Constraint-handling through a multiobjective optimization technique

Carlos A. Coello Coello*
Laboratorio Nacional de Informática Avanzada
Rébsamen 80, A.P. 696
Xalapa, Veracruz, México 91090
e-mail: ccoello@xalapa.lania.mx
Phone: +52 (28) 18-13-02
Fax: +52 (28) 18-15-08

July 28, 1999

Abstract

This paper presents an alternative approach to handle constraints using a population-based multiobjective optimization technique. The proposed approach is used to solve several engineering optimization problems, and the results produced are compared with those obtained using other (GA-based and mathematical programming) techniques.

1 INTRODUCTION

Despite the well-documented success of genetic algorithms (GAs) in a wide range of applications, their use in constrained optimization problems still raises several issues to which a considerable amount of research has been devoted in the last few years. From these, the key issue is how to incorporate constraints of any sort (linear, non-linear, equality or inequality) into the fitness function as to guide the search properly. For several years, practitioners have used penalty functions to incorporate (mainly inequality) constraints into the fitness function, and there have been a lot of successful applications of this approach in all engineering fields. However, penalty functions have some well-known limitations [12], from which the most significant is the difficulty to define good penalty factors, which are normally generated by trial and error, although their value may severely

*Please send all correspondence to: PO Box 60326-394, Houston, Texas 77205.

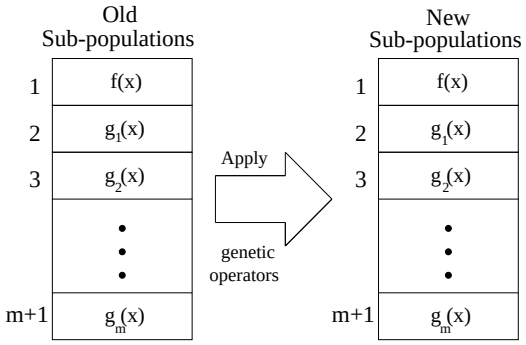

Figure 1: Graphical representation of the approach introduced in this paper.

affect the results produced by the GA [12]. In this paper, a new constraint-handling approach is proposed that does not require the use of a penalty function to handle equality and inequality constraints. This technique is based on a multiobjective optimization approach, and is very suitable for parallelization. The remainder of this paper is organized as follows: first, the proposed approach will be introduced, then 3 engineering optimization problems will be presented, and solved using it. After that, the results produced by other (GA-based and mathematical programming) techniques will be compared with those obtained with the proposed method, and finally there will be some discussion of the results obtained and the expected paths of future research.

2 USE OF MULTIOBJECTIVE OPTIMIZATION TECHNIQUES

The main idea of this approach is to redefine the single-objective optimization of f as a multiobjective optimization problem in which we will have $m + 1$ objectives, where m is the number of constraints. Then, we can apply any multiobjective optimization technique [4] to the new vector $\bar{v} = (f, f_1, \ldots, f_m)$, where $f_1, \ldots, f_m$ are the original constraints of the problem. An ideal solution $\mathbf{X}$ would thus have $f_i(\mathbf{X})$=0 for $1 \leq i \leq m$ and $f(\mathbf{X}) \leq \mathbf{f}(\mathbf{Y})$ for all feasible $\mathbf{Y}$ (assuming minimization).

3 DESCRIPTION OF THE NEW APPROACH

The main idea behind the approach proposed in this paper is to use a population-based multiobjective optimization technique such as VEGA [13] to handle each

of the constraints as an objective in the way indicated before. The technique may be better illustrated by Figure 1. At each generation, the population is split into $m+1$ sub-populations, where m refers to the number of constraints of the problem. Although the size of each sub-population may be variable, for the sake of simplicity, it was decided to allocate the same size to each of them in the experiments reported in this paper. Using the proposed scheme, a fraction of the population will be selected using the (unconstrained) objective function as its fitness; another fraction will use the first constraint as its fitness and so on. For the sub-population guided by the objective function, the evaluation of such objective function for a given vector $\mathbf{X}$ (decoded from the chromosome) is used directly as the fitness function (multiplied by (-1) if it is a minimization problem), with no penalties of any sort. For all the other sub-populations, the algorithm used was the following (we assume that every constraint $g(\mathbf{X}) \geq \mathbf{0}$ is considered satisfied):

if $g_j(\mathbf{X}) < 0.0$	**then**	fitness $= g_j(\mathbf{X})$
else if $v \neq 0$	**then**	fitness $= -v$
else		fitness $= f$

where $g_j(\mathbf{X})$ refers to the constraint corresponding to sub-population $j+1$ (this is assuming that the first sub-population is assigned to the objective function f), and v refers to the number of constraints that are violated ($\leq m$). Crossover and mutation are applied as usual to the entire population, allowing the mixing of all the sub-populations as in the original proposal of VEGA [13]. There are a few interesting things that can be observed from this procedure. First, each sub-population associated with a constraint will try to reduce the amount in which that constraint is violated. If the solution evaluated does not violate the constraint corresponding to that sub-population, but it is infeasible, then the sub-population will try to minimize the total number of violations, joining then the other sub-populations in the effort of driving the GA to the feasible region. This aims at combining the distance from feasibility with information about the number of violated constraints, which is the same heuristic normally used with penalty functions. However, traditionally it is necessary to define in advance either an static penalty value or a dynamic penalty function that estimates this distance from feasibility, whereas in the current approach such distance is estimated automatically by the above algorithm using the constraint violation information derived from the GA run. Finally, if the solution encoded is feasible, then this individual will be 'merged' with the first sub-population, since it will be evaluated with the same fitness function (i.e., the objective function). It is important to clarify that the current approach does not use dominance to impose an order on the constraints based on their violation (like in the case of COMOGA [14]) which is a more expensive process (in terms of CPU time) that also requires additional parameters. In fact, the current approach does not rank individuals, but it uses instead different fitness functions for each of the sub-population allocated (whose number depends on the number of con-

straints) depending on the feasibility of the individuals contained within each of them. This is easier to implement, does not require special operators to preserve feasiblity (like in the case of Parmee and Purchase's approach [11]), makes unnecessary the use of a sharing function to preserve diversity (like with traditional multiobjective optimization techniques), and does not require extra parameters to control the mixture of feasible and infeasible individuals (like in the case of COMOGA [14]). It is interesting to notice that the use of the unconstrained objective function in one of the sub-populations may assign good fitness values to infeasible individuals. However, because the number of constraints will normally be greater than one, the other sub-populations will drive the GA to the feasible region. In fact, the sub-population evaluated with the objective function will be useful to keep diversity in the population, making then unnecessary the use of sharing techniques. The behavior expected under this scheme is to have few feasible individuals at the beginning, and then gradually produce solutions that may be feasible with respect to some constraints but not with respect to others. Over time, the building blocks of these sub-populations will combine to produce individuals that are feasible, but not necessarily optimum. At that point the direct use of the objective function will help the GA to approach the optimum, but since some infeasible solutions will still be present in the population, those individuals will be responsible to keep the diversity required to avoid stagnation. Although VEGA is known to have difficulties in multiobjective optimization problems due to the fact that it tries to find individuals that excel only in one dimension regardless of the others (the so-called "middling" problem [13]), that drawback turns out to be an advantage in this context, because what we want to find are precisely solutions that are completely feasible, instead of good compromises that may not satisfy one of the constraints.

4 EXAMPLES

Two examples taken from the optimization literature will be used to show the way in which the proposed approach works. Notice that the ranges shown for the design variables are the same reported in the original references from where these problems were obtained.

4.1 Example 1 : Himmelblau's Nonlinear Optimization Problem

This problem was originally proposed by Himmelblau [8], and it was chosen to try the new approach because it has been used before as a benchmark for GA-based techniques that use penalties. In this problem, there are 5 design variables $(x_1, x_2, x_3, x_4, x_5)$, 6 nonlinear inequality constraints and 10 boundary conditions. The problem can be stated as follows:

$$Minimize \quad f(\mathbf{X}) = \mathbf{5.3578547 x_3^2} +$$

$$+0.8356891x_1x_5 + 37.293239x_1 - 40792.141 \tag{1}$$

Subject to:

$$g_1(\mathbf{X}) = \mathbf{85.334407 + 0.0056858x_2x_5} + \\ +0.00026x_1x_4 - 0.0022053x_3x_5 \tag{2}$$

$$g_2(\mathbf{X}) = \mathbf{80.51249 + 0.0071317x_2x_5} + \\ +0.0029955x_1x_2 + 0.0021813x_3^2 \tag{3}$$

$$g_3(\mathbf{X}) = \mathbf{9.300961 + 0.0047026x_3x_5} + \\ +0.0012547x_1x_3 + 0.0019085x_3x_4 \tag{4}$$

$$0 \leq g_1(\mathbf{X}) \leq \mathbf{92} \tag{5}$$

$$90 \leq g_2(\mathbf{X}) \leq \mathbf{110} \tag{6}$$

$$20 \leq g_3(\mathbf{X}) \leq \mathbf{25} \tag{7}$$

$$78 \leq x_1 \leq 102 \tag{8}$$

$$33 \leq x_2 \leq 45 \tag{9}$$

$$27 \leq x_3 \leq 45 \tag{10}$$

$$27 \leq x_4 \leq 45 \tag{11}$$

$$27 \leq x_5 \leq 45 \tag{12}$$

4.2 Example 2 : Design of a 10-bar plane truss

Consider the 10-bar plane truss shown in Figure 2 [1]. The problem is to find the cross-sectional area of each member of this truss, such that we minimize its weight, subject to stress and displacement constraints. The weight of the truss is given by:

$$f(x) = \sum_{j=1}^{10} \rho\, A_j\, L_j \tag{13}$$

where x is the candidate solution, A_j is the cross-sectional area of the jth member, L_j is the length of the jth member, and ρ is the weight density of the material. The assumed data are: modulus of elasticity, $E = 1.0 \times 10^4$ ksi 68965.5 MPa), $\rho = 0.10$ lb/in^3 (2768.096 kg/m^3), and a load of 100 kips (45351.47 Kg) in the negative y-direction is applied at nodes 2 and 4. The maximum allowable stress of each member is called σ_a, and it is assumed to be ± 25 ksi (172.41 MPa). The maximum allowable displacement of each node (horizontal and vertical) is represented by u_a, and is assumed to be 2 inches

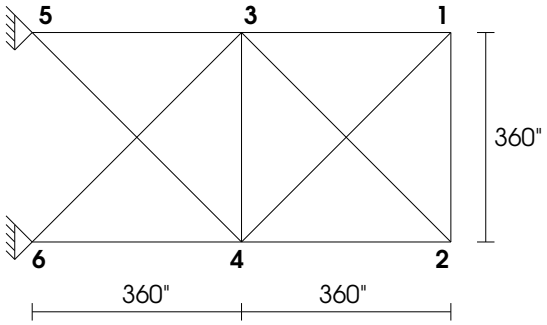

Figure 2: 10-bar plane truss used for the third example

(5.08 cm). There are 10 stress constraints, and 12 displacement constraints (we can really assume only 8 displacement constraints because there are two nodes with zero displacement, but they will nevertheless be considered as additional constraints by the new approach). The cross-section of each element can be different, thus the problem has 10 design variables.

5 COMPARISON OF RESULTS

In the experiments reported in this paper, a GA with fixed-point representation [3] was used, together with two-point crossover, binary tournament selection and non-uniform mutation [10]. In all cases, the GA was run 81 times, varying the mutation rate from 0.1 to 0.9 in increments of 0.1 (mutation rates were considered with respect to the whole string, and not on a bit-per-bit basis). The results reported correspond to the best solution found in these 81 runs. For further details of the experimental setup the interested reader may consult [2].

5.1 EXAMPLE 1

This problem was originally proposed by Himmelblau [8] and solved using the Generalized Reduced Gradient method (GRG). Gen and Cheng [5] solved this problem using a genetic algorithm based on both local and global reference. The result shown in Table 1 is the best of the two reported by Gen and Cheng [5]. Homaifar, Qi, and Lai [9] solved this problem using a genetic algorithm with a population size of 400, and their results were the best previously reported in the literature for this problem (see Table 1). The solution shown for the technique proposed here is the best produced after 81 runs in which the crossover and mutation rates were iterated from 0.1 to 0.9 in a nested loop, and the

Design Variables	Best solution found			
	This paper	**Gen**	**Homaifar**	**GRG**
x_1	78.5958	81.4900	78.0000	78.6200
x_2	33.0100	34.0900	33.0000	33.4400
x_3	27.6460	31.2400	29.9950	31.0700
x_4	45.0000	42.2000	45.0000	44.1800
x_5	45.0000	34.3700	36.7760	35.2200
$g_1(\mathbf{X})$	91.956402	90.522543	90.714681	90.520761
$g_2(\mathbf{X})$	100.545111	99.318806	98.840511	98.892933
$g_3(\mathbf{X})$	20.251919	20.060410	19.999935	20.131578
$f(\mathbf{X})$	**−30810.359**	**−30183.576**	**−30665.609**	**−30373.949**

Table 1: Comparison of results for the first example (Himmelblau's function).

following ranges were used for the design variables: $78.0000 \leq x_1 \leq 102.0000$, $33.0000 \leq x_2 \leq 45.0000$, $27.0000 \leq x_3 \leq 45.0000$, $27.0000 \leq x_4 \leq 45.0000$, and $27.0000 \leq x_5 \leq 45.0000$. The values for all the variables were considered with a 4-decimal precision. The total population size used was 160 (40 individuals for each of the 4 sub-populations) and the maximum number of generations was 100.

5.2 Example 2

This problem was used by Belegundu [1] to evaluate the following numerical optimization techniques: Feasible directions (CONMIN and OPTDYN), Pshenichny's Recursive Quadratic Programming (LINRM), Gradient Projection (GRP-UI), Exterior Penalty Function (SUMT), Multiplier Methods (M-3, M-4 and M-5). The results reported by Belegundu [1] are compared to the current approach in Tables 2 and 3 (all the solutions presented are feasible). To solve this problem, it was necessary to add a module responsible for the analysis of the plane truss. This module uses the matrix factorization method included in Gere and Weaver [6] together with the stiffness method [6] to analyze the structure, and returns the values of the stress and displacement constraints, as well as the total weight of the structure. The solution shown for the technique proposed here is the best produced after 81 runs in which the crossover and mutation rates were iterated from 0.1 to 0.9 in a nested loop, and the range $0.1 \leq x \leq 299.00$ was used for the 10 design variables. The values for all the variables were considered with a 2-decimal precision. The total population size used was 230 (10 individuals for each of the 23 sub-populations) and the maximum number of generations was 100.

Design Variables	Best solution found			
	This paper	**CONMIN**	**OPTDYN**	**LINRM**
x_1	30.00	25.28	25.77	21.57
x_2	0.10	1.90	0.10	10.98
x_3	22.40	24.87	25.11	22.08
x_4	16.19	15.83	19.39	14.95
x_5	0.10	0.10	0.10	0.10
x_6	0.57	1.75	0.10	10.98
x_7	7.74	16.76	15.36	18.91
x_8	22.15	19.73	20.32	18.42
x_9	20.80	20.98	20.74	18.40
x_{10}	0.10	2.51	1.14	13.51
$f(\mathbf{X})$	**5082.76**	**5563.32**	**5471.48**	**6428.89**

Table 2: Comparison of results for the second example (10-bar plane truss). Part I.

Design Variables	Best solution found				
	GRP-UI	**SUMT**	**M-3**	**M-4**	**M-5**
x_1	24.78	30.69	25.84	31.62	25.84
x_2	4.17	2.37	3.07	11.81	2.88
x_3	24.79	31.62	26.42	31.62	26.45
x_4	14.45	11.66	12.77	17.50	12.75
x_5	0.10	0.10	0.10	31.62	0.10
x_6	4.17	3.71	3.44	10.25	3.77
x_7	17.46	21.71	19.34	31.62	19.38
x_8	19.26	20.90	19.17	31.62	19.18
x_9	19.27	13.97	18.76	31.62	18.77
x_{10}	5.26	3.26	4.42	31.62	4.38
$f(\mathbf{X})$	**5727.05**	**5932.21**	**5719.19**	**11279.22**	**5726.08**

Table 3: Comparison of results for the second example (10-bar plane truss). Part II.

6 DISCUSSION

In the examples presented before, the new approach found better solutions than those previously reported in the literature by using relatively small sub-population sizes. However, the selection of an appropriate sub-population size (assuming that they are all the same) remains an issue as when using a GA with a single population. Determining the maximum number of generations presents a similar problem, although in this case it is possible to monitor the population so that the GA is stopped when there is not enough diversity anymore. The problems selected to illustrate the technique had different kinds of constraints so that the new algorithm could be tested under different conditions. Example 1 for instance, is an engineering design problem, with their constraints given in algebraic form. Example 2 is a numerical optimization problem that has been used several times before to test constraint handling approaches. Finally, example 3 does not have its constraints defined in algebraic form either, since they are derived from the module that performs the analysis of the structure. The main drawback of the new technique may be the number of sub-populations that may be needed in larger problems, since they will increase linearly with the number of constraints. However, it is possible to deal with that problem in two different ways: first, some constraints could be tied; that means that two or more constraints could be assigned to the same sub-population. That would significantly reduce the number of sub-populations in highly constrained problems. Second, we could parallelize the approach, in which case a high number of sub-populations will not be a serious drawback, since they could be processed concurrently. The current algorithm would however need modifications as to decide the sort of interactions between a master process (responsible for actually optimizing the whole problem) and the slave sub-processes (all the sub-populations responsible for the constraints of the problem). That is in fact the area of research currently being pursued by the author.

7 CONCLUSIONS AND FUTURE WORK

This paper has introduced a GA-based approach that uses a multiobjective optimization technique to handle constraints, instead of using the more traditional penalty approach. The new approach worked well in several test problems that had been previously solved using GA-based and mathematical programming techniques, producing in all cases results better than those previously reported in the literature. The technique was able to achieve such good results with relatively small sub-populations, and without the need to use any extra parameters for the GA, although the issue of selecting the most appropriate sub-population size as well as the maximum number of generations remains open as in the case of the simple GA [7]. The first extension of this work is to develop a parallel implementation of the algorithm, so that instead of using a single population

and split it according to the number of constraints, several (fairly small) sub-populations are generated, each being responsible for a single constraint or set of constraints. Some interesting issues that a parallel version of this algorithm arise are for example the migration policies required to exchange information, the consequences of restricting crossover, the effect of the topology used by the parallel architecture on the overall performance of the GA, the significance of the evolution of the small sub-populations responsible for the constraints concurrently with the evolution of a main population containing a mixed of feasible and infeasible solutions.

8 ACKOWLEDGMENTS

The author acknowledges support from CONACyT through project number I-29870 A.

References

[1] Ashok Dhondu Belegundu. *A Study of Mathematical Programming Methods for Structural Optimization.* Dept. of civil and environmental engineering, University of Iowa, Iowa, Iowa, 1982.

[2] Carlos A. Coello Coello. Treating Constraints as Objectives for Single-Objective Evolutionary Optimization. *Engineering Optimization*, 32, 1999. (Accepted for publication).

[3] Carlos Artemio Coello Coello. *An Empirical Study of Evolutionary Techniques for Multiobjective Optimization in Engineering Design.* PhD thesis, Department of Computer Science, Tulane University, New Orleans, LA, apr 1996.

[4] Carlos M. Fonseca and Peter J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, Spring 1995.

[5] Mitsuo Gen and Runwei Cheng. *Genetic Algorithms & Engineering Design.* John Wiley & Sons, Inc, New York, 1997.

[6] James M. Gere and William Weaver. *Analysis of Framed Structures.* D. Van Nostrand Company, Inc., 1965.

[7] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Co., Reading, Massachusetts, 1989.

[8] David M. Himmelblau. *Applied Nonlinear Programming.* McGraw-Hill, New York, 1972.

[9] A. Homaifar, S. H. Y. Lai, and X. Qi. Constrained Optimization via Genetic Algorithms. *Simulation*, 62(4):242–254, 1994.

[10] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, second edition, 1992.

[11] I. C. Parmee and G. Purchase. The development of a directed genetic search technique for heavily constrained design spaces. In I. C. Parmee, editor, *Adaptive Computing in Engineering Design and Control-'94*, pages 97–102, Plymouth, UK, 1994. University of Plymouth, University of Plymouth.

[12] Jon T. Richardson, Mark R. Palmer, Gunar Liepins, and Mike Hilliard. Some guidelines for genetic algorithms with penalty functions. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–197, George Mason University, 1989. Morgan Kaufmann Publishers.

[13] J. David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93–100. Lawrence Erlbaum, 1985.

[14] Patrick D. Surry, Nicholas J. Radcliffe, and Ian D. Boyd. A Multi-Objective Approach to Constrained Optimisation of Gas Supply Networks : The COMOGA Method. In Terence C. Fogarty, editor, *Evolutionary Computing. AISB Workshop. Selected Papers*, Lecture Notes in Computer Science, pages 166–180. Springer-Verlag, Sheffield, U.K., 1995.