Proceedings of the
1999 ASME Design Engineering Technical Conferences
September 12-15, 1999, Las Vegas, Nevada

DETC99/DAC-8584

COMPARISON OF TWO MULTIOBJECTIVE OPTIMIZATION TECHNIQUES WITH AND WITHIN GENETIC ALGORITHMS

Shapour Azar * Brian J. Reynolds Sanjay Narayanan

Department of Mechanical Engineering,
University of Maryland,
College Park, MD 20742
*phone: (301) 405-5250, fax: (301) 314-9477, e-mail: azarm@eng.umd.edu

KEYWORDS

Optimization-based design, genetic algorithms, multiobjective optimization

ABSTRACT

Engineering decision making involving multiple competing objectives relies on choosing a design solution from an optimal set of solutions. This optimal set of solutions, referred to as the Pareto set, represents the tradeoffs that exist between the competing objectives for different design solutions. Generation of this Pareto set is the main focus of multiple objective optimization. There are many methods to solve this type of problem. Some of these methods generate solutions that cannot be applied to problems with a combination of discrete and continuous variables. Often such solutions are obtained by an optimization technique that can only guarantee local Pareto solutions or is applied to convex problems. The main focus of this paper is to demonstrate two methods of using genetic algorithms to overcome these problems. The first method uses a genetic algorithm with some external modifications to handle multiple objective optimization, while the second method operates within the genetic algorithm with some significant internal modifications. The fact that the first method operates with the genetic algorithm and the second method within the genetic algorithm is the main difference between these two techniques. Each method has its strengths and weaknesses, and it is the objective of this paper to compare and contrast the two methods quantitatively as well as qualitatively. Two multiobjective design optimization examples are used for the purpose of this comparison.

INTRODUCTION

It is common in engineering decision making problems to have multiple design objectives (see, for example, Eschenauer et al., 1990). There are many methods for solving such problems, each of which has some advantages and disadvantages, and most have evolved or been refined from some earlier version, as discussed in the survey b Stadler (1979) and Hwang and Masud (1979), Palli et al.(1998), and more recently b Miettinen (1999). Most of these methods generate solutions that cannot be applied to problems with a combination of discrete and continuous variables. Often, such solutions can only be guaranteed to be locall optimum. However, optimization with Genetic Algorithms (hereafter referred to as GAs) can obtain discrete, global, and non-convex solutions. In this paper, we will look at two different ways of utilizing GAs to solve multiobjective design optimization problems. The first method uses an existing GA with some modifications external to the GA, and the second method operates within the GA with more significant modifications to it. This brings up the concept of multiobjective design optimization with and within GAs.

The multiobjective optimization with genetic algorithms just uses an existing GA (Goldberg, 1989) with some modifications external to the GA. The GA basically operates independently of the optimization problem formulation with only some minimal data passed to and from it. However, the multiobjective optimization within the GA works on the inside of the GA making some significant modifications to the algorithm. Each method has its advantages and disadvantages with respect to one another. In general, as it will be shown in the paper, the multiobjective optimization with GA requires more computational time but is easier to

apply, and within GA requires less computational time but can encounter some difficulties in generating the Pareto set. So, the overall objective of this paper is to compare and contrast quantitatively as well as qualitatively two different methods of multiobective optimization that utilize GAs, highlighting the strengths and weaknesses of each.

The remainder of the paper is organized as follows. The formulation of a general multiobjective optimization problem and some terminology are given in the next section. Next, an overview of GAs is. The second and third sections explain the two multiobjective optimization techniques that are to be compared. The fourth section includes the application of these methods to two different engineering design problems for the purpose of the comparison. The paper is concluded with the remarks in the last section.

OVERVIEW OF MULTIOBJECTIVE OPTIMIZATION PROBLEMS

The formulation of a typical multiobjective optimization problem with *m* objective functions is shown below in Eq.(1).

$$\begin{aligned} &Minimize: \mathbf{f}(\mathbf{x}) = \{ f_1(\mathbf{x}),..., f_i(\mathbf{x}),..., f_m(\mathbf{x})\} \\ &subject\ \ to: \mathbf{x} \in D \\ &D = \{\mathbf{x} : g_j(\mathbf{x}) \le 0,\ j = 1,..., J; h_k(\mathbf{x}) = 0, k = 1,...K\} \end{aligned} \tag{1}$$

where $\mathbf{x}$ is the design vector containing the n components of design variables, $f_i(\mathbf{x})$ is the i^{th} objective function to be minimized, $g_j(\mathbf{x})$ is the j^{th} inequality constraint and $h_k(\mathbf{x})$ is the k^{th} equality constraint. The feasible design space or set of all design vectors that satisfy all the constraints is denoted as D. I Eq.(1), the word 'minimize' means that: (i) all objective functions are simultaneously minimized, (ii) the objectives are at least partly conflicting with one another, and (iii) there does not exist a single solution that is optimal with respect to every objective function. This is the general form of a constrained multiobjective optimization problem that will be used in the example problems later on in this paper. To facilitate the presentation, a brief overview of some of the multiobjective optimization terminology in the form of definitions is given below (see, for example, Chankong and Haimes, 1983).

Pareto solution: A design solution $\mathbf{x}^* \in D$ is said to be (strongly) Pareto optimal if there does not exist another solution $\mathbf{x} \in D$ such that $f_i(\mathbf{x}) \le f_i(\mathbf{x}^*)$ for all $i = 1, \ldots, m$ wit strict inequality for at least one i. Pareto solutions are 'non-inferior' with respect to each other. A collection of Pareto solutions is referred to as a Pareto set. A solution that is not Pareto is referred to as an *inferior* solution.

Proper Pareto solution: A Pareto solution is proper if the tradeoff rate between the objectives in the neighborhood of that solution is bounded. In other words, in the neighborhood of a proper Pareto point, a finite increase in one objective is possible only at the expense of some reasonable decrease in one other objective. A prope Pareto solution is a good candidate design solution.

Ideal and nadir points: If each of the objective functions in Eq.(1) is individually minimized subject to the constraints defining the feasible region D, then the components of an ideal vector (or an ideal point), $\{f_1^*, \ldots, f_m^*\}$, are obtained. The ideal point is often used as a reference point in multiobjective optimization problems and is the best solution that can be achieved. However, it is extremely unlikely that an optimized solution fo Eq.(1) achieves an ideal point. In a minimization problem, as in Eq.(1), the ideal point provides a lower bound of the Pareto optimal set. In contrast, an uppe bound of the Pareto set defines the components of a nadi point. The nadir point is given by $\{f_{1*}, \ldots, f_{m*}\}$.

Good value: A good value for the i^{th} objective is an estimate of the i^{th} component of the ideal vector, f_i^*. It is a value towards which the decision maker would like a particula objective to take on.

Bad value: A bad value for the i^{th} objective is an estimate of the i^{th} component of the nadir vector, f_{i*}. It is a value awa from which the decision maker would like a particula objective to take on.

AN OVERVIEW OF GENETIC ALGORITHMS

In the most general sense, GA-based optimization is a stochastic search method that involves the random generation of potential design solutions and then systematically evaluates and refines the solutions until a stopping criteria is met (Goldberg, 1989). The GA software package used in this paper, originally designed for unconstrained single-objective design optimization problems, was developed at Argonne National Labs (Levine, 1996). A constrained single-objective optimization problem can be converted to an unconstrained single-objective form by a penalty method (see, for example, Papalambros and Wilde, 1988). The GA can be furthe extended, as will be shown in this paper, to multiobjective constrained optimization problems. Figure 1 shows graphically how the most basic GA operations are performed.

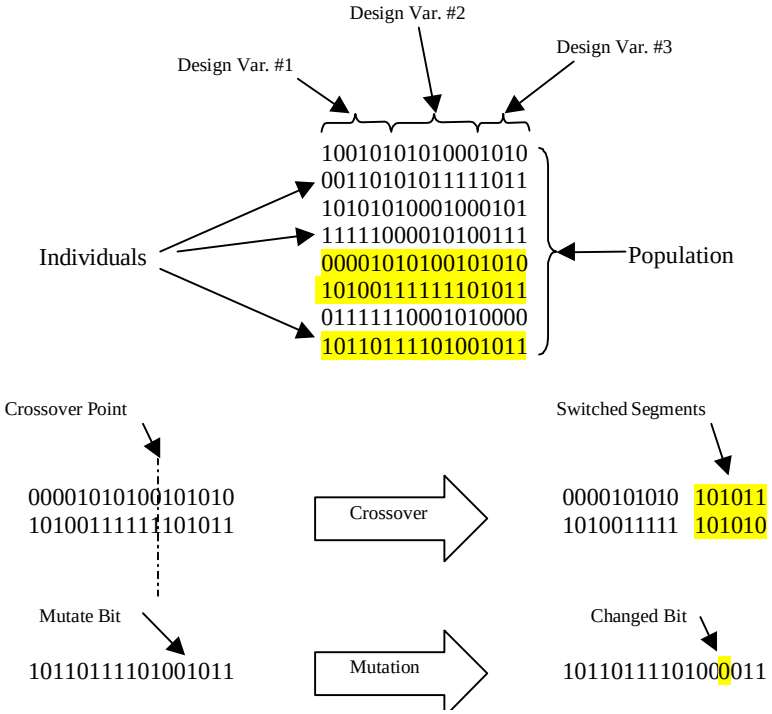

Figure 1: Graphical Description of Genetic Algorithm Basics

The top block of binary numbers in Figure 1 represents a population. Each row is an individual that represents one solution to the design problem. One individual is made up of all of the design variables concatenated. The GA user can decide how many binary digits are needed to represent each design variable. In Figure 1, there are 3 design variables of length 6, 7, and 4 binary digits. Initially, the population is generated randomly, and then the solutions are ranked from best to worst and a specified number of the lowest ranked individuals are replaced with combinations of the highest ranked individuals. The process of determining which of the highest ranked individuals are to be used is called selection. There are several differing methods of selection that can be used. Once selected, two individuals go through a process called crossover. The crossover operation is also shown in Figure 1, wherein two individuals (or parents) exchange a segment of the binary digits creating two new individuals (o offspring). Another basic GA operation is mutation. Mutation can occur on the two individuals selected fo crossover or on a single individual. The mutation operation randomly mutates or changes the bits within the individual based on the mutation probability set by the user. Finding optimal values for the parameters used in the GA operations can be problematic. Parameter values that result in a relatively fast convergence for one problem may be slow fo another. However, some general guidelines on the population size can be made on the basis of the binary string length of an individual.

This brings up some of the disadvantages of GAs. Sometimes, the algorithm can generate inferior points if the design variables are not represented accurately enough, or if the feasible domain of the problem is irregular. So, there is no absolute guarantee that the GA will converge to the best solution, but in most cases, it will converge to a very good approximation of the best solution. The GA may also require a large amount of computational time and/or function calls to generate solutions when compared to a conventional gradient-based optimization method. Therefore, the main advantage is that it solves problems in which the gradient does not exist, problems that include discrete-type variables, problems where the feasible space is non-convex, and problems where global rather than local solutions are sought.

In order to extend the GA to a multiobjective problem, some steps have to be taken. The methods considered in this paper are called I-SHOT (Interactive Sequential Hybrid Optimization Technique) and MOGA (Multi-Objective Genetic Algorithm), and these are briefly explained in the following sections (see, Narayanan and Azarm, 1999a and 1999b, for more information).

INTERACTIVE SEQUENTIAL HYBRID OPTIMIZATION TECHNIQUE (I-SHOT

This multiobjective optimization technique has three main attributes: (i) the method guarantees detection of str and proper Pareto optimal solutions, (ii) it reduces the feasible region at each iteration with only a brief interaction with the decision maker, and (iii) it allows the decision make to have control over the range of solutions generated at each iteration. The last attribute isn't significant for the purpose of this paper, since the range of solutions generated will be the

same for both methods and can be set at the beginning of the optimization process. The I-SHOT method itself was developed by Narayanan and Azarm (1999a) based on an extension to the hybrid method (i.e., a combination of the weighting and ε-constraint methods) by Chankong and Haimes (1983) and Steuer's (1986) Interactive weighted Tchebycheff (IWT) approach. The hybrid method was to improve upon the ε-constraint technique, which turns a multiobjective problem into a single objective problem b constraining all objective functions except one and then minimizing the remaining objective (Haimes et al., 1971). By combining the ε-constraint method with the weighting method and the IWT method, the decision maker can not only obtain proper Pareto optimal solutions, but also have control over the optimization process to further refine the Pareto set according to his or her preferences (Narayanan and Azarm, 1999a).

Note that in the weighting method, the weighted combination of objectives generally does not detect points in a non-convex Pareto set. This difficulty, however, is overcome by the hybrid method that uses a combination of the weighting and ε-constraint methods. The interactive stage can be replaced with a systematic reduction of the Pareto set, which will also detect non-convex regions of the Pareto set. The focus of this paper isn't on the interactive characteristics of the I-SHOT technique (covered elsewhere by Narayanan and Azarm, 1999a), but on just the generation of the full range of Pareto optimal points, therefore, the decision maker preferences aren't taken into consideration during the generation of the Pareto set.

The formulation used to generate each individual Pareto point is shown below in Eq.(2).

$$\begin{aligned} &\text{Minimize}: \sum_{i=1}^{m} w_i \left[\frac{(f_i(\mathbf{x}) - \varepsilon_{i,Good})}{(\varepsilon_{i,Bad} - \varepsilon_{i,Good})} \right] \\ &\text{Subject to}: \ f_i(\mathbf{x}) \le \varepsilon_i, i = 1,...,m; \mathbf{x} \in D \\ &\text{where}: \varepsilon_i \in [\varepsilon_{i,good}, \varepsilon_{i,bad}] \end{aligned} \tag{2}$$

Here, in addition to converting the multiobjective problem into a single objective form (i.e., weighted sum of the scaled objectives), an upper bound (hereafter referred to as 'objective constraint') is imposed on each objective. In Eq.(2), $\varepsilon_{i,good}$ is the good value for objective function i, $\varepsilon_{i,bad}$ is the bad value for objective function i, and w_i is the weighting facto (defined in step 2 below). The good and bad values are used to scale each of the objectives such that 0 will correspond to the good, scaled value, and 1 will correspond to the bad value. This allows the objectives to be compared on an even level, which may not be possible when the range of values of each objective can be orders of magnitude apart. Anothe advantage of this is that all objective functions, including those that are to be maximized, are automatically converted to a minimization form. Then, the different Pareto optimal solutions can be obtained by systematically altering the weighting factor (as shown in step 2 below). Each iteration consists of generating a set of Pareto points, obtaining the decision maker's feedback, and reducing the range of the Pareto set for the next iteration by placing constraints on the objective functions. The steps in I-SHOT are described below.

Step 1 – Formulate problem: The problem model is developed according to Eq.(1) and then converted into the form of Eq.(2). The decision maker selects the targets fo $\varepsilon_{i,good}$ and $\varepsilon_{i,bad}$ values for the first iteration. For subsequent iterations, these targets may be obtained according to step 5 (see below). The objective functions are scaled using these values. The decision maker then determines the number of Pareto solutions to be generated.

Step 2 – Generate weights: The weighting vector is generated. The number of elements in a weighting vector, a collection of ($w_1, \ldots, w_i, \ldots, w_m$), will correspond to the number of solutions that will be generated. The weighting vector will be generated randomly according to the formulation below in Eq.(3). Each individual weighting factor, w_i , lies on the interval between 0 and 1, but not including 0 or 1, and all of the weighting factors sum to 1.

$$\begin{aligned} &W\text{i} = \text{random}(0,1), \forall \ i = 1,..., m \\ &\overline{W} = \sum_{i=1}^{m} W_i \end{aligned} \tag{3}$$

$$w_i = \begin{cases} W_i + \dfrac{(1-\overline{W})}{\overline{W}}(W_i) & -if \ \overline{W} > 1- \\ W_i & -if \ \overline{W} = 1- \\ W_i + \dfrac{(1-\overline{W})}{(m-\overline{W})}(1-W_i) & -if \ \overline{W} < 1- \end{cases} \tag{3}$$

Step 3 – Generate Pareto set: Solve the hybrid problem, Eq.(2), for each of the weighting factors generated. Since Eq.(2) is essentially a single objective constrained optimization problem, any optimization method can be used depending on the problem characteristics such as the type of variables, linear or non-linear objective functions and constraints. In this paper, however, the GA is used to do the optimization.

Step 4 – Stop or reduce ε-range: If the decision maker is satisfied with the Pareto set, then the process is stopped. Otherwise, a new range of $\varepsilon_{i,good}$ and $\varepsilon_{i,bad}$ (maximum and minimum values for the objectives) values are chosen as follows:

$$\begin{aligned} \varepsilon_{i,bad}^{new} &= \varepsilon_{i,bad} - \frac{\Delta\varepsilon_i}{r} \\ \varepsilon_{i,good}^{new} &= \varepsilon_{i,good} + \frac{\Delta\varepsilon_i}{r} \\ \Delta\varepsilon_i &= \varepsilon_{i,bad} - \varepsilon_{i,good} \end{aligned} \tag{4}$$

Here, r is a reduction factor, $\varepsilon_{i,bad}$ is the old bad value, $\varepsilon_{i,good}$ is the old good value, $\Delta\varepsilon_i$ is the difference between the old bad and old good values, and the ε^{new} values are the new good and bad values for objective function f_i. This modification of the ε-range together with the ε-constraints allows the detection of the non-convex points of the Pareto set.

The steps, 1 through 4, represent one iteration. Additional iterations are implemented until the decision maker is satisfied with the prope Pareto set that is generated.

MULTI-OBJECTIVE GENETIC ALGORITHM (MOGA)

MOGA is essentially a modification and extension of the single objective GA. Instead of converting the multiple objective problem into a single objective form and obtaining one non-inferior Pareto point at a time, as in I-SHOT, MOGA obtains the Pareto points all at once. Figure 2 illustrates the steps in MOGA. The algorithm initiates in the same way as in a conventional GA, with the generation of an initial population. For the initial population, the non-inferior points or individuals are identified. Note that, while these individuals are non-inferior for the current population, they are non-Pareto for the problem in an absolute sense. If these individuals are non-inferior and feasible for the current population, then they are given the highest rank in the population. These non-inferior individuals become parents and are set to produce offspring, and then the process is repeated. As such, the population is gradually improved as it approaches the final population and the corresponding Pareto set for the problem. A step by step approach fo implementing MOGA is given below. The unique features of this MOGA are discussed later on in the paper in the example section. MOGA is described in more details in Narayanan and Azarm (1999b), and is based on an extension of the technique b Fonseca and Fleming (1998).

Step 1 – Problem formulation and coding of variables: The problem model is developed according to Eq.(1). Each variable in the problem is represented by a binary string. The length of the string is determined by the desired precision. Strings, representing all variables, are then concatenated to form an individual.

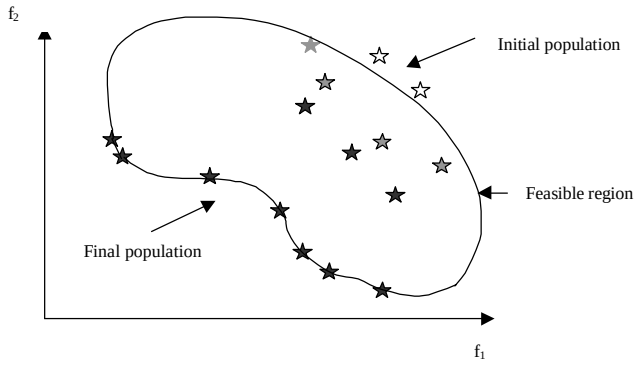

Figure 2: Graphical Illustration of MOGA

Step 2 – Generation of initial population: An initial population of individuals is randomly generated.

Step 3 – Population ranking and fitness assignment: All of the objective functions are evaluated for each individual. The ranking of the individuals is substantially different from that of a single objective GA. Each individual is compared with every other individual to determine whether or not it is inferior with respect to others and to what extent. All of the constraints are evaluated for each of the non-inferior individuals. For each constraint that is violated, the individual is penalized and its fitness value is reduced. Those individuals that do not violate any constraint are given the highest fitness value, and all the others are assigned lowe fitness values. It may be advisable to be lenient in the earl generations because all of the individuals in the population may be violating some constrains. However, the actual fitness is problem dependent and may need some experimentation to determine how to best penalize solutions for violation of each constraint.

Step 4 – Selection: Before the parents are selected to produce offspring, the population is filtered by deleting some individuals from each niche. The number of individuals being deleted depends on how crowded or how dense the niche is. The selected parents are then tested for relative locations as part of the mating restriction. This is to prevent close parents from mating and hence to get a more evenl spread and uniform sampling of the Pareto set. One way to implement this would be to compute the distance (L_2 norm) between the two parents selected for reproduction by:

$$L_{ij} = \sqrt{\sum_{k=1}^{m} (f_k(x_i) - f_k(x_j))^2} \qquad (5)$$

where x_i and x_j are two solutions. If this distance is found to be less than some limiting distance (tolerance), the parents shall not be allowed to mate. The limiting distance would depend on how dense the niches are and what the range of the population is. The selection process is carried out until a sufficient number of parents have been selected to produce offspring. The L_2 norm was used for the distance for the results obtained in this paper.

Step 5 – Crossover and Mutation: At this step, the selected individuals are crossed-over and mutated to produce the next generation. That is, each individual exchanges a segment of its binary string to create two new individuals. The mutation operation can occur with crossover or independent of it. This is a simple operation where the bits of an individual are randomly chosen to mutate or change. After crossover and mutation occur the next generation is formed.

Steps 3 through 5 are repeated until the decision maker pauses the algorithm in order to impose/review constraints on the objectives or until the stopping criterion has been

satisfied. The stopping criterion in MOGA is formed by using a scheme for the L_2 norm. The scheme is illustrated as follows:

- For each individual in the non-inferior set (i.e., the Pareto set in transition), the L_2 distance from a desired target point (chosen by the decision maker, preferably an ideal point) is computed. Hence, for each generation, one set of L_2 metrics is obtained.
- The mean and standard deviation of these L_2 norms are calculated.
- As the non-inferior set improves across generations, the points on the set get closer and closer to the target. Therefore, the distance of each point on the set from the target decreases. This decrease in L_2 metrics can be measured by the mean. If the improvement in the mean is less than some small number, MOGA is assumed to have converged and stopped.

EXAMPLES

The two methods described in the previous sections will now be applied to two multiobjective optimization examples for the purpose of comparison. The GA package used to generate these results was slightly modified from a package developed by Levine (1996). The first example is a relativel simple example taken from Kirsch (1981) with some modifications that involves the minimization of volume and stress of a *two-bar truss*. The second example is taken from Messac (1996), also with some modifications. It is a problem with a *vibrating platform*, that involves the maximization of fundamental frequency as one objective, and the minimization of cost added in the paper as the othe objective.

Example 1: Two-Bar Truss

Figure 3 illustrates the *two-bar truss* that is to be optimized. This problem was adapted from the problem b Kirsch (1981). It is comprised of two stationary pinned joints, A and B, where each one is connected to one of the two bars in the truss. The two bars are pinned where the join one another at joint C, and a 100 kN force acts directl downward at that point. The cross-sectional areas of the two bars are represented as x_1 and x_2, the cross-sectional areas of trusses AC and BC respectively. Finally, y represents the perpendicular distance from the line AB that contains the two-pinned base joints to the connection of the bars where the force acts (joint C). The *two-bar truss* is shown below.

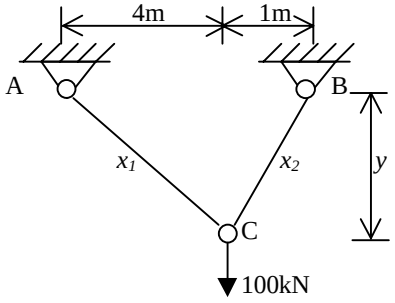

Figure 3: Two-Bar Truss

The problem has been modified into a two-objective problem in order to show the non-inferior Pareto set clearly in two dimensions. The stresses in AC and BC should not exceed 100,000 kPa and the total volume of material should not exceed 0.1 m^3. The reason the objective constraints have been imposed is that the Pareto set is asymptotic and extends from -∞ to ∞. As x_1 and x_2 go to zero, f_{volume} goes to zero and $f_{stress,AC}$ and $f_{stress,BC}$ go to infinity. As x_1 and x_2 go to infinity, f_{volume} goes to infinity and $f_{stress,AC}$ and $f_{stress,BC}$ go to zero. Hence, in order to generate Pareto optimal solutions in a reasonable range, objective constraints are imposed. The problem formulation is shown below.

$$\text{Minimize } f_{volume} = x_1(16+y^2)^{0.5} + x_2(1+y^2)^{0.5}$$
$$\text{Minimize } f_{stress,AC} = \frac{20(16+y^2)^{0.5}}{yx_1} \tag{6}$$

$$\begin{aligned} \text{Subject to: } & f_{volume} \le 0.1 \\ & f_{stress,AC} \le 100000 \\ & f_{stress,BC} \le 100000 \\ & 1 \le y \le 3 \\ & x_1, x_2 > 0 \end{aligned}$$

where

$$f_{stress,BC} = \frac{80(1+y^2)^{0.5}}{yx_2} \tag{7}$$

Comparison of Two-Bar Solutions

The two-bar truss results for I-SHOT and MOGA are shown in Figures 4 and 5, respectively. For this example, MOGA outperformed I-SHOT. It took about 1/30 th the number of function calls to generate the non-inferior Pareto set. However, the actual processing time required to generate the solution was not as dramatically different. On a Pentium II 450 MHz processor, the I-SHOT method required roughl 8.5 seconds to generate the set, and the MOGA method required roughly 2.0 seconds. Also, for this example, the MOGA solution was spread more evenly, giving more potential solutions to choose from (34 distinct points, see Figure 5), where the I-SHOT solution had some overlapping points (21 distinct points, see Figure 4). Both methods

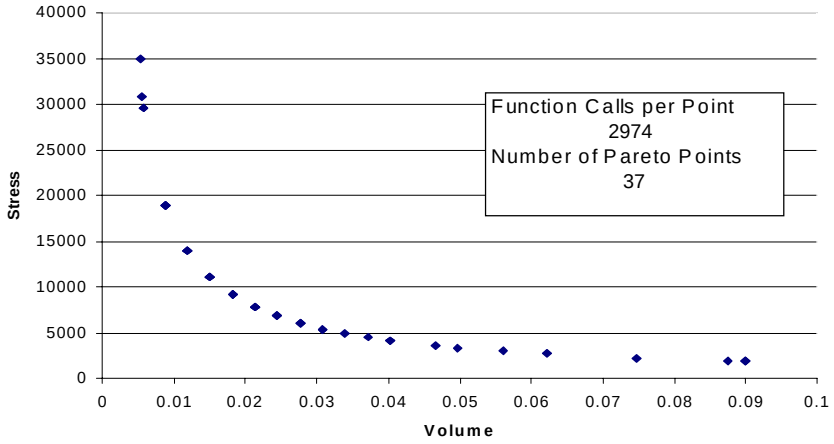

Figure 4: Two-Bar Truss (I-SHOT Solution)

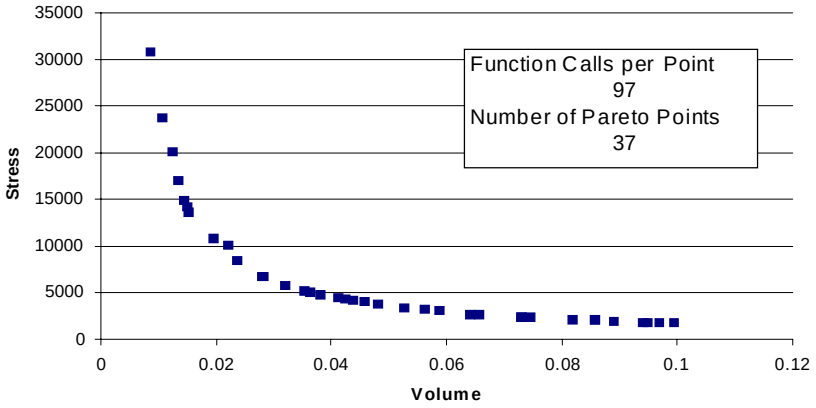

Figure 5: Two-Bar Truss (MOGA Solution)

generated 37 total points, but some of these were in fact duplicates, and thus are not shown as 37 distinct points in the figures. The GA parameters used to generate these solutions are shown in the following table.

	I-SHOT	**MOGA**
Mutation Prob	0.01	0.01
Crossover Prob.	0.85	0.85
Selection type	SUS	SUS
Crossover type	Two-point	Two-point

Table 1: GA Parameters

Noting the differences in the two-bar truss solutions, another comparison was made with respect to the actual processing time and function calls for a given population size. As shown in Figures 6 and 7 below, the two methods were compared at population sizes of 100, 300, 500, and 1000. For a population size of 100, MOGA outperformed I-SHOT in both tests. However, when the population size increased, MOGA eventually required more processing time than the I-SHOT method. MOGA did require fewer function-calls fo all of the population sizes. So, if the function call is dominant (requires the most time), then MOGA, as shown here, may be more efficient. When the problem requires high precision, generally a larger population size is needed. If the binary string representing one solution of the problem becomes long, then a small population size will only represent an increasingly smaller portion of the solution space. This is analogous to picking 4 integers between 1 and 10. The 4 selected integers represent 40% of the possible solution space. Here, the 4 integers represent the population size, and 10 are the maximum number of possible solutions that is governed by the binary string length of one solution. Next, if 4 integers are picked between 1 and 100, then only 4% of the possible solution space are represented. With onl 4% of the possible solutions being represented, the population, in many cases, can not be considered a good statistical representation of the solution set. The solution set distribution can sometimes be highly irregular, and the Pareto set can be concentrated in a small region, therefore, the population size required for the algorithm to converge to the Pareto solution will have to be proportionally larger. It is, fo this reason, that there exist some problems that the I-SHOT method may be more efficient. For instance, this may include problems where the objective functions are relatively simple, but the range of the design variables is large and the required precision high.

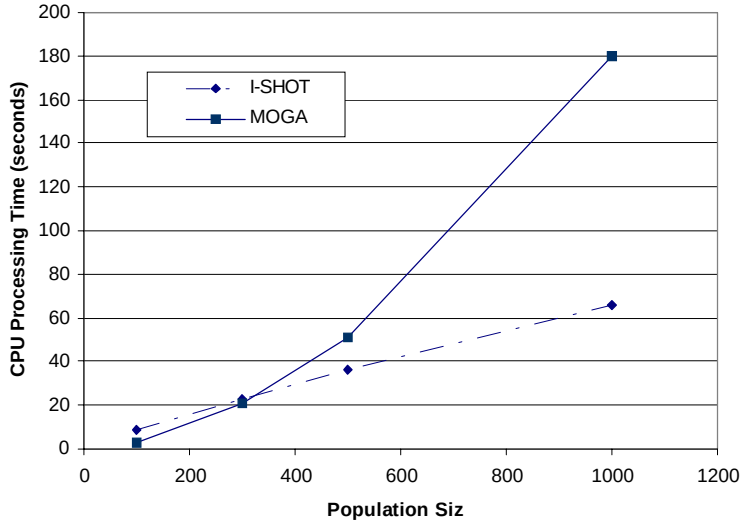

Figure 6: Two Bar Truss Processing Time Comparison

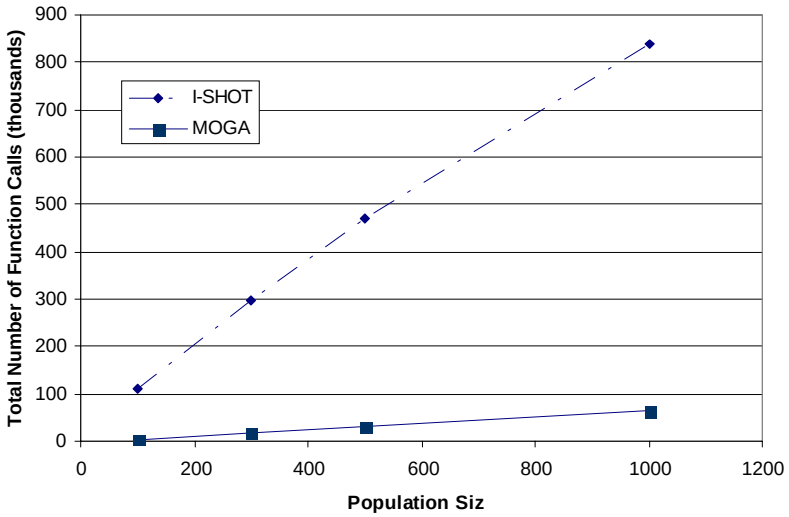

Figure 7: Two Bar Truss Function Call Comparison

Example 2: Vibrating Platform

The *vibrating platform example* is shown in Figure 8 below. This problem used in this paper was adopted from Messac (1996) with some modifications. It consists of a pinned-pinned sandwich beam with a vibrating motor on top. The sandwich has 5 layers of 3 different materials. There is a middle layer, and two sandwich layers. The distance from the center of the beam to the outer edge of each layer comprises three of the sizing design variables, d_1, d_2, and d_3. The width of the beam, b, and the length of the beam, L, are the other two sizing variables. Finally, there are three combinatorial variables for the material type M_i, where i=1, 2, 3, for the different materials that can be used for each layer. So, there are 8 design variables, 3 combinatorial variables for the material types of the 3 layers, and 5 sizing variables. The mass density, ρ, Young's modulus, E, and cost per unit volume, c, for each material are given in Table 2.

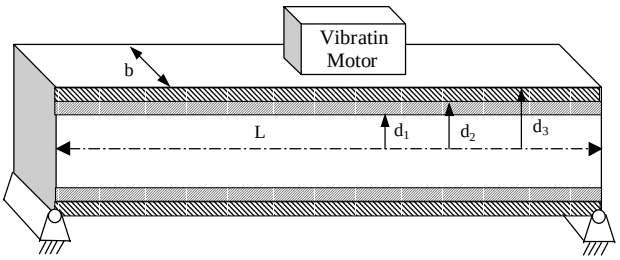

Figure 8: Vibrating Platform Apparatus

Material M_i	ρ_I(kg/m^3)	E_i(N/m^2)	C_i($/volume)
1	100	1.6×10^9	500
2	2,770	70×10^9	1,500
3	7,780	200×10^9	800

Table 2: Beam Layer Material Properties

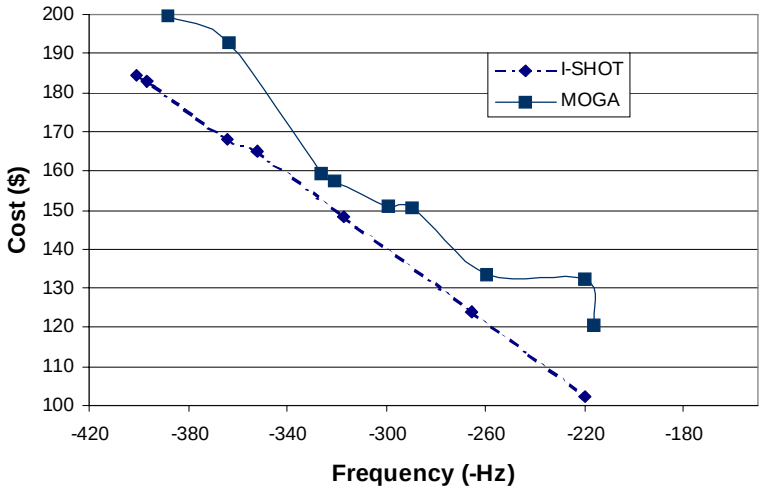

Figure 9: Direct Comparison between Solutions

The two design objectives are to maximize the natural frequency of the beam, and to minimize the material cost. The maximization of the fundamental frequency will be converted to a minimization form by minimizing the negative of the fundamental frequency. The problem formulation is shown below

$$\text{Minimize} \quad f_1(d_1,d_2,d_3,b,L,M_i) = -\left(\frac{\pi}{2L^2}\right)\left(\frac{EI}{\mu}\right)^{1/2}$$

$$EI = \left(\frac{2b}{3}\right)\left[E_1 d_1^3 + E_2(d_2^3 - d_1^3) + E_3(d_3^3 - d_2^3)\right]$$

$$\mu = 2b\left[\rho_1 d_1 + \rho_2(d_2 - d_1) + \rho_3(d_3 - d_2)\right]$$

$$\text{Minimize} \quad f_2(d_1,d_2,d_3,b,M_i) = 2b\left[c_1 d_1 + c_2(d_2 - d_1) + c_3(d_3 - d_2)\right]$$

$$\text{Subject to :} \quad g_1 : \ \mu L - 2800 \le 0$$

$$g_2 : \ d_2 - d_1 - 0.15 \le 0$$

$$g_3 : \ d_3 - d_2 - 0.01 \le 0$$

$$0.05 \le d_1 \le 0.5$$

$$0.2 \le d_2 \le 0.5$$

$$0.2 \le d_3 \le 0.6$$

$$0.35 \le b \le 0.5$$

$$3 \le L \le 6$$

Here, E_i is the modulus of elasticity of laye i=1,2,3, ρ_i is the density of laye i, c_i is the cost of laye i. Each of these parameters can be for any one of the three materials, depending on the material type variable M_i. It is assumed that the material types for the three layers are mutuall exclusive. In other words, the same material cannot be used for more than one layer. However, the layers are allowed to have zero thickness. The first three constraints refer to uppe bounds on the mass of the beam, thickness of layer 2, and thickness of layer 3, respectively, and they are labeled g_1 through g_3. The last 5 constraints are the set constraints on the sizing variables.

Comparison of Vibrating Platform Solutions

The results for the vibrating platform are shown in Figure 9. The lines connecting the points are fo visualization purposes only and do not represent the actual Pareto frontiers. The population sizes for MOGA and I-SHOT are 120 and 450 respectively. The number of bits used to represent each variable was the same for both methods, 7. For this example, MOGA again required considerably less function calls per Pareto point (406 compared to 7909 for I-SHOT), but the Pareto set is not quite as good. The processing time was not compared because different parameters were needed to generate each of the Pareto sets. To objectively compare processing time, solution sets of comparable quality should be used, which is not the case here. To generate the solution set using the MOGA method, some difficulties were encountered. The following parameters were varied several times before an acceptable Pareto solution was generated:

- Population size.
- Number individuals to replace per generation.
- Mutation probability.

This illustrates one of the advantages of the I-SHOT method, its ease of application. Generating the solution using the I-SHOT method worked for a wider range of the GA parameter values. Even though it took a couple of trials to find the Pareto frontier using I-SHOT, its first few were a relativel good approximation. However, the first few trials of MOGA gave very little information about the general form of the solution, and if the I-SHOT solution did not already exist, the decision maker might not realize that the MOGA solutions generated are in fact Pareto solutions.

CONCLUSION

Each of the two methods, I-SHOT and MOGA, demonstrated have merit worthy of mentioning. Neithe method outperforms the other on all aspects, but each method has its own strong points for certain types of problems. In general, the greatest advantage in using GA-based optimization is the wide range of applicability. Both of these methods demonstrate this advantage by generating design solutions for the mixed-discrete problem (see, example 2: the vibrating platform). For the two-bar truss problem, the MOGA method has the greatest efficiency. By working *within* the GA, MOGA generated the solution set all at once. That is, one population was allowed to evolve into the eventual Pareto set. However, for certain complex problems requiring larger population sizes, as in the vibrating platform example, the modified GA in MOGA was unable to generate as good of a representation of the Pareto set as did the I-SHOT method. The I-SHOT method works *with* the GA, thereby generating solutions one at a time. That is, a population is formed and evolved for each Pareto point generated. Also, the I-SHOT method allows greater control over the optimization process. The number of Pareto points to be generated can be chosen, a priori, and the bounds can be reduced to refine the Pareto set, whereas the number of points generated using MOGA cannot be selected upfront.

Modifications of the GA used in the MOGA method could produce a more reliable generation of the Pareto set, and this is a topic for future work. Also, some further work on allowing information feedback for the I-SHOT method may result in a more efficient generation of the Pareto set. Both methods leave some areas for improvement that will be covered as part of our future work.

ACKNOWLEDGMENT

The work upon which this paper is based on was supported in part by ONR (N000149710688, N000149810842), and NSF (DMI9700059). Such support does not constitute an endorsement of the opinions expressed in the paper by the funding agency. The input of Ms. Jin Wu who helped us with obtaining the MOGA results is also acknowledged.

REFERENCES

Changkong V. and Haimes Y.Y., 1983, Multiobjective Decision Making: Theory and Methodology, Elsevier Science Publishing Co. Inc., New York.

Eschenauer, H., Koski, J., and Osyczka, A.,(Editors), 1990, Multicriteria Design Optimization, Springer-Verlag, Ne York.

Fonseca, C.M. and Fleming, P.J., 1998 , "Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms-Part I: A unified formulation," IEEE Trans. Systems, Man, Cyber., 28, (1), 26-37.

Goldberg, D. E., 1989, "Genetic Algorithms in Search, Optimization, and Machine Learning," Addison-Wesley, Reading, Mass.

Haimes, Y. Y., Lasdon, L.S., and Wismer, D. A., 1971, "On a Bicriterion Formulation of the Problems of Integrated System Identification and System Optimization," IEEE Trans. on Systems, Man and Cybernetics SMC-1, 296-297.

Hwang C.L., and Masud, A.S.M., 1979, Multiple Objective Decision Making, Methods and Applications, Springer-Verlag, Berlin.

Kirsch, U., 1981, Optimal Structural Design, McGraw-Hill Co., New York.

Levine, D., 1996, "PGA Pack Parallel Genetic Algorithm Library," Mathematics and Computer Science, Argonne National Lab, Argonne, IL.

Messac, A., 1996, "Physical Programming: Effective Optimization for Computational Design," AIAA Journal 34, (1), 149-158.

Miettinen, K.M., 1999, Nonlinear Multiobjective Optimization, Kluwer Academic Publishers, Boston.

Narayanan, S., and Azarm, S., 1999a, "A Multiobjective Interactive Sequential Hybrid Optimization Technique fo Design Decision Making," Engineering Optimization (accepted).

Narayanan, S., and Azarm, S., 1999b, "On Improving Multiobjective Genetic Algorithms for Design Optimization," Structural Optimization (accepted).

Palli, N., Azarm, S., McCluskey, P., and Sundararajan, R., 1998, "An Interactive Multistage ε-Inequality Constraint Method for Multiple Objectives Decision Making," Trans. of ASME, Journal of Mechanical Design, 120, (4), pp. 678-686.

Papalambros, P.Y., and Wilde, D.J., 1988, Principles of Optimal Design, Cambridge University Press, New York.

Stadler, W., 1979, "A Survey of Multicriteria Optimization o the Vector Maximum Problem, Part I: 1776-1960," Journal of Optimization Theory and Application, 29, (1), 1-52.

Steuer R. E., 1986, Multiple Criteria Optimization, Theory Computation and Applications, John Wiley & Sons, Inc., New York.