A Brief Review of Penalty Methods in Genetic Algorithms for Optimization

Mitsuo Gen **Runwei Cheng**
Department of Industrial and Systems Engineering
Ashikaga Institute of Technology, Ashikaga 326, Japan
Email: gen@genlab.ashitech.ac.jp

Abstract : Penalty technique perhaps is the most common technique used in the genetic algorithms for constrained optimization problems. In recent years, several techniques have been proposed in the area of evolutionary computation. However, there is no general guideline on designing penalty function and constructing an efficient penalty function is quite problem-dependent. The purpose of the paper is to give a tutorial survey of recent works on penalty techniques used in genetic algorithms and to give a better classification on existing works, which may be helpful for revealing the intrinsic relationship among them and for providing some hints for further studies on penalty techniques.

Key words : Genetic algorithms, penalty techniques, constrained optimization.

1 Introduction

Genetic algorithms have been receiving an increasing attention as a novel optimization techniques for constrained optimization problems [1]. The central problem of applying genetic algorithms to constrained optimization is how to handle constraints because genetic operators used to manipulate the chromosomes often yield infeasible offspring. Recently, several techniques have been proposed to handle constraints in genetic algorithms. For recent survey papers on the problem we refer to Michalewicz [2, 3]. The existing techniques can be roughly classified as follows:

- *rejecting strategy*
- *repairing strategy*
- *modifying genetic operators strategy*
- *penalizing strategy*

Each of these strategies have advantages and disadvantages. *Rejecting strategy* discards all infeasible chromosomes throughout whole evolutionary process. *Repairing strategy* depends on the existence of a deterministic repair procedure to converting an infeasible offspring into a feasible one. With the *modifying genetic operators strategy*, one has to invent problem-specific representation and specialized genetic operators to maintain the feasibility of chromosomes. These strategies have the advantage that they never generate infeasible solutions but have the disadvantage that they consider no points outside the feasible regions. For highly constrained problem, infeasible solution may take a relatively big portion in population. Glover has suggested that constraint management techniques allowing movement through infeasible regions of the search space tend to yield more rapid optimization and produce better final solutions than do approaches limiting search trajectories only to feasible regions of the search space [4]. The *penalizing strategy* is such a kind of technique proposed to consider infeasible solutions in genetic search.

Penalty technique perhaps is the most common technique used in the genetic algorithms for constrained optimization problems. In essential, this technique transforms the constrained problem into an unconstrained problem by penalizing infeasible solutions. The main issue of penalty strategy is how to design the penalty function $p(\boldsymbol{x})$ which can effectively guide genetic search towards to promising area of solution space. Several techniques have been proposed in the area of evolutionary computation. However, there is no general guideline on designing penalty function. Constructing an efficient penalty function is quite problem-dependent. The purpose of the paper is to give a brief review of major recent works on penalty techniques used in genetic algorithms and also to give a better classification on existing works, which may be helpful for revealing the intrinsic relationship and providing some hints for further studies on penalty techniques.

2 Nonlinear Programming

Nonlinear programming deals with the problem of optimizing an objective function in the presence of equality and inequality constraints. Nonlinear programming is an extremely important tool used in almost every area of engineering, operations research, and mathematics because many practical

problems cannot be successfully modeled as a linear program. The general nonlinear programming model can be written as follows:

$$\max \quad f(\boldsymbol{x}) \tag{1}$$

$$\text{s. t.} \quad g_j(\boldsymbol{x}) \leq 0, \quad j = 1, 2, \cdots, q \tag{2}$$

$$h_j(\boldsymbol{x}) = 0, \quad j = 1, 2, \cdots, m \tag{3}$$

$$\boldsymbol{x} \in X \tag{4}$$

where $f(\boldsymbol{x})$ is the *objective function*, constraints (2) is the *inequality constraint*, and constraints (3) is the *equality constraint*. The set X is the *domain constraint*, which might typically include lower and upper bounds on the variables. A vector $\boldsymbol{x} \in X$ satisfying all the constraints is called a *feasible solution* to the problem. The collection of all such solutions forms the *feasible region*. The nonlinear programming problem then is to find a feasible point $\bar{\boldsymbol{x}}$ such that $f(\boldsymbol{x}) \leq f(\bar{\boldsymbol{x}})$ for each feasible point $\boldsymbol{x}$. Such a point is called an *optimal solution*.

In general, solution space contains two parts: feasible area and infeasible area. Unlike conventional solution methods for nonlinear programming, we do not make any assumption about these subspaces; in particular, they need not be convex or be connected as shown in Figure 1. Penalizing infeasible chromosomes is far from trivial. From the figure we can know that infeasible solution b is much near to optima a than infeasible solution d and feasible solution c. We may hope to give less penalty to b than d even though it is a little far from feasible area than d. We also can believe that b contains much more information about optima than c even though it is infeasible. However, we have no any priori knowledge about optima, so generally it is very hard for us to judge which one is better than others.

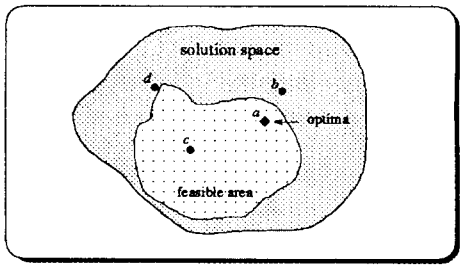

Figure 1: Solution space: feasible area and infeasible area

3 Penalty function

3.1 Evaluation function with penalty term

Penalty techniques transform the constrained problem into an unconstrained problem by penalizing infeasible solutions. In general, there are two possible ways to construct the evaluation function with penalty term. One is to take the addition form expressed as follows:

$$eval(\boldsymbol{x}) = f(\boldsymbol{x}) \; + \; p(\boldsymbol{x}) \tag{5}$$

where $\boldsymbol{x}$ represents a chromosome, $f(\boldsymbol{x})$ the objective function of problem, and $p(\boldsymbol{x})$ the penalty term. For maximization problems, we usually require that,

$$\begin{cases} p(\boldsymbol{x}) = 0; & \text{if } \boldsymbol{x} \text{ is feasible} \\ p(\boldsymbol{x}) < 0; & \text{otherwise} \end{cases} \tag{6}$$

Let $|p(\boldsymbol{x})|_{\max}$ and $|f(\boldsymbol{x})|_{\min}$ be the maximum of $|p(\boldsymbol{x})|$ and minimum of $|f(\boldsymbol{x})|$ among current population, respectively. We also require that

$$|p(\boldsymbol{x})|_{\max} \leq |f(\boldsymbol{x})|_{\min} \tag{7}$$

to avoid negative fitness value. For minimization problems, we usually require that,

$$\begin{cases} p(\boldsymbol{x}) = 0; & \text{if } \boldsymbol{x} \text{ is feasible} \\ p(\boldsymbol{x}) > 0; & \text{otherwise} \end{cases} \tag{8}$$

The second way is to take the multiplication form expressed as follows:

$$eval(\boldsymbol{x}) = f(\boldsymbol{x})\, p(\boldsymbol{x}) \tag{9}$$

In this case, for maximization problems we require that,

$$\begin{cases} p(\boldsymbol{x}) = 1; & \text{if } \boldsymbol{x} \text{ is feasible} \\ 0 \leq p(\boldsymbol{x}) < 1; & \text{otherwise} \end{cases} \tag{10}$$

and for minimization problems, we require that,

$$\begin{cases} p(\boldsymbol{x}) = 1; & \text{if } \boldsymbol{x} \text{ is feasible} \\ p(\boldsymbol{x}) > 1; & \text{otherwise} \end{cases} \tag{11}$$

Note that for the minimization problems, the fitter chromosome have the lower value of $eval(\boldsymbol{x})$. For some selection methods, it is required to be transformed into fitness values in order to ensure that the fitter one has the larger values.

The central issue is how to design the penalty function $p(\boldsymbol{x})$ which can effectively guide genetic search towards to promising area of solution space. There is no general guideline on designing penalty function and constructing an efficient penalty function is quite problem-dependent. Richardson and Palmer suggested that [5]:

(1) *Penalties which are functions of the distance from feasibility are better performers than those which are merely functions of the number of violated constraints.*

(2) *For a problem having few constraints and few full solutions, penalties which are solely functions of the number of violated constraints are not likely to find solutions.*

(3) *Good penalty functions can be constructed from two quantities, the maximum completion cost and the expected completion cost.*

(4) *Penalties should be close to the expected completion cost, but should not frequently fall below the solutions found. When penalty often underestimates the completion cost, then the search may not find a solution.*

The relationship between infeasible chromosome and the feasible part of the search space plays a significant role in penalizing infeasible chromosomes: the penalty value is corresponding to the 'amount' of its infeasibility under some measurement.

3.2 Classification on penalty function

Several handling infeasibility techniques have been proposed in the area of evolutionary computation. In general, we can classify them into two classes:

- *constant penalty*
- *variable penalty*

The constant penalty approach is known as less effective for complex problem and most recent research works put attention on the variable penalty.

In general, variable penalty approach contains two components:

- *variable penalty ratio*
- *penalty amount for the violation of constraints*

The variable penalty ratio can be adjusted according to

- *the degree of violation of constraints*
- *the iteration number of genetic algorithms*

The first approach increases the penalty pressure as the violation becomes severe, which leads to the class of *static penalty*, and the second approach increases the penalty pressure along with the growing of evolutionary process, which leads to the class of *dynamic penalty* as discussed by Michalewicz [3].

Essentially, penalty is a function of the distance from feasible area. This can be given as the following three possible ways:

- *the function of absolute distance of a single infeasible solution*
- *the function of relative distance of all infeasible solutions in current population*
- *the function of adaptive penalty term*

Most methods take the first approach. For highly constrained problem, the ratio of infeasible to feasible solutions is relatively high at each generation. In such cases, the second and third approach is hopeful to make a good balance between the preservation of information and the pressure for infeasibility.

The penalty approaches can be further distinguished as

- *problem-dependent*
- *problem-independent*

Most penalties techniques belong to the class of problem-dependent approach.

The penalty approaches also can be distinguished as

- *with parameter*
- *without parameter*

Most penalties techniques belong to the class of parameterized approach. It seems that the parameterized penalty functions incline to problem-dependent one.

3.3 Homaifar, Qi and Lai's method

Homaifar *et al.* have considered following nonlinear programming problem [6]:

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) \\ \text{s. t.} \quad & g_j(\boldsymbol{x}) \geq 0, \quad j = 1, 2, \cdots, m \end{aligned}$$

and take the addition form of evaluation function.

$$eval(\boldsymbol{x}) = f(\boldsymbol{x}) \; + \; p(\boldsymbol{x})$$

The penalty function is constructed with two components: (1) variable penalty factor and (2) penalty for the violation of constraints as follows:

$$p(\boldsymbol{x}) = \begin{cases} 0; & \text{if } \boldsymbol{x} \text{ is feasible} \\ \sum_{j=1}^{m} r_j g_j(\boldsymbol{x}); & \text{otherwise} \end{cases} \tag{12}$$

where r_j is a variable penalty coefficient for the j-th constraint. For each constraint, they create several levels of violation. Depending on the level of violation, r_j varies accordingly. However, determining the level of violation for each constraints and choosing suitable values of r_j are not an easy task and problem-dependent.

Recent experiments of Michalewicz have indicated that the quality of solution heavily depends on the values of these penalty coefficients [3]. If the penalty coefficients are moderate, the algorithm may converge to an infeasible solution; on the other hand, if the penalty coefficients are too large, the method is equivalent to rejecting strategy.

3.4 Joines and Houck's method

Joines and Houck have considered the following nonlinear programming problem [7]:

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) \\ \text{s. t.} \quad & g_j(\boldsymbol{x}) \geq 0, \quad j = 1, 2, \cdots, q \\ & h_j(\boldsymbol{x}) = 0, \quad j = q+1, 2, \cdots, m \end{aligned}$$

and take the addition form of evaluation function.

$$eval(\boldsymbol{x}) = f(\boldsymbol{x}) \; + \; p(k, \boldsymbol{x})$$

The penalty function is also constructed with two components: (1) variable penalty factor and (2) penalty for the violation of constraints as follows:

$$p(k, \boldsymbol{x}) = \rho_k^{\alpha} \sum_{j=1}^{m} d_j^{\beta}(\boldsymbol{x}) \tag{13}$$

where k is the iteration of genetic algorithm, α and β are parameters used to adjust the scale of penalty value. The penalty term for single constraint $d_j(\boldsymbol{x})$ is given as follows:

$$d_j(\boldsymbol{x}) = \begin{cases} 0; & \text{if } \boldsymbol{x} \text{ is feasible} \\ |g_j(\boldsymbol{x})|; & 1 \le j \le q \text{ (if not)} \\ |h_j(\boldsymbol{x})|; & q+1 \le j \le m \text{ (if not)} \end{cases} \tag{14}$$

$$\rho_k = C \times k \tag{15}$$

where C is a constant. The penalty on infeasible chromosomes is increased along with evolutionary process due to the term of ρ_k. Comparing with Homaifar, Qi and Lai's method, the difference of two methods is that variable penalty term varies with the iteration of genetic algorithm for Joine and Houck's method while it varies according to the the level of violation for Homaifar, Qi and Lai's method.

The results of experiments of Joines and Houck indicated that the quality of the solution was very sensitive to the values of the three parameters. How to determine the variable penalty term is problem-dependent and also it is necessary to design the component with a proper dynamic property suitable for a given problem because this component is constantly increased along with growing of generations which approaches to give infeasible chromosomes *death penalty* at the later generation of genetic algorithms. In the most experiments of Michalewicz, the method converged in early generations due to this reason [3].

3.5 Michalewicz and Attia's method

Michalewicz and Attia have considered the following nonlinear programming problem [3]:

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) \\ \text{s. t.} \quad & g_j(\boldsymbol{x}) \le 0, \quad j = 1, 2, \cdots, q \\ & h_j(\boldsymbol{x}) = 0, \quad j = q+1, 2, \cdots, m \end{aligned}$$

and take the addition form of evaluation function.

$$eval(\boldsymbol{x}) = f(\boldsymbol{x}) \; + \; p(\tau, \boldsymbol{x})$$

The penalty function is also constructed with two components: (1) variable penalty factor and (2) penalty for the violation of constraints as follows:

$$p(\tau, \boldsymbol{x}) = \frac{1}{2\tau} \sum_{j \in A} d_j^2(\boldsymbol{x}) \tag{16}$$

where A is the set of active constraints, which consists all nonlinear equations and all violated nonlinear inequalities. A constraint $g_j(\boldsymbol{x})$ is violated at point $\boldsymbol{x}$ if and only if $g_j(\boldsymbol{x}) > \delta$ $(j = p+1, \ldots, m)$, where δ is a parameter to decides whether a constraint is active or not. τ is the variable penalty component, called *temperature*. The penalty term for single constraint $d_j(\boldsymbol{x})$ is given as follows:

$$d_j(\boldsymbol{x}) = \begin{cases} \max\{0, g_j(\boldsymbol{x})\}; & \text{for } 1 \le j \le q \\ |h_j(\boldsymbol{x})|; & \text{for } q+1 \le j \le m \end{cases} \tag{17}$$

They built a system called Genocop II with the technique [8]. Note that Genocop II is realized with the mechanism of SA-alike rather than GA. The variable component τ, beginning with a *starting* temperature τ_0 and ending at a *freezing* temperature τ_f, decreases in steps of the main loop according to a given cooling scheme. Genocop I is embedded in the main loop used to find an improved point. Within each execution of Genocop I, the temperature τ is fixed as a constant. The method is quite sensitive to the values of the parameters. The question of how to settle these parameters for a particular problem remains open.

3.6 Smith, Tate and Coit's method

The adapting penalty function is firstly proposed by Smith and Tate [9], which can alter the magnitude of the penalty dynamically by scaling according to the fitness of the best solution yet found. As better feasible and infeasible solutions are found, the penalty imposed on a given infeasible solution will change. Coit and Smith further extended their previous works in [10] and proposed the concept of *near-feasibility threshold* (*NFT*). Exterior penalty functions are characterized as being nondecreasing functions of the "distance" of a given solution from the feasible region. The *NFT* is the threshold distance from the feasible region at which the user would consider the search as "getting warm". The penalty function will encourage the GA to explore within the feasible region and the *NFT*-neighborhood of the feasible region, and discourage search beyond that threshold.

They considered the following nonlinear programming problem :

$$\begin{aligned} \max \quad & f(\boldsymbol{x}) \\ \text{s. t.} \quad & g_j(\boldsymbol{x}) \le b_j, \quad j = 1, 2, \cdots, m \end{aligned}$$

and take the addition form of evaluation function.

$$eval(\boldsymbol{x}) = f(\boldsymbol{x}) \; + \; p(\boldsymbol{x})$$

The penalty function is constructed with two components: (1) relative penalty coefficients for the violation of constraints and (2) adaptive penalty term as follows:

$$p(\boldsymbol{x}) = -\sum_{j=1}^{m} \left(\frac{\Delta b_j(\boldsymbol{x})}{\Delta b_j^{\text{nef}}} \right)^k (f_{all}^* - f_{feas}^*) \tag{18}$$

where k is a parameter which is used to adjust the severity of the penalty function. $\Delta b_j(\boldsymbol{x})$ is the value of violation for constraint j. Δb_j^{nef} is the *near-feasibility threshold* for constraint j. How to give a proper NFT is problem-dependent. f^*_{feas} is the objective function value of the best feasible solution yet found and f^*_{all} the unpenalized objective function value of the best overall solution yet found.

Note that the adaptive term $(f^*_{all} - f^*_{feas})$ may cause this penalty approach two kind of dangers: (1) zero-penalty and (2) over-penalty. For the case that $f^*_{all} = f^*_{feas}$ even though there exist infeasible solutions, this approach will give zero-penalty to all infeasible solutions; for the case that an infeasible one with very large value f^*_{all} occurred at the early stage of evolutionary process, this approach will give over-penalty to all infeasible solutions from then.

3.7 Yokota, Gen, Ida and Taguchi's method

Yokota, Gen, Ida and Taguchi have considered the same nonlinear programming problem as Smith, Tate and Coit do and take the multiplication form of evaluation function [11].

$$eval(\boldsymbol{x}) = f(\boldsymbol{x})\ p(\boldsymbol{x})$$

The penalty function is constructed as follows:

$$p(\boldsymbol{x}) = 1 - \frac{1}{m}\sum_{j=1}^{m}\left(\frac{\Delta b_j(\boldsymbol{x})}{b_j}\right)^k \tag{19}$$

$$\Delta b_j(\boldsymbol{x}) = \max\{0, g_j(\boldsymbol{x}) - b_j\} \tag{20}$$

where $\Delta b_j(\boldsymbol{x})$ is the value of violation for constraint j. This penalty function can be viewed as a special case of Smith, Tate and Coit's method that the *near-feasibility threshold* (NFT) for constraint j is set to be as $\Delta b_j^{\text{nef}} = b_j$. So it give a relative mild penalty than Smith, Tate and Coit's method. Note that the penalty function is designed with non-parameterized approach and is problem-independent.

3.8 Gen and Cheng's method

Gen and Cheng further refined their work in order to give a much severe penalty to infeasible one [12]. Let $\boldsymbol{x}$ be a chromosome in current population $P(t)$, the penalty function is constructed as follows:

$$p(\boldsymbol{x}) = 1 - \frac{1}{m}\sum_{j=1}^{m}\left(\frac{\Delta b_j(\boldsymbol{x})}{\Delta b_j^{\max}}\right)^k \tag{21}$$

$$\Delta b_j(\boldsymbol{x}) = \max\{0, g_j(\boldsymbol{x}) - b_j\} \tag{22}$$

$$\Delta b_j^{\max} = \max\{\epsilon, \Delta b_j(\boldsymbol{x}); \boldsymbol{x} \in P(t)\} \tag{23}$$

where $\Delta b_j(\boldsymbol{x}^i)$ is the value of violation for constraint j for the i-th chromosome, $\Delta b_j^{\max}$ is the maximum of violation for constraint j among current population, and ϵ is a small positive number used to have penalty avoid from zero-division. For highly constrained optimization problems, the infeasible solutions take a relative big portion among population at each generation. This penalty approach adjusts the ratio of penalties adaptively at each generation in order to make a balance between the preservation of information and the pressure for infeasibility and avoid over-penalty.

4 Discussion

The previous section surveyed several penalty functions. Each of them has advantages and disadvantages. The appropriate choice of the penalty function may depend on the nature of problem at hand. Followings are some discussions on the consideration when constructing a penalty function for a given problem.

Most methods take an addition form of evaluation function. In general, the absolute values of objective function $|f(\boldsymbol{x})|$ and penalty function $|p(\boldsymbol{x})|$ should be in comparable level for average case. Too small value of $|p(\boldsymbol{x})|$ will give an under-penalty for infeasible one and too large value will give an over-penalty. For many practical problems, constraints have different units or quantities in order. In such case, if penalty function on constraints is constructed with uniform way, it may occur that severe penalties give to some constraints while mild penalties give to remained constraints. This will cause genetic search approach to ineffective. This problem received less attention in literature. There is no such problem for the multiplication form of evaluation function. Adaptive penalty approach [10] makes a good attempt to adjust the amount of penalty according to the state of infeasibility in population. This work need to be further refined.

Dynamic approach give a constantly increasing penalty to infeasible solutions along with the increase of iteration number. The approach may work well in the case that (1) the optima lies within the feasible region and (2) the ratio of feasible solution to infeasible solution at each generation may relatively low. If the optima rides on the border of infeasible and feasible regions, we may hope that genetic search approaches to the optima from both side of feasible and infeasible areas. However, dynamic approach tends to give a death penalty to infeasible one at latter generation. This method has less attraction for the case.

For the case that optima rides on the border of infeasible and feasible regions, the technique of *near-feasibility threshold* given by Smith, Tate and Coit [10] seems much attractive. However, how to give a proper *near feasibility threshold* for each con-

straints is problem-dependent.

Most penalties techniques belong to the class of problem-dependent approach. For example, in Homaifar, Qi and Lai's method, we need to determine the level of violation of constraints and choose suitable values of r_j for a particular problem. In Joines and Houck's method, we need to determine three parameters properly for a given problem. In Michalewicz and Attia's method, we need to give a *starting* temperature, *freezing* temperature, and cooling scheme for a given problem. The penalty method given by Gen and Cheng belongs to problem-independent approach [12].

An ideal penalty amount to an infeasible one should consider following two factors:

- *how far from feasible region*
- *how close to optima*

All existing methods consider only the first factor. It seems very difficult to embed the second information into penalty function because we have no any priori knowledge about optima. Can we find a surrogate way ?

It is also worthwhile to make some good test problems. Most test problems given in literatures seem simple and small. A typical test problem may be characterized with following factors: (1) complex solution space (may not be convex or connected); (2) complex objective functions with multidodal; (3) the position of optima (lies in feasible area or rides on border); (4) the size of problem (number of variables and number of constraints); and (5) the ratio of feasible to infeasible solutions (the higher the ratio the much difficult the problem). It may be interesting to make an extensive numerical tests with different penalty methods.

Acknowledgment

This work was supported in part by a research grant from the Ministry of Education, Science and Culture, Japanese Government: Grant-in-Aid for Scientific Research, the International Scientific Research Program (No. 07045032).

References

[1] Gen, M. and R. Cheng, *Genetic Algorithms and Engineering Design*, John Wiley & Sons, New York, 1996(to appear).

[2] Michalewicz, Z., Genetic algorithms, numerical optimization, and constraints, In Eshelman, L. J., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 151–158, 1995.

[3] Michalewicz, Z., A survey of constraint handling techniques in evolutionary computation methods, In McDonnell, J., R. Reynolds, and D. Fogel, editors, *Proceeding of the 4th Annual Conference on Evolutionary Programming*, pp. 135–155, 1995.

[4] Glover, F. and H. Greenberg, New approaches for heuristic search: a bilateral linkage with artificial intelligence, *European Journal of Operational Research*, vol.39, pp.119–130, 1989.

[5] Richardson, J., M. Palmer, G. Liepins, and M. Hilliard, Some guidelines for genetic algorithms with penalty functions, In Schaffer, J., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 191–197, 1989.

[6] Homaifar, A., C. Qi, and S. Lai, Constrained optimization via genetic algorithms, *Simulation*, vol.62, no.4, pp.242–254, 1994.

[7] Joines, J. and C. Houck, On the use of nonstationary penalty functions to solve nonlinear constrained optimization problems with GAs, In Fogel, D., editor, *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp. 579–584, 1994.

[8] Michalewicz, Z. and N. Attia, Evolutionary optimization of constrained problems, In Sebald, A. and L. Fogel, editors, *Proceeding of the 3rd Annual Conference on Evolutionary Programming*, pp. 98–108, 1994.

[9] Smith, A. and D. Tate, Genetic optimization using a penalty function, In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 499–505, 1993.

[10] Coit, D. and A. Smith, Penalty guided genetic search for reliability design optimization, *International Journal of Computers and Industrial Engineering*, 1995(to appear).

[11] Yokota, T., M. Gen, K. Ida, and T. Taguchi, Optimal design of system reliability by an improved genetic algorithm, *Transactions of Institute of Electronics, Information and Computer Engineering*, vol.J78-A, no.6, pp.702–709, 1995.

[12] Gen, M. and R. Cheng, Interval programming using genetic algorithms, *Sixth International Symposium on Robotics and Manufacturing*, France, 1996.