Penalty Functions

Alice E. Smith and David W. Coit

Department of Industrial Engineering

University of Pittsburgh

Pittsburgh, Pennsylvania 15261 USA

Section C 5.2 of *Handbook of Evolutionary Computation*

Editors in Chief

Thomas Baeck

David Fogel

Zbigniew Michalewicz

A Joint Publication of Oxford University Press and Institute of Physics Publishing

Revised January 1996

Published as "Constraint-Handling Techniques - Penalty Functions," Alice E. Smith and David W. Coit, in *Handbook of Evolutionary Computation*, Institute of Physics Publishing and Oxford University Press, Bristol, U.K., 1997, Chapter C5.2.

C 5.2 Penalty Functions

This chapter begins with the motivation and general form of penalty functions as used in evolutionary computation. The main types of penalty functions - constant, static, dynamic, adaptive - are described within a common notation framework. References from the literature concerning these exterior penalty approaches are presented. The chapter concludes with a brief discussion of promising areas of future research in penalty methods for constrained optimization by evolutionary computation.

C 5.2.1 Introduction to Penalty Functions

Penalty functions have been a part of the literature on constrained optimization for decades. Two basic types of penalty functions exist; exterior penalty functions, which penalize infeasible solutions, and interior penalty functions, which penalize feasible solutions. It is the former type of penalty functions which is discussed throughout section C 5.2, however the area of interior penalty functions is of potential research interest in evolutionary computation. The main idea of interior penalty functions is that an optimal solution requires that a constraint be active (i.e., tight) so that this optimal solutions lies on the boundary between feasibility and infeasibility. Knowing this, a penalty is applied to feasible solutions when the constraint is not active, so-called "interior solutions." For a single constraint, this approach is straightforward (although it has not been seen in the evolutionary computation literature), however for the more common case of multiple constraints, the implementation of interior penalty functions is considerably more complex.

Three degrees of exterior penalty functions exist: (1) barrier methods in which no infeasible solution is considered, (2) partial penalty functions in which a penalty is applied near the feasibility boundary, and (3) global penalty functions that are applied throughout the infeasible region (Schwefel 1995, page 16). In the area of combinatorial optimization, the popular Lagrangian relaxation method (Avriel 1976, Fisher 1981, Reeves 1993) is a variation on the same theme: temporarily relax the problem's most difficult constraints, using a modified objective function to avoid straying too far from the feasible region. In general, a penalty function approach is as follows. Given an optimization problem, the following is the most general formulation of constraints:

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{x} \in \mathrm{A} \\ & \mathbf{x} \in \mathrm{B} \end{aligned} \tag{1}$$

where $\mathbf{x}$ is a vector of decision variables, the constraints "$\mathbf{x} \in \mathrm{A}$" are relatively easy to satisfy, and the constraints "$\mathbf{x} \in \mathrm{B}$" are relatively difficult to satisfy, the problem can be reformulated as

$$\begin{aligned} \min \quad & f(\mathbf{x}) + p(d(\mathbf{x}, \mathrm{B})) \\ \text{s.t.} \quad & \mathbf{x} \in \mathrm{A} \end{aligned} \tag{2}$$

where $d(\mathbf{x}, \mathrm{B})$ is a metric function describing the distance of the solution vector $\mathbf{x}$ from the region B, and $p(\cdot)$ is a monotonically non-decreasing penalty function such that $p(0) = 0$. If the exterior penalty function, $p(\cdot)$, grows quickly enough outside of B, the optimal solution of (1) will also be optimal for (2). Furthermore, any optimal solution of (2) will provide an upper bound on the optimum for (1), and this bound will in general be tighter than that obtained by simply optimizing $f(\mathbf{x})$ over A.

In practice, the constraints "$\mathbf{x} \in \mathrm{B}$" are expressed as inequality and equality constraints in the form of

$$g_i(\mathbf{x}) \leq 0 \text{ for } i = 1, \ldots, q$$

$$h_i(\mathbf{x}) = 0 \text{ for } i = q + 1, \ldots, m$$

where q = number of inequality constraints

$m - q$ = number of equality constraints

Various families of functions $p(\cdot)$ and $d(\cdot)$ have been studied for evolutionary optimization to dualize constraints. Different possible distance metrics, $d(\cdot)$, include a count of the number of violated constraints, the Euclidean distance between $\mathbf{x}$ and B as suggested by Richardson et al. (1989), a linear sum of the individual constraint violations or a sum of the individual constraint violations raised to an exponent, κ. Variations of these approaches have been attempted with different degrees of success. Some of the more notable examples are described in the following sections.

It can be difficult to find a penalty function that is an effective and efficient surrogate for the missing constraints. The effort required to tune the penalty function to a given problem instance or repeatedly calculate it during search may negate any gains in eventual solution quality. As noted by Siedlecki and Sklansky (1989), much of the difficulty arises because the optimal solution will frequently lie on the boundary of the feasible region. Many of the solutions most similar to the genotype of the optimum solution will be infeasible. Therefore, restricting the search to only feasible solutions or imposing very severe penalties makes it difficult to find the schemata that will drive the population toward the optimum as shown in the research of Smith and Tate (1993), Anderson and Ferris (1994), Coit et al. (1996) and Michalewicz (1995). Conversely, if the penalty is not severe enough, then too large a region is searched and much of the search time will be used to explore regions far from the feasible region. Then, the search will tend to stall outside the feasible region. A good comparison of six penalty function strategies applied to continuous optimization problems is given in Michalewicz (1995). These strategies include both static and dynamic approaches, as discussed below, as well as some less generic approaches such as sequential constraint handling (Schoenauer and Xanthakis 1993) and forcing all infeasible solutions to be dominated by all feasible solutions in a given generation (Powell and Skolnick 1993).

C 5.2.2 Static Penalty Functions

A simple method to penalize infeasible solutions is to apply a constant penalty to those solutions that violate feasibility in any way. The penalized objective function would then be the unpenalized objective function plus a penalty (for a minimization problem). A variation is to construct this simple penalty function as a function of the number of constraints violated, where there are multiple constraints. The penalty function for a problem with m constraints would then be as below (for a minimization problem):

$$f_p(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{m} C_i \delta_i \tag{3}$$

$$\text{where} \begin{cases} \delta_i = 1, \text{ if constraint } i \text{ is violated} \\ \delta_i = 0, \text{ if constraint } i \text{ is satisfied} \end{cases}$$

$f_p(\mathbf{x})$ is the penalized objective function, $f(\mathbf{x})$ is the unpenalized objective function, and C_i is a constant imposed for violation of constraint i. This penalty function is based only on the number of constraints violated, and is generally inferior to the second approach based on some distance metric from the feasible region (Goldberg 1989, Richardson et al. 1989).

More common and more effective is to penalize according of distance to feasibility, or the "cost to completion," as termed by Richardson et al. (1989). This was done crudely in the constant penalty functions of the preceding paragraph by assuming distance can be stated solely by number of constraints violated. A more sophisticated and more effective penalty includes a distance metric for each constraint, and adds a penalty that becomes more severe with distance from feasibility. Complicating this approach is the assumption that the distance metric chosen appropriately provides information concerning the nearness of the solution to feasibility, and the further implicit assumption that this nearness to feasibility is relevant in the same magnitude to the fitness of the solution. Distance metrics can be continuous (see for example, Juliff 1993) or discrete (see for example, Patton et al. 1995), and could be linear or nonlinear (see for example, Le Riche et al. 1995).

A general formulation is as follows for a minimization problem:

$$f_p(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{m} C_i d_i^{\kappa} \tag{4}$$

$$\text{where} \quad d_i = \begin{cases} \delta_i g_i(\mathbf{x}), & \text{for } i = 1, \ldots, q \\ |h_i(\mathbf{x})|, & \text{for } i = q+1, \ldots, m \end{cases}$$

d_i is the distance metric of constraint i applied to solution $\mathbf{x}$ and κ is a user-defined exponent, with values of κ of 1 or 2 often used. Constraints 1 through q are inequality constraints, so the penalty will only be activated when the constraint is violated (as shown by the δ function above), while constraints q + 1 through m are equality constraints which will activate the penalty if there is any distance between the solution value and the constraint value (as shown in the absolute distance above). In equation 4 above, defining C_i is more difficult. The advice from Richardson et al. (1989) is to base C_i on the expected or maximum cost to repair the solution (i.e. alter the solution so it is feasible). For most problems, however, it is not possible to determine C_i using this rationale. Instead, it must be estimated based on the relative scaling of the distance metrics of multiple constraints, the difficulty of satisfying a constraint, and the seriousness of a constraint violation, or be determined experimentally.

Many researchers in evolutionary computation have explored variations of distance-based static penalty functions (e.g., Baeck and Khuri 1994, Goldberg 1989, Huang et al. 1994, Olsen 1994, Richardson et al. 1989). One example (Thangiah 1995) uses a linear combination of three constant distance based penalties for the three constraints of the vehicle routing with time windows problem. Another novel example is from Le Riche et al. (1995) where two separate distance-based penalty functions are used for each constraint in two genetic algorithm segregated subpopulations. This "double penalty" somewhat improved robustness to penalty function parameters since the feasible optimum is approached with both a severe and a lenient penalty. Homaifar et al. (1994) developed a unique static penalty function with multiple violation levels established for each constraint. Each interval is defined by the relative degree of constraint violation. For each interval l, a unique constant, C_{il} is then used as a penalty function coefficient. This approach has the considerable disadvantage of requiring iterative tuning through experimentation of a large number of parameters.

C 5.2.3 Dynamic Penalty Functions

The primary deficiency with static penalty functions is the inability of the user to determine criteria for the C_i coefficients. Also, there are conflicting objectives involved with allowing exploration of the infeasible region, yet still requiring that the final solution be feasible. A variation of distance-based penalty functions, that alleviates much of these difficulties, is to incorporate a dynamic aspect that (generally) increases the severity of the penalty for a given distance as the search progresses. This has the property of allowing highly infeasible solutions early in the search, while continually increasing the penalty imposed to eventually move the final solution to the feasible region. A general form of a distance based penalty method incorporating a dynamic aspect based on length of search, *t*, is as follows for a minimization problem:

$$f_p(\mathbf{x},t) = f(\mathbf{x}) + \sum_{i=1}^{m} s_i(t) d_i^{\kappa} \tag{5}$$

where $s_i(t)$ is a monotonically non-decreasing in value with *t*. Metrics for *t* include number of generations or the number of solutions searched. Recent uses of this approach include Joines and Houck (1994) for continuous function optimization and Olsen (1994) and Michalewicz and Attia (1994), which compare several penalty functions, all of which consider distance, but some also consider evolution time. A common objective of these dynamic penalty formulations is that they result in feasible solutions at the end of evolution. If $s_i(t)$ is too lenient, final infeasible solutions may result, and if $s_i(t)$ is too severe, the search may converge to non-optimal feasible solutions. Therefore, these penalty functions typically require problem-specific tuning to perform well. One explicit example of $s_i(t)$ is as follows, from Joines and Houck (1994),

$$s_i(t) = (C_i\, t)^{\alpha}$$

where α is constant equal to 1 or 2, as defined by Joines and Houck.

C 5.2.4 Adaptive Penalty Functions

While incorporating distance together with the length of the search into the penalty function has been generally effective, these penalties ignore any other aspects of the search. In this respect, they are not adaptive to the ongoing success (or lack thereof) of the search and cannot guide the search to particularly attractive regions or away from unattractive regions based on what has already been observed. A few authors have proposed making use of such search-specific information. Siedlecki and Sklansky (1989) discuss the possibility of adaptive penalty functions, but their method is restricted to binary-string encodings with a single constraint, and involves considerable computational overhead.

Bean and Hadj-Alouane (1992) and Hadj-Alouane and Bean (1992) propose penalty functions that are revised based on the feasibility or infeasibility of the best, penalized solution during recent generations. Their penalty function allows either an increase or a decrease of the imposed penalty during evolution as shown below, and was demonstrated on multiple choice integer programming problems with one constraint. This involves the selection of two constants, β_1 and β_2 ($\beta_1>\beta_2>1$), to adaptively update the penalty function multiplier, and the evaluation of the feasibility of the best solution over successive intervals of N_f generations. As the search progresses, the penalty function multiplier is updated every N_f generations based on whether or not the best solution was feasible during that interval. Specifically, the penalty function is as follows,

$$f_p(\mathbf{x},k) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_k d_i^{\kappa} \tag{6}$$

$$\lambda_{k+1} = \begin{cases} \lambda_k \beta_1, & \text{if previous } N_f \text{ generations have infeasible best solution} \\ \lambda_k / \beta_2, & \text{if previous } N_f \text{ generations have feasible best solution} \\ \lambda_k, & \text{otherwise} \end{cases}$$

Smith and Tate (1993) and Tate and Smith (1995) used both search length and constraint severity feedback in their penalty function, which was enhanced by the work of Coit et al. (1996). This penalty function involves the estimation of a near-feasible threshold (*NFT*) for each constraint. Conceptually, the *NFT* is the threshold distance from the feasible region at which the user would consider the search as "getting warm." The penalty function encourages the evolutionary algorithm to explore within the feasible region and the *NFT*-neighborhood of the feasible region, and discourage search beyond that threshold. This formulation is given below:

$$f_p(\mathbf{x},t) = f(\mathbf{x}) + (F_{feas}(t) - F_{all}(t)) \sum_{i=1}^{m} \left(\frac{d_i}{NFT_i} \right)^{\kappa} \tag{7}$$

where $F_{all}(t)$ denotes the unpenalized value of the best solution yet found, and $F_{feas}(t)$ denotes the value of the best feasible solution yet found. The $F_{all}(t)$ and $F_{feas}(t)$ terms serve several purposes. First, they provide adaptive scaling of the penalty based on the results of the search. Second, they combine with the NFT_i term to provide a search specific and constraint specific penalty.

The general form of NFT_i is:

$$NFT_i = \frac{NFT_{oi}}{1+\Lambda_i} \tag{8}$$

where NFT_{oi} is an upper bound for NFT_i. Λ_i is a dynamic search parameter used to adjust NFT_i based on the search history. In the simplest case, Λ_i can be set to zero and a static NFT_i results. Λ_i can also be defined as a function of the search, for example, a function of the generation number (t), i.e., $\Lambda_i = f(t) = \lambda_i t$. A positive value of λ_i results in a monotonically decreasing NFT_i (and thus, a larger penalty) and a larger λ_i more quickly decreases NFT_i as the search progresses, incorporating both adaptive and dynamic elements.

If NFT_i is intuitively ill-defined, it can be set at a large value initially with a positive constant λ_i used to iteratively guide the search to the feasible region. This dynamic NFT_i circumvents the need to perform experimentation to determine appropriate penalty function parameter values. However, if problem-specific information is at hand, a more efficient search can take place by *a priori* defining a tighter region or even static values of NFT_i.

C 5.2.5 Future Directions in Penalty Functions

Two areas requiring further research are the development of completely adaptive penalty functions that require no user-specified constants and the development of improved adaptive operators to exploit characteristics of the search as they are found. The notion of adaptiveness is to leverage the information gained during evolution to improve both the effectiveness and the efficiency of the penalty function used. Another area of interest is to explore the assumption that multiple constraints can be linearly combined to yield an appropriate penalty function. This implicit assumption of all penalty functions used in the literature assumes that constraint violations incur independent penalties and therefore, there is no interaction between constraints. Intuitively, this seems to be a possibly erroneous assumption, and one could make a case for a penalty that increases more than linearly with the number of constraints violated.

C 5.2 References

E J Anderson and M C Ferris 1994 Genetic algorithms for combinatorial optimization: the assembly line balancing problem *ORSA Journal on Computing* **6** 161-173

M. Avriel 1976 *Nonlinear Programming: Analysis and Methods* Prentice Hall Englewood Cliffs NJ

T Baeck and S Khuri 1994 An evolutionary heuristic for the maximum independent set problem *Proceedings of the First IEEE Conference on Evolutionary Computation* 531-535

J C Bean and A B Hadj-Alouane 1992 A dual genetic algorithm for bounded integer programs University of Michigan Technical Report 92-53 to appear in *RAIRO - RO*

D W Coit A E Smith and D M Tate 1996 Adaptive penalty methods for genetic optimization of constrained combinatorial problems *INFORMS Journal on Computing* in print

M L Fisher 1981 The Lagrangian relaxation method for solving integer programming problems *Management Science* **27** 1-18

D E Goldberg 1989 *Genetic Algorithms in Search Optimization and Machine Learning* Addison-Wesley Reading MA

A B Hadj-Alouane and J C Bean 1992 A genetic algorithm for the multiple-choice integer program University of Michigan Technical Report 92-50 to appear in *Operations Research*

A Homaifar S H-Y Lai and Z Qi 1994 Constrained optimization via genetic algorithms *Simulation* 62 242-254

W-C Huang C-Y Kao and J-T Horng 1994 A genetic algorithm approach for set covering problem *Proceedings of the First IEEE Conference on Evolutionary Computation* 569-573

J A Joines and C R Houck 1994 On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's *Proceedings of the First IEEE Conference on Evolutionary Computation* 579-584

K Juliff 1993 A multi-chromosome genetic algorithm for pallet loading *Proceedings of the Fifth International Conference on Genetic Algorithms* 467-473

R G Le Riche C Knopf-Lenoir and R T Haftka 1995 A segregated genetic algorithm for constrained structural optimization *Proceedings of the Sixth International Conference on Genetic Algorithms* 558-565

Z Michalewicz 1995 Genetic algorithms numerical optimization and constraints *Proceedings of the Sixth International Conference on Genetic Algorithms* 151-158

Z Michalewicz and N Attia 1994 Evolutionary optimization of constrained problems *Proceedings of the 3rd Annual Conference on Evolutionary Programming* 98-108

A L Olsen 1994 Penalty functions and the knapsack problem *Proceedings of the First IEEE Conference on Evolutionary Computation* 554-558

A L Patton W F Punch III and E D Goodman 1995 A standard GA approach to native protein conformation prediction *Proceedings of the Sixth International Conference on Genetic Algorithms* 574-581

D Powell and M M Skolnick 1993 Using genetic algorithms in engineering design optimization with non-linear constraints *Proceedings of the Fifth International Conference on Genetic Algorithms* 424-430

C R Reeves 1993 *Modern Heuristic Techniques for Combinatorial Problems* John Wiley & Sons New York

J T Richardson M R Palmer G Liepins and M Hilliard 1989 Some guidelines for genetic algorithms with penalty functions *Proceedings of the Third International Conference on Genetic Algorithms* 191-197

M Schoenauer and S Xanthakis 1993 Constrained GA optimization *Proceedings of the Fifth International Conference on Genetic Algorithms* 573-580

H-P Schwefel 1995 *Evolution and Optimum Seeking* John Wiley & Sons

W Siedlecki and J Sklansky 1989 Constrained genetic optimization via dynamic reward-penalty balancing and its use in pattern recognition *Proceedings of the Third International Conference on Genetic Algorithms* 141-150

A E Smith and D M Tate 1993 Genetic optimization using a penalty function *Proceedings of the Fifth International Conference on Genetic Algorithms* 499-505

D M Tate and A E Smith 1995 Unequal area facility layout using genetic search *IIE Transactions* **27** 465-472

S R Thangiah 1995 An adaptive clustering method using a geometric shape for vehicle routing problems with time windows *Proceedings of the Sixth International Conference on Genetic Algorithms* 536-543