl, 1997). Especially, a class of the CSPs such that each constraint in the problems is related only to two variables are called binary CSPs. In this paper, we treat a class of discrete binary CSPs, where discrete means that each variables in given problems are associated to a finite set of discrete labels. An example of the graph coloring problem (Minton, 1994), one of binary CSPs, which is one of the benchmark problems in CSP is delineated in Figure 3. As depicted in the figure, CSPs are defined by (U, L, T, R): U, L, T and R denote set of units, set of labels, unit constraint relations and unit-label constraint relations, respectively. In this 3-coloring problem, i.e., coloring with three colors, r, g and b, for instance, the set U of units consists of the nodes in the graph of a given problem. The elements in the set L of labels denote three colors that should be coloring. The unit constraint relations T mean as the edge in the graph of the given problem. The unit-label constraint relations R are set of 2-compound labels that the constraints are existing. To solve CSPs is to search for

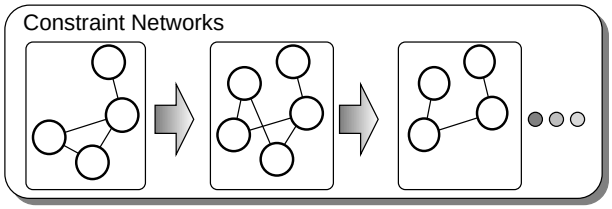


Figure 4: An instance of Dynamic Constraint Satisfaction Problems: At every interval C_t, the nature of instance is randomly changed.

solutions such that no constraints are violated, where the graph representation of CSP in Figure 3(b) called Constraint Network is often used.

We introduce two indices representing the characteristics of problems: tightness and density. The tightness T_{ij} of an arc ij denotes the proportion of existing constraint between two variables i and j, that is,

$$T_{ij} = \frac{\text{the number of all constraints on an arc } ij}{\text{the number of all compound-labels on an arc } ij}.$$

Further, the tightness of a problem is the average value of T_{ij} over all arcs. The density D of a problem indicates the proportion of constraint that actually exists between any pair of variables, i.e.,

$$D = \frac{\text{the number of all constraint-relations in a problem}}{\text{the number of all pairs of variables in a problem}}.$$

3.2 Dynamic Constraint Satisfaction Problems

In this paper, Dynamic Constraint Satisfaction Problems are defined as a sequence of instances of static Constraint Networks. That is, at intervals of constant C_t, the nature of instances, such as the number of variables, the topology of unit constraint relations or unit-label constraint relation, and so on, is changed as delineated in Figure 4. By using this definition, we can improve the ability to represent instances associated to practical problems.

Hence, there are several ways to change the property of instances: As for changing the shape of constraint networks, add a node, delete a node, increase the size of a domain, and decrease the size of a domain is enumerated. Also, as for changing the topology of constraints in constraint networks, there are modifying constraint relations and modifying compound-constraints. In practice, all changes of constraint networks are represented by combining these ways. In the case of applying GAs to solve CSPs, we have to change the coding method in GAs according to the changes of the shape of constraint networks. In this paper, thus, we adopt only the modifying constraint relations as way to change the property of the constraint network.

Furthermore, when we try to solve CSPs by using GAs, there are two categories of changing property as follows:

KNOWN GAs can observe which constraint relations are changed. In this case, for changed variables, new alleles are inserted to corresponding gene locus.

UNKNOWN GAs cannot observe which constraint relations are changed. Note that we incorporate the case, such that GAs can observe which variables are changed but cannot know which variables are affected by constraint propagation from changed variable.

4 Experimental Results

In this section, several experimental results on Dynamic General Constraint Satisfaction Problems described below are examined. As described in section 3, Dynamic General CSPs is represented by a sequence of the constraint networks associated to General CSPs. At every interval C_t, the nature of the constraint networks is changed. In this paper, the interval C_t is set to be 200000 times of fitness evaluations. The general CSPs are randomly generated as follows: First, specify the *tightness* and *density* in the sense in section 3. Next, for all combination of two variables, decide whether unit constraint relation is set to be each of the pairs of variables by taking account of the value of *density*. Finally, for all unit constraint relation, the number of the unit-label constraint relations is set to be directly proportional to the *tightness*.

The Dynamic General CSPs are set such that both of the number of variables and the size of domain are set to be 10. The fitness function is defined as $1/(1 + \text{the number of violated constraints})$. In all experiments, the GA parameters for the SGA and the H-GA are set to be of the same value, the probability P_c of crossover, the probability P_m of mutation and the numbers of the elitist are set to be 0.8, 0.01 and 5, respectively. Those for the P-GA are set to be $P_c = 0.8$ and $P_m = 0.05$. Further, don't care symbol in the P-GA is generated with a high probability. Also, the number of runs for each of the tuple (*tightness*, *density*) is set to be 100.

Figure 5 shows the experimental results on Dynamic General CSP, (*density*, *tightness*, *interval*) =

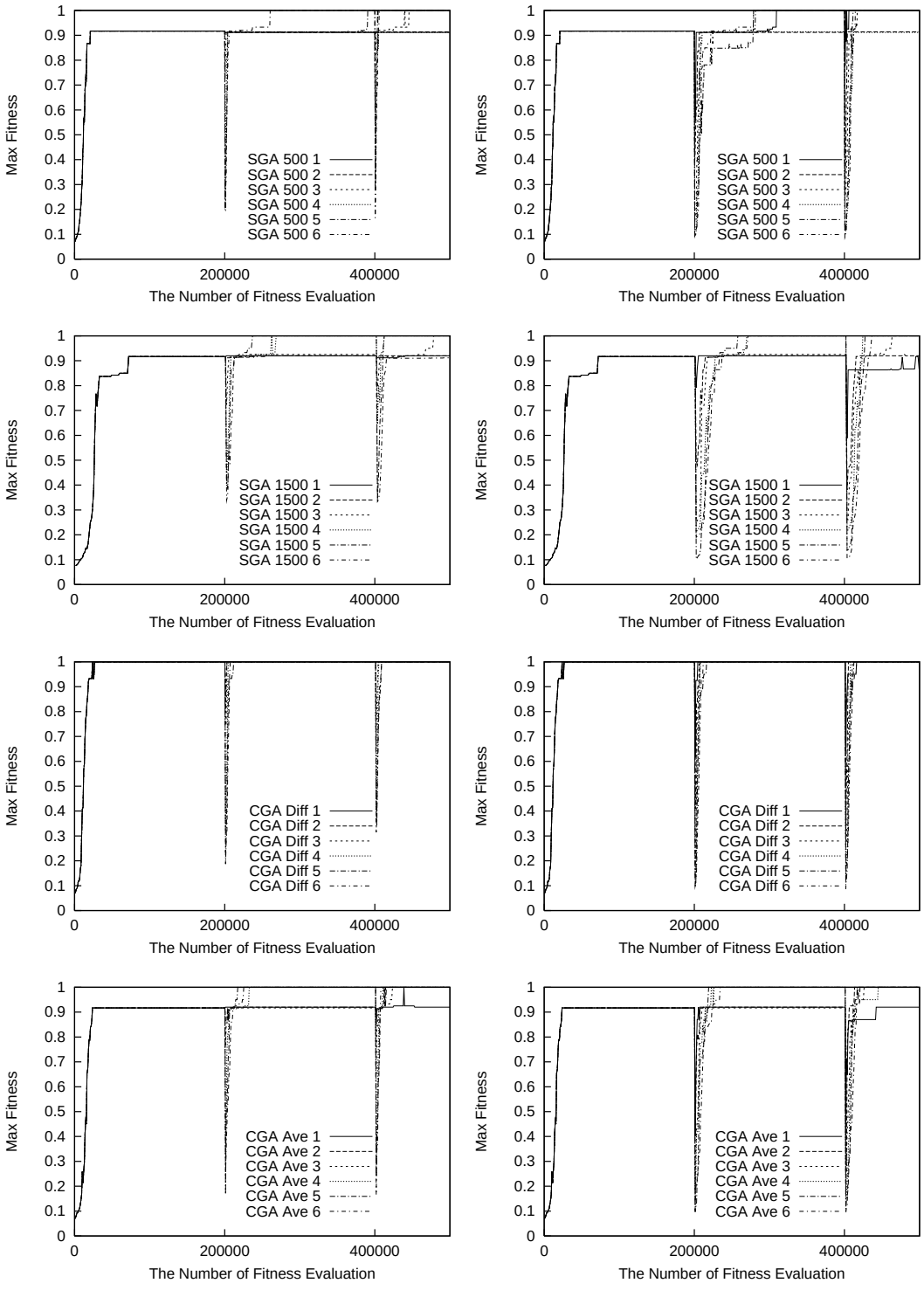


Figure 5: Experimental Results: Dynamic General Constraint Satisfaction Problems (*density*, *tightness*, *interval*) = (70, 30, 200000); LEFT COLUMN:*KNOWN*, RIGHT COLUMN:*UNKNOWN*; 1st ROW:SGA–population size is 500, 2nd ROW:SGA–population size is 1500, 3rd ROW:CGA (Differentiating)–H-GA's population size is 400 and P-GA's population size is 100, 4th ROW:CGA (Averaging).

$(70, 30, 200000)$. We examined comparisons on DGCSPs for a variety of the couple of *tightness* and *density*. For all couples of them, CGA using differentiating method outperforms SGA or performs roughly the same as SGA. This figure is one of typical results, where the horizontal axis, the vertical axis, and each lines in the all graphs denote the number of fitness evaluations, the Max fitness value in the population, the Max fitness value for a variety of the proportion of changed nodes. The left column and the right column in this figure denote *KNOWN* and *UNKNOWN* problems, respectively. The first and second rows in the figure denote SGA such that the population size is 500 and 1500, respectively. The third row denotes CGA such that H-GA's is 400, P-GA's is 100, and fitness evaluation of P-GA is differentiating method. The fourth row is same as the third one, except for that fitness evaluation of P-GA is averaging method.

As depicted in this figure, CGA using differentiating method can search for new satisfiable solutions after environmental changes quickly. Moreover, CGA using averaging method converges into local optima prematurely, after environmental changes, the CGA with averaging method discovers satisfiable solutions almost every experiments. On the other hand, SGA cannot search for new satisfiable solutions effectively, if GAs don't know when environmental changes are occurred. It seems that such recovering ability of CGA results from P-GA, that is, P-GA serve as a means of the diversity preserving.

5 Conclusion

In this paper, we examined the behavior of GAs on dynamic environments. Also, we defined the framework of the Dynamic Constraint Satisfaction Problems. The experimental results carried out in the section 4 confirm us the effectiveness of our approach, i.e., Coevolutionary Genetic Algorithms.

References

A. N. Aizawa (1994), Evolving SSE: A Stochastic Schemata Exploiter, *Proceedings of the 1st International Conference on Evolutionary Computation*, 525-529.

H. J. C. Barbosa (1997), A coevolutionary genetic algorithm for a game approach to structural optimization, *Proceedings of the Seventh International Conference of Genetic Algorithm*, 545-552.

R. Dechter and A. Dechter (1989), Belief Maintenace in Dynamic Constraint Networks, *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, Vol. I, 37-42.

D. E. Goldberg (1989), *Genetic Algorithm in Search, Optimization, and Machine Learning*, Addison-Wesley.

H. Handa, N. Baba, O. Katai T. Sawaragi and T. Horiuchi (1997), Genetic Algorithm involving Coevolution Mechanism to Search for Effective Genetic Information, *Proceedings of the 4th International Conference on Evolutionary Computation*, 709-714.

H. Handa, O. Katai, T. Sawaragi and N. Baba (1998), Solving Constraint Satisfaction Problems by Using Coevolutionary Genetic Algorithms, *Proceedings of the 5th Interantional Conference on Evolutionary Computation*, 21-26.

W. D. Hills (1992), Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure, *Artificial Life II*, 313-324.

E. Marchiorl (1997), Combining Constraint Processing and Genetic Algorithms for Constraint Satisfaction Problems, *Proceedings of the Seventh International Conference of Genetic Algorithm*, 330-337.

S. Minton, M. D. Johnston, A. B. Philips and P. Larid (1994). Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, *Constraint-Based Reasoning*, 161-206, E. C. Freuder and A. K. Mackworth Editors, The MIT Press.

I. C. Parmce and H. D. Vekeria (1997), Co-operative Evolutionary Strategies for Single Component Design, *Proceedings of the Seventh International Conference of Genetic Algorithm*, 529-536.

M. A. Potter, K. A. De Jong and J. J. Grefenstette (1995), A Coevolutionary Approach to Learning Sequential Decision Rules, *Proceedings of the Sixth International Conference of Genetic Algorithm*, 366-372.

J. Paredis (1993), Genetic State-Space Search for Constraint Optimization Problems, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Vol. II, 967-972.

J. Paredis (1994), Co-evolutionary Constraint Satisfaction, *Proceedings of Parallel Problem Solving from Nature III*, 46-55.

R. G. Reynolds and C. Chung (1996), A Self-adaptive Approach to Representation Shifts in Cultural Algorithms, *Proceedings of the 3rd International Conference on Evolutionary Computation*, 94-99.

E. Tsang (1993), *Foundation of Constraint Satisfaction*, Academic Press.