Non-linear Goal Programming Using Multi-Objective Genetic Algorithms

Kalyanmoy Deb*
Kanpur Genetic Algorithms laboratory (KanGAL)
Department of Mechanical Engineering
Indian Institute of Technology, Kanpur
Kanpur, PIN 208 016, India
E-mail: deb@iitk.ac.in

Technical Report No. CI-60/98
October 1998
Department of Computer Science/XI
University of Dortmund, Germany

Abstract

Goal programming is a technique often used in engineering design activities primarily to find a compromised solution which will simultaneously satisfy a number of design goals. In solving goal programming problems, classical methods reduce the multiple goal-attainment problem into a single objective of minimizing a weighted sum of deviations from goals. Moreover, in tackling non-linear goal programming problems, classical methods use successive linearization techniques, which are sensitive to the chosen starting solution. In this paper, we pose the goal programming problem as a multi-objective optimization problem of minimizing deviations from individual goals. This procedure eliminates the need of having extra constraints needed with classical formulations and also eliminates the need of any user-defined weight factor for each goal. The proposed technique can also solve goal programming problems having non-convex trade-off region, which are difficult to solve using classical methods. The efficacy of the proposed method is demonstrated by solving a number of non-linear test problems and by solving an engineering design problem. The results suggest that the proposed approach is an unique, effective, and most practical tool for solving goal programming problems.

Keywords: Goal programming, Genetic algorithms, Engineering design

1 Introduction

Developed in the year 1955, goal programming method has enjoyed innumerable applications in engineering design activities[1−5]. Goal programming is different in concept from non-linear programming or optimization techniques in that the goal programming attempts to find one or more solutions which satisfy a number of goals to the extent possible. Instead of finding solutions which absolutely minimize or maximize objective functions, the task is to find solutions that, if possible, satisfy a set of goals, otherwise, violates the goals minimally. This makes the approach more appealing to practical designers compared to optimization methods.

The most common approach to classical goal programming techniques is to construct a non-linear programming problem (NLP) where a weighted sum of deviations from targets is minimized[6]. The NLP problem also contain a constraint for each goal, restricting the corresponding criterion function value to be within the specified target values. A major drawback with this approach is that it requires the user to specify a set of weight factors, signifying the relative importance of each criterion. This makes the approach subjective to the user. Moreover, the weighted goal programming approach has difficulty in finding solutions in problems having non-convex feasible decision space. Although there exists other methods such as lexicographic goal

*Presently visiting Computer Science Department/LS11, University of Dortmund, Germany (deb@ls11.informatik.uni-dortmund.de) programming or minimax goal programming[6,7], these methods are also not free from the dependence on the relative weight factor for each criterion function.

In this paper, we suggest using a multi-objective genetic algorithm (GA) to solve the goal programming problem. In order to use a multi-objective GA, each goal is converted into an equivalent objective function. Unlike the weighted goal programming method, the proposed approach does not add any artificial constraint into its formulation. Since, multi-objective GAs have been shown to find multiple Pareto-optimal solutions[8,9], the proposed approach is likely to find multiple solutions to the goal programming problem, each corresponding to a different setting of the weight factors. This makes the proposed approach independent from the user. Moreover, since no explicit weight factor for each criterion is used, the method is also not likely to have any difficulty in finding solutions for problems having non-convex feasible decision space.

It is worthwhile to highlight here that the use of a multi-objective optimization technique to solve goal programming problems is not new[8,9] and is novel. But the inefficiency of classical non-linear multi-objective optimization methods has led the researchers and practitioners to only concentrate on solving linear goal programming problems. Although there are some attempts to use sequential linear goal programming approaches, where a linear approximation of the non-linear problem is solved sequentially, the methods have not been successful[6]. Multi-objective GAs are around for last five years or so and have been shown to solve various non-linear multi-objective optimization problems successfully[10−13]. As a result of these interests, there exist now a number of multi-objective GA implementations[8,9,14−16]. In this paper, we show how one such GA implementation can make the non-linear goal programming easier and practical to use.

In the remainder of the paper, we briefly discuss the concept of goal programming and argue why classical goal programming methods are not adequate tools. Thereafter, we present the working principle of one multi-objective GA implementation—non-dominated sorting GA (NSGA). The usefulness of the proposed approach is demonstrated by solving five different test problems and an engineering design problem using NSGA.

2 Goal Programming

Goal programming was first introduced in an application of a single-objective linear programming problem by Charnes, Cooper, and Ferguson[1]. However, goal programming gained popularity after the works of Ignizio[4], Lee[17], and others. Romero[6] presented a comprehensive overview and listed a plethora of engineering applications where goal programming technique has been used. The main idea in goal programming is to find solutions which attain a pre-defined target for one or more criterion function. If there exists no solution which achieves targets in all criterion functions, the task is to find solutions which minimize deviations from targets. Goal programming is different from non-linear programming problems (NLPs), where the main idea is to find solutions which optimizes one or more criteria[18,19]. There is no concept of a goal in a mathematical programming problem. We illustrate the concept of goal programming by considering a single-criterion problem.

Let us consider a design criterion $f(\vec{x})$, which is a function of a solution vector $\vec{x}$. In the context of NLP, the objective is to find the solution vector $\vec{x}^*$ which will minimize or maximize $f(\vec{x})$. Without loss of generality, we consider criterion functions which is to be minimized (such as fabrication cost of an engineering component). In most design problems, there exists a number of constraints which make a certain portion ($\vec{x} \in \mathcal{F}$) of the search space feasible. It is imperative that the optimal solution $\vec{x}^*$ is feasible, that is, $\vec{x}^* \in \mathcal{F}$. In a goal programming, a target value t is chosen for every design criterion. One of the design goals may be to find a solution which attains a cost of t:

$$\begin{aligned} &\text{goal } (f(\vec{x}) = t)\,, \\ &\vec{x} \in \mathcal{F}. \end{aligned} \tag{1}$$

If the target cost t is smaller than the minimum possible cost $f(\vec{x}^*)$, naturally there exists no feasible solution which will attain the above goal exactly. The objective of goal programming is then to find that solution which will minimize the deviation d between the achievement of goal and the aspiration target, t. The solution for this problem is still $\vec{x}^*$ and the overestimate is $d = f(\vec{x}^*) - t$. Similarly, if target cost t is larger than the maximum feasible cost $f_{\max}$, the solution of the goal programming problem is $\vec{x}$ which makes $f(\vec{x}) = f_{\max}$.

However, if the target cost t is within $[f(\vec{x}^*), f_{\max}]$, the solution to the goal programming problem is that feasible solution $\vec{x}$ which makes the criterion value exactly equal to t. Although this solution may not be the optimal solution of the constrained $f(\vec{x})$, this solution is the outcome of the above goal program.

In the above example, we have considered a single-criterion problem. Goal programming is hardly used for single criterion problems. In fact, goal programming brings interesting scenarios when multiple criteria are considered. In the above example, an 'equal-to' type goal is discussed. However, there can be four different types of goal criteria, as shown below[7]:

1. Less-than-equal-to ($f(\vec{x}) \leq t$),
2. Greater-than-equal-to ($f(\vec{x}) \geq t$),
3. Equal-to ($f(\vec{x}) = t$), and
4. Within a range ($f(\vec{x}) \in [t^l, t^u]$).

In order to tackle above goals, usually two non-negative deviation variables (n and p) are introduced. For the less-than-equal-to type goal, the positive deviation p is subtracted from the criterion function, so that $f(\vec{x}) - p \leq t$. Here, the deviation p quantifies the amount by which the criterion value has surpassed the target t. The objective of goal programming is to minimize the deviation p so as to find the solution for which the deviation is minimum. If $f(\vec{x}) > t$, the deviation p should take a non-zero positive value, otherwise it must be zero. For the greater-than-equal-to type goal, a negative deviation n is added to the criterion function, so that $f(\vec{x}) + n \geq t$. The deviation n quantifies the amount by which the criterion function has not satisfied the target t. Here, the objective of goal programming is to minimize the deviation n. For $f(\vec{x}) < t$, the deviation n should take a nonzero positive value, otherwise it must be zero. For the equal-to type goal, the criterion function needs to have the target value t, thus both positive and negative deviations are used, so that $f(\vec{x}) - p + n = t$. Here, the objective of goal programming is to minimize the summation $(p + n)$, so that the obtained solution is minimally away from the target in either direction. If $f(\vec{x}) > t$, the deviation p should take a non-zero positive value and if $f(\vec{x}) < t$, the deviation n should take a non-zero positive value. For $f(\vec{x}) = t$, both deviations p and n must be zero. The fourth type of goal is handled by using two constraints: $f(\vec{x}) - p \leq t^l$ and $f(\vec{x}) + n \geq t^u$. The objective here is to minimize the summation $(p + n)$. All of the above constraints can be replaced by a generic equality constraint:

$$f(\vec{x}) - p + n = t. \tag{2}$$

For a 'less-than-equal-to' type goal, the deviation n is a slack variable which makes the inequality constraint into an equality constraint[18]. For a 'range' type goal, there are two such constraints, one with t^l having p as the slack variable and the other with t^u having n as the slack variable. Thus, to solve a goal programming problem, each goal is converted into a at least one equality constraint, and the objective is to minimize all deviations p and n. Goal programming methods differ in the way the deviations are minimized. Here, we briefly discuss three popular methods, although there exists other goal programming approaches. In all methods, we assume that there are M criterion functions $f_j(\vec{x})$, each having one of the above four types of goal.

2.1 Weighted Goal Programming

A composite objective function with deviations from each of M criterion function is used, as described below:

$$\begin{array}{rll} \text{Minimize} & \sum_{j=1}^{M}(\alpha_j p_j + \beta_j n_j), & \\ \text{Subject to} & f_j(\vec{x}) - p_j + n_j = t_j, & \text{for each goal } j, \\ & \vec{x} \in \mathcal{F} & \\ & n_j, p_j \geq 0, & \text{for each goal } j. \end{array} \tag{3}$$

Here, the parameters α_j and β_j are weighting factors for positive and negative deviations of the j-th criterion function. For less-than-equal-to type goals, the parameter β_j is zero. Similarly, for greater-than-equal-to type goals, the parameter α_j is zero. For range-type goals, there exists a pair of constraints for each criterion function. Usually, the weight factors α_j and β_j are fixed by the decision-maker, which makes the method subjective to the user. We illustrate this matter through a simple example problem:

$$\begin{array}{rl} \text{goal} & (f_1 = 10x_1 \leq 2), \\ \text{goal} & (f_2 = \frac{10+(x_2-5)^2}{10x_1} \leq 2), \\ \text{Subject to} & \mathcal{F} \equiv (0.1 \leq x_1 \leq 1, \quad 0 \leq x_2 \leq 10). \end{array} \tag{4}$$

The *decision space*, which is the feasible solution space ($\vec{x} \in \mathcal{F}$) is shown in Figure 1 (shaded region). The

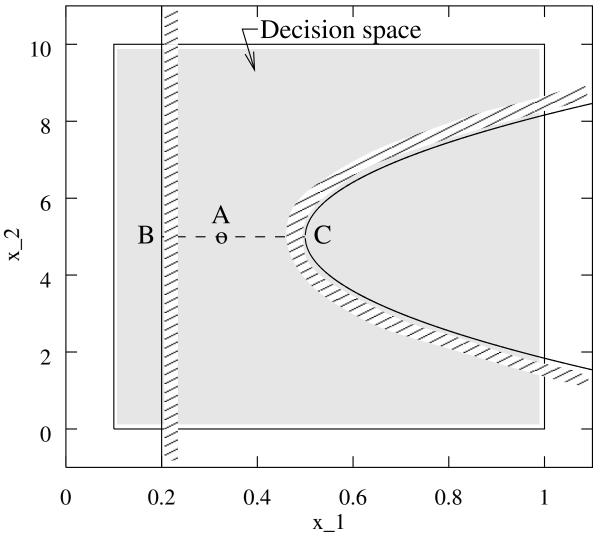

Figure 1: The goal programming problem is shown in solution space.

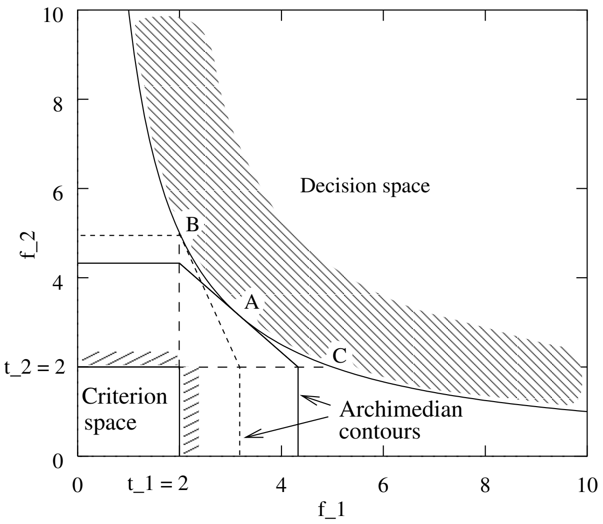

Figure 2: The goal programming problem is shown in function space.

goal lines ($f_1 \leq 2$ and $f_2 \leq 2$) are also shown. It is clear that there exists no feasible solution which achieves both goals. Figure 2 also shows that the *criterion space* and the decision space do not overlap. Thus, the resulting solution to above goal programming problem will violate either or both the above goals, but in a minimum sense. In solving this problem using the weighted goal programming, the following NLP problem is constructed:

$$\begin{array}{rl} \text{Minimize} & \alpha_1 p_1 + \alpha_2 p_2, \\ \text{Subject to} & 10x_1 - p_1 \leq 2, \\ & \frac{10+(x_2-5)^2}{10x_1} - p_2 \leq 2, \\ & 0.1 \leq x_1 \leq 1, \quad 0 \leq x_2 \leq 10, \\ & p_1, p_2 \geq 0. \end{array} \tag{5}$$

Note that in the above NLP problem the deviations n_1 and n_2 in the constraints are eliminated by using a '$\leq$' relation. Figures 1 and 2 make the concept of goal programming clear. Since no solution in the decision space lies in the criterion space, the objective of goal programming is to find that solution in the decision space which minimizes the deviation from the criterion space in both criteria. Here comes the dependence of the resulting solution on the weight factors α_1 and α_2. By choosing a value of these weight factors, one, in fact, constructs an artificial penalty function (or known as an utility function) away from the criterion space. The above formulation constructs a penalty function as shown in Figure 3(a) for each criterion. Thus, the objective $\alpha_1 p_1 + \alpha_2 p_2$ produces contours (known as Archimedian contours) as shown in Figure 2. The concept of the above minimization problem is to find the contour which touches the decision space. In other words, the minimization procedure finds that contour which has the minimum value and contains at least one solution from the decision space. If equal importance to both objectives (that is, $\alpha_1 = \alpha_2 = 0.5$) is given, the minimum contour (marked by solid lines) is shown in Figure 2 and the resulting solution (marked as 'A') is as follows:

$$x_1 = 0.3162, \quad x_2 = 5.0, \quad p_1 = p_2 = 1.162.$$

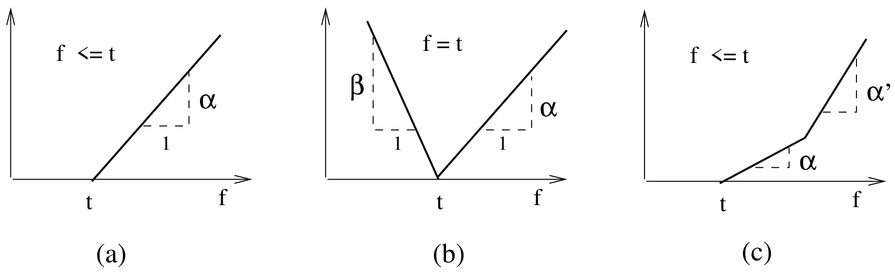

Figure 3: Different penalty functions are shown.

At this solution, the criterion function values are $f_1 = 3.162$ and $f_2 = 3.162$, thereby violating goals $f_1 \leq 2$ and $f_2 \leq 2$.

Interesting scenario emerge when different weight factors are chosen. For example, if $\alpha_1 = 1$ and $\alpha_2 = 0$ are chosen, the resulting contour is shown by a dashed line and the corresponding solution (marked as 'B') is as follows:

$$x_1 = 0.2, \quad x_2 = 5.0, \quad p_1 = 0, \quad p_2 = 3.0.$$

The corresponding criterion values are $f_1 = 2$ (no deviation from target) and $f_2 = 5.0$ (a deviation of 3.0 from target). On the other hand, if $\alpha_1 = 0$ and $\alpha_2 = 1$ are chosen, the resulting solution (marked as 'C') is as follows:

$$x_1 = 0.5, \quad x_2 = 5.0, \quad p_1 = 3.0, \quad p_2 = 0.$$

The corresponding criterion values are $f_1 = 5$ (a deviation of 3.0 from target) and $f_2 = 2$ (no deviation from target). These solutions A, B, and C are shown in Figure 1 as well. This figure shows that there exists many more such solutions that lie in the interval $0.2 \leq x_1 \leq 0.5$ and $x_2 = 5.0$, as shown in Figure 1, each one of which is the solution of the above goal programming problem for a particular set of weight factors α_1 and α_2. Thus, we observe that the solution to the goal programming problem largely depends on the chosen weight factors.

Moreover, as outlined elsewhere[6], there exists a number of other problems with the weighted goal programming method:

1. Since criterion functions $f_j(\vec{x})$ may not be commensurable with each other, the above weighted composite objective function may add 'pints of butter with kilos of potatoes'. It causes difficulty to an user in choosing an appropriate set of weight factors to get a reasonable solution to the problem.

2. Criterion functions may have different range of values, thereby giving unequal importance to all criterion functions. One remedy of this problem is to normalize the criterion functions before using equation 3, however, this approach requires knowledge of lower and upper bounds of each criterion function.

3. The deviation values p_i and n_i may not be of the same order in magnitude as the target values, thereby making some constraints difficult to satisfy.

4. Simple weighting technique mentioned above will not be able to *respond* to differing weight factors in problems having non-convex feasible decision space.

For handling other types of goals, an Archimedian contour with an equivalent penalty function is used[7]. Figure 3(b) shows the penalty function used for an 'equal-to' type goal, involving two weights α and β.

2.2 Lexicographic Goal Programming

In this approach, different goals are categorized into several levels of preemptive priorities. Goals with a lower-level priority is infinitely more important than a goal of a higher-level priority. Thus, it is important to fulfill the goals of first level priority before considering goals of second level of priority. Some researchers argue that such consideration of goals is most practical[4], although there exists some critics of this approach[20].

This approach formulates and solves a number of sequential goal programming problems. First, only goals and corresponding constraints of the first level priority are considered in the formulation of the goal programming problem and is solved. If there exists multiple solution to the problem, another goal programming problem is formulated with goals having the second level priority. In this case, the objective is only to minimize deviation in goals of second level priority. However, the goals of first level priority is used as hard constraints so that obtained solution does not violate the goals of first level priority. This process continues with goals of other higher level priorities in sequence. The process is terminated as soon as one of the goal programming problems results in a single solution. When this happens, all subsequent goals of higher level priorities are meaningless and are known as redundant goals[6]. Since for solving an individual goal programming requires the use of the weighted goal programming approach for nonlinear problems, this method is also not free from the subjectiveness of users and other difficulties mentioned earlier.

2.3 Minimax Goal Programming

This approach is similar to the weighted goal programming approach, but instead of minimizing the weighted sum of deviations from targets, the maximum deviation in any goal from target is minimized. The resulting nonlinear programming problem becomes as follows:

$$\begin{array}{lll} \text{Minimize} & d & \\ \text{Subject to} & \alpha_j p_j + \beta_j n_j \le d, & \text{for each goal } j, \\ & f_j(\vec{x}) - p_j + n_j = t_j, & \text{for each goal } j, \\ & \vec{x} \in \mathcal{F}, & \\ & n_j, p_j \ge 0, & \text{for each goal } j. \end{array} \tag{6}$$

Here, the parameter d becomes the maximum deviation in any goal. Once again, this method requires the choice of weight factors α_j and β_j, thereby making the approach subjective to user.

In the next section, we briefly discuss the multi-objective GAs. Thereafter, we show how multi-objective GAs can be used to solve goal programming problems which do not need any weight factor. In fact, the proposed approach simultaneously finds solutions to the same goal programming problem formed for different weight factors, thereby making the approach both practical and different from the classical approaches.

3 Multi-Objective Genetic Algorithms

Multi-objective optimization problems give rise to a set of *Pareto-optimal* solutions, none of which can be said to be better than other in *all* objectives. In any interesting multi-objective optimization problem, there exists a number of such solutions which are of interest to designers and practitioners. Since no one solution is better than any other solution in the Pareto-optimal set, it is also a goal in a multi-objective optimization to find as many such Pareto-optimal solutions as possible. Unlike most classical search and optimization problems, GAs work with a population of solutions and thus are likely (and unique) candidates for finding multiple Pareto-optimal solutions simultaneously[8,9,14–16]. There are two tasks that are achieved in a multi-objective GA:

1. Convergence to the Pareto-optimal set, and

2. Maintenance of diversity among solutions of the Pareto-optimal set.

GAs with suitable modification in their operators have worked well to solve many multi-objective optimization problems with respect to above two tasks. Most multi-objective GAs work with the concept of *domination*. In the following, we first define domination between two solutions and then describe one multi-objective GA implementation in brief.

For a problem having more than one objective function (say, f_j, $j = 1, \ldots, M$ and $M > 1$), a solution $x^{(1)}$ is said to *weakly* dominate the other solution $x^{(2)}$, if both the following conditions are true[7].

1. The solution $\vec{x}^{(1)}$ is no worse (say the operator $\prec$ denotes worse and $\succ$ denotes better) than $\vec{x}^{(2)}$ in all objectives, or $f_j(\vec{x}^{(1)}) \not\prec f_j(\vec{x}^{(2)})$ for all $j = 1, 2, \ldots, M$ objectives.

2. The solution $\vec{x}^{(1)}$ is strictly better than $\vec{x}^{(2)}$ in at least one objective, or $f_{\bar{j}}(\vec{x}^{(1)}) \succ f_{\bar{j}}(\vec{x}^{(2)})$ for at least one $\bar{j} \in \{1, 2, \ldots, M\}$.

With these conditions, it is clear that in a population of N solutions, the set of non-dominated solutions are likely candidates to be the members of the Pareto-optimal set. It is important to mention here that if the first condition is satisfied with strict $f_j(\vec{x}^{(1)}) \succ f_j(\vec{x}^{(2)})$, then the solution $\vec{x}^{(1)}$ is said to *strongly* dominate the solution $\vec{x}^{(2)}$. In the following, we describe one implementation of a multi-objective GA which attempts to find the best set of non-dominated solutions in the search space.

3.1 Non-dominated Sorting GA (NSGA)

NSGA used here is a real-parameter GA which works directly with the parameter values. We outline the NSGA procedure in Figure 4 and then describe each operator in succession. The variable pop is an array of

```
Generate_initial_population(pop);
if (not terminate)
  Non-dominated_ranking(pop);
  Fitness_assignment(pop);
  for i=1 to N step 2
    parent1 = Selection(pop);
    parent2 = Selection(pop);
    (child1,child2)=Crossover(parent1,
                              parent2);
    newpop_i   = Mutation(child1);
    newpop_i+1 = Mutation(child2);
  pop = newpop;
print non-dominated solutions;
```

Figure 4: A flowchart of NSGA.

dimension N (population size), each member of which is a P-dimensional solution vector. NSGA begins with randomly created solution vector $x_i \in [x_i^l, x_i^u]$. In the following, we describe how a population of solutions is first ranked according to non-domination and then discuss the fitness assignment procedure.

3.1.1 Non-Dominated Ranking

Consider a set of N population members, each having M (> 1) objective function values. The following procedure can be used to find the non-dominated set of solutions:

Step 0: Begin with $i = 1$.

Step 1: For all $j = 1, \ldots, N$ and $j \neq i$, compare solutions $\vec{x}^{(i)}$ and $\vec{x}^{(j)}$ for domination using two conditions for all M objectives.

Step 2: If for any j, $\vec{x}^{(i)}$ is dominated by $\vec{x}^{(j)}$, mark $\vec{x}^{(i)}$ as 'dominated'.

Step 3: If all solutions (that is, when $i = N$ is reached) in the set are considered, Go to Step 4, else increment i by one and Go to Step 1.

Step 4: All solutions that are not marked 'dominated' are non-dominated solutions.

All these non-dominated solutions are assumed to constitute the first non-dominated front in the population. These solutions are temporarily ignored from the population and the above step-by-step procedure is applied again. The resulting non-dominated solutions are assumed to constitute the second non-dominated front. This procedure is continued until all population members are assigned a front. Thus, a population of size N could be classified into a minimum of one non-dominated front of size N or a maximum of N non-dominated fronts of size one.

An aspect of this method is that practically any number of objectives can be used. Both minimization and maximization problems can also be handled by this algorithm. The only place a change is required for the above two cases is the way the non-dominated solutions are identified (according to the conditions for dominance presented earlier). The next step is to assign fitness to each solution in the population.

3.1.2 Fitness Assignment

Since all solutions in a particular non-dominated front are equally important, all are assigned the same fitness value. We begin with solutions of the first non-dominated front. A large *dummy* fitness value (equal to N) is assigned to each non-dominated solution of the first front. However, in order to maintain diversity among solutions, these non-dominated solutions are then *shared* with their dummy fitness values. Sharing method is discussed briefly in the next subsection. Sharing is achieved by dividing the dummy fitness value of an individual by a quantity (called the niche count) proportional to the number of individuals around it. This procedure causes multiple optimal solutions to co-exist in the population. The worst shared fitness value in the solutions of the first non-dominated front is noted for further use.

A dummy fitness value, a little smaller than the worst shared fitness value observed in solutions of first non-dominated set, is assigned to all members of the second non-dominated front. Thereafter, the sharing procedure is performed among the solutions of the second non-domination front and shared fitness values are found as before. This process is continued till all population members are assigned a shared fitness value.

3.1.3 Sharing procedure

Given a set of n_k solutions in the k-th non-dominated front each having a dummy fitness value f_k, the sharing procedure is performed in the following way for each solution $i = 1, 2, \ldots, n_k$:

Step 1: Compute a normalized Euclidean distance measure with another solution j in the k-th non-dominated front, as follows:

$$d_{ij} = \sqrt{\sum_{p=1}^{P} \left(\frac{x_p^{(i)} - x_p^{(j)}}{x_p^u - x_p^l} \right)^2},$$

where P is the number of variables in the problem. The parameters x_p^u and x_p^l are the upper and lower bounds of variable x_p.

Step 2: This distance d_{ij} is compared with a pre-specified parameter σ_{share} and the following *sharing function* value is computed (Deb and Goldberg, 1989):

$$Sh(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{\text{share}}}\right)^2, & \text{if } d_{ij} \le \sigma_{\text{share}}, \\ 0, & \text{otherwise.} \end{cases}$$

Step 3: Increment j. If $j \le n_k$, go to Step 1 and calculate $Sh(d_{ij})$. If $j > n_k$, calculate niche count for i-th solution as follows:

$$m_i = \sum_{j=1}^{n_k} Sh(d_{ij}).$$

Step 4: Degrade the dummy fitness f_k of i-th solution in the k-th non-domination front to calculate the shared fitness, f_i', as follows:

$$f_i' = \frac{f_k}{m_i}.$$

This procedure is continued for all $i = 1, 2, \ldots, n_k$ and a corresponding f'_i is found. Thereafter, the smallest value $f_k^{\min}$ of all f'_i in the k-th non-dominated front is found for further processing. The dummy fitness of the next non-dominated front is assigned to be $f_{k+1} = f_k^{\min} - \epsilon_k$, where ϵ_k is a small positive number.

The above sharing procedure requires a pre-specified parameter σ_{share}, which can be calculated as follows[21]:

$$\sigma_{\text{share}} \approx \frac{0.5}{\sqrt[p]{q}}, \tag{7}$$

where q is the desired number of distinct Pareto-optimal solutions. Although the calculation of σ_{share} depends on this parameter q, it has been been shown elsewhere[9] that the use of above equation with $q \approx 10$ works in many test problems. Moreover, the performance of NSGAs is not very sensitive to this parameter near σ_{share} values calculated using $q \approx 10$.

3.1.4 Selection Operator

After all solutions are assigned a fitness, selection operator is used to find above-average solution stochastically. A stochastic remainder proportionate selection[22] is used with the fitness values, where a solution is selected as a parent in proportion to its fitness value. With such a operator, solutions of the first non-dominated front have higher probability of being a parent than solutions of other fronts. This is intended to search for non-dominated regions, which will finally lead to the Pareto-optimal front. This results in quick convergence of the population towards non-dominated regions and sharing procedure helps to distribute it over this region. Thus, selection operator helps to emphasize better solutions in the population, but does not help to create new solutions, a matter which is performed in the following two operators.

3.1.5 Crossover Operator

Two parent solutions $\vec{x}^{(1)}$ and $\vec{x}^{(2)}$ obtained from selection operator are crossed with a probability $p_c = 0.9$. For a crossover, the solutions are crossed variable-by-variable to create two new children solutions $\vec{y}^{(1)}$ and $\vec{y}^{(2)}$. Depending on the relative distance between the parent parameter values, children solutions are created by using a polynomial probability distribution[23]. Each variable is crossed with a probability of 0.5 using the following step-by-step procedure:

Step 1: Create a random number u between 0 and 1.

Step 2: Calculate β_q as follows:

$$\beta_q = \begin{cases} (u\alpha)^{\frac{1}{\eta_c+1}}, & \text{if } u \leq \frac{1}{\alpha}, \\ \left(\frac{1}{2-u\alpha}\right)^{\frac{1}{\eta_c+1}}, & \text{otherwise}, \end{cases} \tag{8}$$

where $\alpha = 2 - \beta^{-(\eta_c+1)}$ and β is calculated as follows:

$$\beta = 1 + \frac{2}{x_i^{(2)} - x_i^{(1)}} \min[(x_i^{(1)} - x_i^l),\ (x_i^u - x_i^{(2)})],$$

where x_i^l and x_i^u are lower and upper bounds of parameter x_i. The parameter η_c is the distribution index and can take any non-negative value. A small value of η_c allows solutions far away from parents to be created as children solutions and a large value restricts only near-parent solutions to be created as children solutions. In all simulations, we use $\eta_c = 30$.

Step 3: The children solutions are then calculated as follows:

$$\begin{aligned} y_i^{(1)} &= 0.5\left[(x_i^{(1)} + x_i^{(2)}) - \beta_q |x_i^{(2)} - x_i^{(1)}|\right], \\ y_i^{(2)} &= 0.5\left[(x_i^{(1)} + x_i^{(2)}) + \beta_q |x_i^{(2)} - x_i^{(1)}|\right]. \end{aligned}$$

3.1.6 Mutation Operator

A polynomial probability distribution[24] is used to create a solution $z_i^{(j)}$ in the vicinity of a parent solution $y_i^{(j)}$. The following procedure is used for each variable with a probability p_m:

Step 1: Create a random number u between 0 and 1.

Step 2: Calculate the parameter δ_q as follows:

$$\delta_q = \begin{cases} \left[2u + (1-2u)(1-\delta)^{\eta_m+1}\right]^{\frac{1}{\eta_m+1}} - 1, \\ \quad \text{if } u \leq 0.5, \\ 1 - \left[2(1-u) + 2(u-0.5)(1-\delta)^{\eta_m+1}\right]^{\frac{1}{\eta_m+1}}, \\ \quad \text{otherwise}, \end{cases} \tag{9}$$

where $\delta = \min[(y_i^{(j)} - y_i^l),\ (y_i^u - y_i^{(j)})]/(y_i^u - y_i^l)$. The parameter η_m is the distribution index for mutation and takes any non-negative value. We use $\eta_m = 100 + t$ (where t is the iteration number) here.

Step 3: Calculate the mutated child as follows:

$$z_i^{(j)} = y_i^{(j)} + \delta_q (y_i^u - y_i^l).$$

The mutation probability p_m is linearly varied from $1/P$ till 1.0, so that, on an average, one parameter gets mutated in the beginning and all parameters get mutated at the end of a simulation run.

4 Suggested Technique

The objective in the above multi-objective GA is to find non-dominated solutions in the search space defined by the minimality or maximality conditions of the objectives. However, the above multi-objective GA can be used to solve goal programming problems. Here, we discuss a couple of changes that are necessary for this purpose.

4.1 Formulate Objective Functions from Goals

The goals are converted to objective functions of minimizing the deviations. The conversion procedure depends on the type of goals used. We present them in the following table.

Type	Goal	Objective function
$\leq$	goal $(f_j(\vec{x}) \leq t_j)$	Minimize $\langle f_j(\vec{x}) - t_j \rangle$
$\geq$	goal $(f_j(\vec{x}) \geq t_j)$	Minimize $\langle t_j - f_j(\vec{x}) \rangle$
$=$	goal $(f_j(\vec{x}) = t_j)$	Minimize $\lvert f_j(\vec{x}) - t_j \rvert$
Range	goal $(f_j(\vec{x}) \in [t_j^l, t_j^u])$	Min. $\max(\langle t_j^l - f_j(\vec{x}) \rangle, \langle f_j(\vec{x}) - t_j^u \rangle)$

Here the bracket operator $\langle\ \rangle$ returns the value of the operand if the operand is positive, otherwise returns zero. This way a goal programming problem of various kinds is formulated as a multi-objective problem. Although similar other methods have been suggested in classical goal programming texts[6,7], the advantage with the above formulation is that (i) there is no need of any additional constraint for each goal, and (ii) since GAs do not require objective functions to be differentiable, the above objective function can be used.

Although somewhat obvious, we shall show that the NLP problem of solving the weighted goal programming for a fixed set of weight factors is exactly the same as solving the above reformulated problem. We shall only consider the 'less-than-equal-to' type goal, however the same conclusion can be made for other types of goal, as well. Consider the a goal programming problem having one goal of finding solutions in the feasible space $\mathcal{F}$ for which the criterion $f(x) \leq t$. We use equation 3 to construct the corresponding NLP problem:

$$\begin{array}{ll} \text{Minimize} & p \\ \text{Subject to} & f(\vec{x}) - p \leq t, \\ & p \geq 0, \\ & \vec{x} \in \mathcal{F}. \end{array} \tag{10}$$

We can rewrite both constraints involving p as $d \geq \max(0, f(\vec{x}) - t)$. When $(f(\vec{x}) - t)$ is negative, the above problem has the solution $d = 0$ and when $(f(\vec{x}) - t)$ is positive, the above problem has the solution $d = f(\vec{x}) - t$. This is exactly what is achieved by simply solving the problem: Minimize $\langle f(\vec{x}) - t \rangle$.

Since we now have a way to convert a goal programming problem into an equivalent multi-objective problem, we can use multi-objective GAs to solve the goal programming problem. In certain cases, there may exist a unique solution to a goal programming problem, no matter what weight factors are chosen. In such cases, the equivalent multi-objective optimization problem is similar to a problem without conflicting objectives and the resulting Pareto-optimal set contains only one solution. However, in most cases, goal programming problems are sensitive to the chosen weight factors and resulting solution to the problem largely depends on specific weight factors used. The advantage of using the multi-objective reformulation is that each Pareto-optimal solution corresponding to the multi-objective problem becomes the solution of the original goal programming problem for a specific set of weight factors. Thus, by using multi-objective GAs, we can get multiple solutions to the goal programming problem simultaneously, which are not subjective to the user.

4.2 Using Weak Condition for Dominance

Most multi-objective search and optimization algorithms use the weak condition for dominance[6] presented earlier. In such cases, a solution $\vec{x}^{(1)}$ need not be better than $\vec{x}^{(2)}$ in all objectives to dominate. In the least, if $\vec{x}^{(1)}$ is better than $\vec{x}^{(2)}$ in only one objective and is equal to $\vec{x}^{(2)}$ in all other objectives, then $\vec{x}^{(1)}$ dominates $\vec{x}^{(2)}$. NSGA finds non-dominated solutions in a population by eliminating all dominated solutions. Thus, checking with the weak condition of dominance will allow more solutions to be qualified as dominated solutions than checking with the string condition of dominance. For the non-dominated solutions, the opposite is true. Thus, using the weak condition for dominance allows more *strict* non-dominated solutions to be retained.

With the above formulation of objective functions from goals, it is clear that there will be many solutions for which the formulated objective value is zero (for example, in $\leq$-type goal all solutions having $f_j - t_j \leq 0$). Since in goal programming these solutions are not of much interest, a weak dominance condition will not include these solutions in the non-dominated set.

4.3 Estimating Relative Weights

After multiple solutions are found, designers can then use higher-level decision-making approaches or compromise programming[25] to choose one particular solution. Each solution $\underline{\vec{x}}$ can be analyzed to find the relative importance of each criterion function as follows:

$$w_j = \frac{|t_j| / |f_j(\underline{\vec{x}}) - t_j|}{\sum_{i=1}^{M} |t_i| / |f_i(\underline{\vec{x}}) - t_i|}. \tag{11}$$

For a 'range' type goal, the target t_j can be substituted by either t_j^l or t_j^u depending on which is closer to $f(\underline{\vec{x}})$.

Moreover, the proposed approach also does not pose any other difficulties that the weighted goal programming method has. Since solutions are compared criterion-wise, there is no danger of comparing butter with potatoes, nor there is any difficulty of scaling in criterion function values. Furthermore, we shall show in the next section that this approach allows to find critical solutions to non-convex goal programming problems, which are difficult to find using the weighted goal programming method.

5 Proof-of-Principle Results

In order to show the working of the proposed approach, we first solve a number of test problems. In the next section, we shall apply the technique on an engineering design problem. In all test problems, we have used NSGAs described in Section 3.1, although any other multi-objective GA implementations with the reformulation suggested earlier can be used.

5.1 Test Problem P1

We first consider the example problem given in equation 4. The goal programming problem is converted into a two-objective optimization problem as follows:

$$\begin{array}{rl} \text{Minimize} & \langle f_1(x_1, x_2) - 2 \rangle, \\ \text{Minimize} & \langle f_2(x_1, x_2) - 2 \rangle, \\ \text{Subject to} & \mathcal{F} \equiv (0.1 \leq x_1 \leq 1, \quad 0 \leq x_2 \leq 10). \end{array} \tag{12}$$

Here, the criterion functions are $f_1 = 10x_1$ and $f_2 = (10 + (x_2 - 5)^2)/(10x_1)$. We use a population of size 50 and run NSGA for 50 generations. A $\sigma_{\text{share}} = 0.158$ (equation 7 with $P = 2$ and $q = 10$) is used. As discussed earlier, the feasible decision space lies above the hyperbola. All 50 solutions in the initial population and all non-dominated solutions at the final population are shown in Figure 5, which is plotted with criterion function values f_1 and f_2. All final solutions have $\underline{x}_2 = 5$ and $0.2 \leq \underline{x}_1 \leq 0.5$. The figure also shows

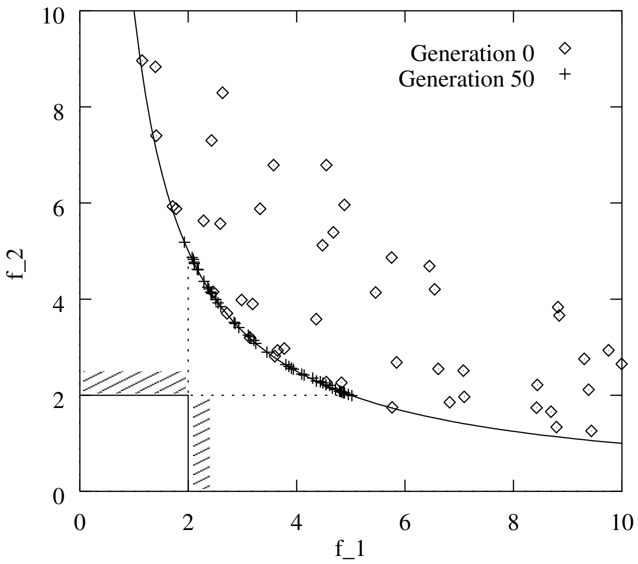

Figure 5: NSGA solutions are shown on a f_1-f_2 plot of problem P1.

marks the region (with dashed lines) of true solutions of this goal programming problem with different weight factors. The figure shows that NSGA in a single run has been able to find different solutions in the desired range. Although the other regions (for $f_1 < 2$ and $f_1 > 5$) on the hyperbola are Pareto-optimal solutions of the two-objective optimization problem of minimizing f_1 and f_2, the reformulation of the objective functions allow the NSGA to find only the required region, which are also solutions of the goal programming problem. Table 1 shows five different solutions obtained by the NSGA. Relative weight factors for each solution are also computed using equation 11. If the first criterion is of more importance, solutions in first and second row can be chosen, whereas if second criterion is of more importance solutions in fourth or fifth rows can be chosen. The solution in the third row shows a solution where both criteria are of more or less equal importance. The advantage of using the proposed technique is that all such (and many more as shown in Figure 5) solutions can be found simultaneously in one single run.

Table 1: Five solutions to the goal programming problem are shown.

$\underline{x}_1$	$\underline{x}_2$	$f_1(\vec{\underline{x}})$	$f_2(\vec{\underline{x}})$	w_1	w_2
0.2029	5.0228	2.0289	4.9290	0.9902	0.0098
0.2626	5.0298	2.6260	3.8083	0.7428	0.2572
0.3145	5.0343	3.1448	3.1802	0.5076	0.4923
0.3690	5.0375	3.6896	2.7107	0.2972	0.7027
0.4969	5.0702	4.9688	2.0135	0.0045	0.9955

5.2 Test Problem P2

We use the following goal programming problem:

$$\begin{array}{ll} \text{goal} & (f_1 = x_1 \geq 0.9), \\ \text{goal} & (f_2 = (1 - \sqrt{x_1(1-x_1)})(1 + 10x_2^2) = 0.55), \\ \text{Subject to} & 0 \leq x_1 \leq 1, \quad 0 \leq x_2 \leq 1. \end{array} \tag{13}$$

Here, the first goal is of 'greater-than-equal-to' type and the second goal is of 'equal-to' type. The feasible decision space is the region above the circle shown in Figure 6. The criterion space is the line AB. Since there is no feasible solution which lies in the criterion space (on the line AB), the solution to this goal programming problem is the region on the circle marked by the dashed lines. Each solution in this region corresponds to a goal programming problem with a specific set of weight factors. The solutions to this problem are $x_2 = 0$ and $0.71794 \leq x_1 \leq 0.9$, depending on the weight factors used. If weight factors $\alpha_1 = 1$ and $\alpha_2 = 0$ are chosen, the desired solution is $x_1 = 0.9$ and $x_2 = 0.7$. On the other hand, if a weight factors $\alpha_1 = 0$ and $\alpha_2 = 1$ are chosen, the solution is $x_1 = 0.71794$ and $x_2 = 0.55$. For other weight factors, intermediate points on the circle will be the resulting solution of the goal programming problem. To solve using NSGA, the above problem is converted into an equivalent two-objective optimization problem, as follows:

$$\begin{array}{ll} \text{Minimize} & \langle 0.9 - f_1(x_1, x_2) \rangle, \\ \text{Minimize} & |f_2(x_1, x_2) - 0.55|, \\ \text{Subject to} & 0 \leq x_1 \leq 1, \quad 0 \leq x_2 \leq 1. \end{array} \tag{14}$$

Same NSGA parameters as that in test problem 1 are used here. Figure 6 shows how NSGA has able to find many solutions in the desired range. The figure shows that NSGA finds many such solutions in a single run.

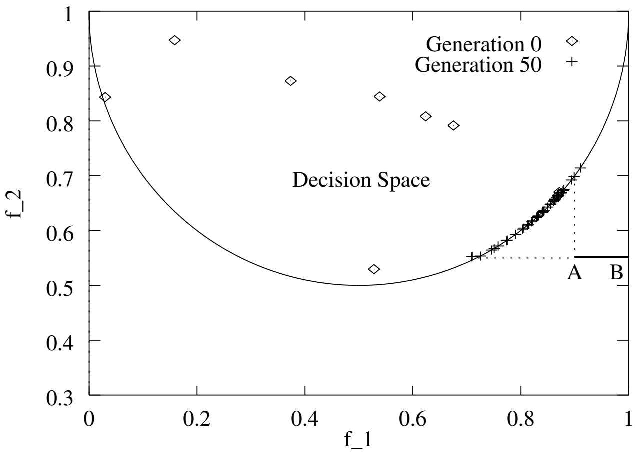

Figure 6: NSGA solutions are shown on a f_1-f_2 plot of problem P2.

All 50 initial population members could not be shown in the range of the figure, because many solutions lie outside the plotting area.

5.3 Test Problem P3

We alter the above problem to create a different goal programming problem:

$$\begin{array}{ll} \text{goal} & (f_1 = x_1 \in [0.25, 0.75]), \\ \text{goal} & (f_2 = (1 - \sqrt{x_1(1-x_1)})(1 + 10x_2^2) \leq 0.4), \\ \text{Subject to} & 0 \leq x_1 \leq 1, \quad 0 \leq x_2 \leq 1. \end{array} \tag{15}$$

The feasible decision space and the criterion space are shown in Figure 7. As evident from the figure, the solution to this problem is drastically different from that in test problem P2. There exists only one solution

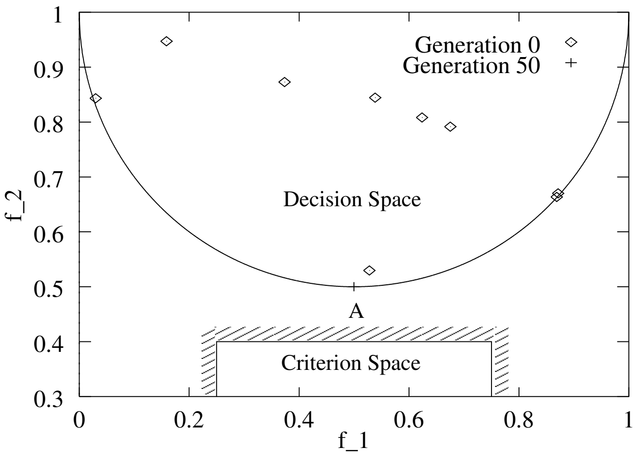

Figure 7: NSGA solutions are shown on a f_1-f_2 plot of problem P3.

($\underline{x}_1 = \underline{x}_2 = 5$) to this problem, no matter what non-zero weight factors are chosen. This is because, this solution (marked as 'A' in the figure) makes the shortest deviation from the criterion space. A NSGA with identical parameter setting and identical initial population as used in test problem P2 are used. It is observed that after 50 generations, all population members converge at the solution marked by 'A'. This test problem shows that although a multi-objective optimization technique is used, the use of reformulated objective function allows to find a single solution as the Pareto-optimal solution.

5.4 Test Problem P4

Next, we consider a goal programming problem which will cause weighted goal programming approach difficulty in finding most of the desired solutions. This problem is a simple modification to problem P3:

$$\begin{array}{ll} \text{goal} & (f_1 = x_1 \in [0.25, 0.75]), \\ \text{goal} & (f_2 = (1 + \sqrt{x_1(1-x_1)})(1 + 10x_2^2) \leq 1.25), \\ \text{Subject to} & 0 \leq x_1 \leq 1, \quad 0 \leq x_2 \leq 1. \end{array} \tag{16}$$

Figure 8 shows the decision space and the criterion space in a f_1-f_2 plot. Once again, there is no feasible solution which satisfies both goals. The solutions to the above problem lie on the circle in the region AB and CD, since each solution in the region will make the deviations from the criterion space minimum for a particular set of weight factors. With the weighted goal programming method, it is expected to find the solutions A, B, C, or D only. The Archimedian contour line with the shortest weighted deviation is shown by the dashed line. When a different combination of weight factors is chosen, a different Archimedian contour will be obtained. But any such contour cannot become tangent to any other point within AB or CD. Thus, the intermediate solutions cannot be found using the weighted goal programming approach. However, if a two-sloped penalty function (as shown in Figure 3(c)) is used, such intermediate solutions can be found. Such an Archmedian contour is shown by dotted lines. This two-slope method requires user to choose two weight factors (α and α') instead of one for each goal and makes the goal programming procedure complicated by increasing the number of constraints. Moreover, the difficulty of dependence of the resulting solution on the weight factors still remains with the latter penalty function approach.

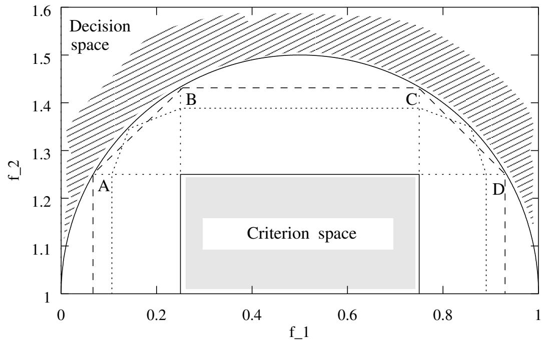

Figure 8: Criterion and decision spaces are shown for test problem P4.

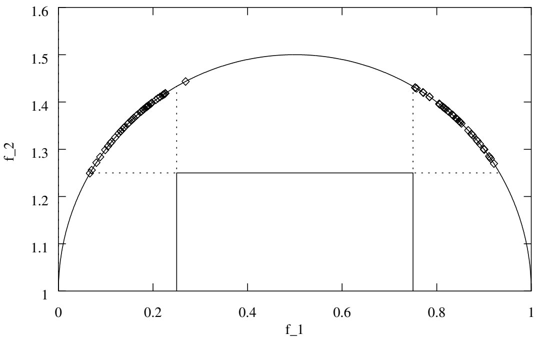

Figure 9: NSGA solutions are shown on a f_1-f_2 plot of test problem P4.

We use NSGAs with a population size of 100 and with all other parameters same as that used earlier. The solutions after 50 generations are shown in Figure 9. The figure shows that solutions in both regions AB and CD are found by the NSGA in one single run.

5.5 Test Problem P5

As the final test problem, we choose one similar to that used in Ignizio[4]:

$$\begin{aligned} &\text{goal } (f_1 = x_1 x_2 \geq 16), \\ &\text{goal } (f_2 = (x_1 - 3)^2 + x_2^2 \leq 9), \\ &\text{Subject to} \quad 6x_1 + 7x_2 \leq 42. \end{aligned} \tag{17}$$

Ignizio[4] considered the constraint as the third goal and emphasized that the first level priority in the problem is to find solutions which will satisfy the constraint. However, here we argue that such a first priority goal can be taken care of by using it as a *hard* constraint so that any solution violating the constraint will receive a large penalty. Since the constraint is explicitly taken care of, the next level priority is to find solution(s) which will minimize the deviation in two goals presented in equation 15. Figures 10 and 11 show the problem in the solution space and in the function space, respectively. The feasible search space is shown by plotting about 25,000 points in Figure 11. The figure shows that there exists no feasible solution which satisfies both goals. A population size of 100 is used and the NSGA is run for 50 generations. Other parameters identical to that used in the previous test problem is used. The solution obtained from NSGA is as follows:

$$x_1 = 3.568, \quad x_2 = 2.939, \quad f_1 = 10.486, \quad f_2 = 8.961.$$

This solution is marked on both figures. This solution is feasible and lies on the constraint boundary. It does not violate the second goal, however it violates the first goal by an amount of $(16 - 10.486)$ or 5.514. Figure 11 shows that it violates the first goal ($f_1 \geq 16$) minimally (keeping minimum distance from the feasible search space).

6 An Engineering Design

This problem has been well studied in the context of single-objective optimization[19]. A beam needs to be welded on another beam and must carry a certain load F (Figure 12). In the context of single-objective optimal design, it is desired to find four design parameters (thickness of the beam, b, width of the beam t, length of weld ℓ, and weld thickness h) for which the cost of the beam is minimum. The overhang portion of the beam has a length of 14 inch and $F = 6,000$ lb force is applied at the end of the beam. It is intuitive that

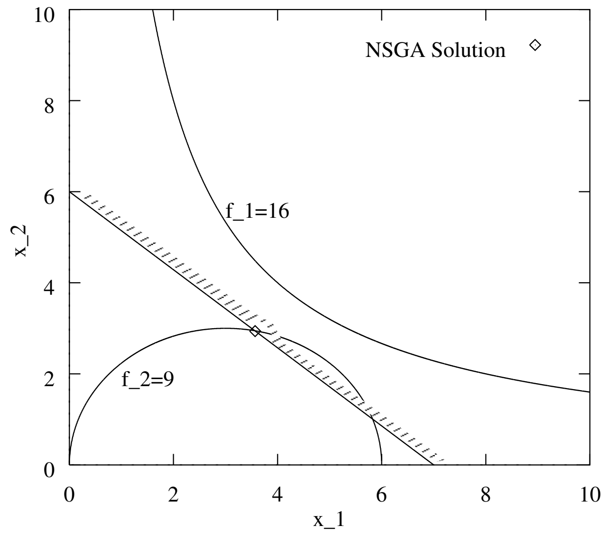

Figure 10: Criterion and decision spaces are shown for test problem P5.

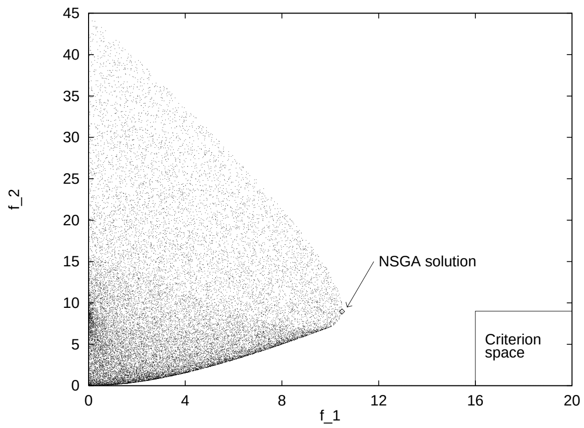

Figure 11: The NSGA solution is shown on a f_1-f_2 plot of test problem P5.

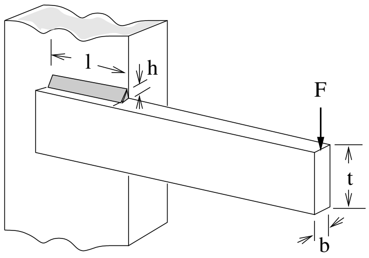

Figure 12: The welded beam design problem.

an optimal design for cost will make all four design variables to take small values. When the beam dimensions are small, it is likely that the deflection at the end of the beam is going to be large. Thus, the design solutions for minimum cost and minimum end deflection are conflicting to each other. In other words, a design that is near-optimal from cost consideration is not near-optimal from near small deflection consideration and vice versa. Such conflicting objectives appear in many instances of engineering design. In the following, we present a goal programming problem from the cost and deflection consideration:

$$\begin{array}{rl}
\text{goal } (f_1(\vec{x}) = 1.10471h^2\ell + 0.04811tb(14.0+\ell) \leq 5), & \\
\text{goal } (f_2(\vec{x}) = \frac{2.1952}{t^3b} \leq 0.001), & \\
\text{Subject to} & g_1(\vec{x}) \equiv 13,600 - \tau(\vec{x}) \geq 0, \\
& g_2(\vec{x}) \equiv 30,000 - \sigma(\vec{x}) \geq 0, \\
& g_3(\vec{x}) \equiv b - h \geq 0, \\
& g_4(\vec{x}) \equiv P_c(\vec{x}) - 6,000 \geq 0, \\
& 0.125 \leq h, b \leq 5.0, \\
& 0.1 \leq \ell, t \leq 10.0.
\end{array} \tag{18}$$

We would like to have a design for which the cost is smaller than 5 units and the deflection is smaller than 0.001 inch. If there exists any such solution, that solution is the desired solution. But, if such a solution does not exist, we are interested in finding a solution which will minimize the deviation in cost and deflection from 5 and 0.001, respectively.

There are four constraints. The first constraint makes sure that the shear stress developed at the support location of the beam is smaller than the allowable shear strength of the material (13,600 psi). The second constraint makes sure that normal stress developed at the support location of the beam is smaller than the allowable yield strength of the material (30,000 psi). The third constraint makes sure that thickness of the beam is not smaller than the weld thickness from a practical standpoint. The fourth constraint makes sure that the allowable buckling load (along t direction) of the beam is more than the applied load F. A violation of any of the above four constraints will make the design unacceptable. Thus, in terms of discussion in Ignizio[4], satisfaction of these constraints is the first priority. The stress and buckling terms are given as follows[19]:

$$\begin{aligned}
\tau(\vec{x}) &= \sqrt{\tau'^2 + \tau''^2 + \ell\tau'\tau''/\sqrt{0.25(\ell^2 + (h+t)^2)}},\\
\tau' &= \frac{6,000}{\sqrt{2}h\ell},\\
\tau'' &= \frac{6,000(14+0.5\ell)\sqrt{0.25(\ell^2+(h+t)^2)}}{2\,\{0.707h\ell(\ell^2/12+0.25(h+t)^2)\}},\\
\sigma(\vec{x}) &= \frac{504,000}{t^2 b},\\
P_c(\vec{x}) &= 64,746.022(1-0.0282346t)tb^3.
\end{aligned}$$

We handle these constraints using the bracket-operator penalty function[18]. Penalty parameters of 100 and 0.1 are used for the first and second criterion functions, respectively. Such dependence of penalty parameters on objective functions can be avoided by using a more efficient constraint handling technique suggested recently[26].

In order to investigate the search space, we plot about many random feasible solutions in f_1-f_2 space in Figure 13. The corresponding criterion space (cost $\leq$ 5 and deflection $\leq$ 0.001) is also shown in the

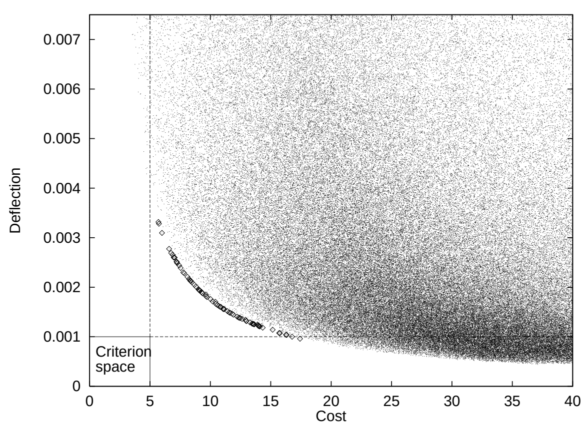

Figure 13: NSGA solutions (each marked with a 'diamond') are shown on objective function space.

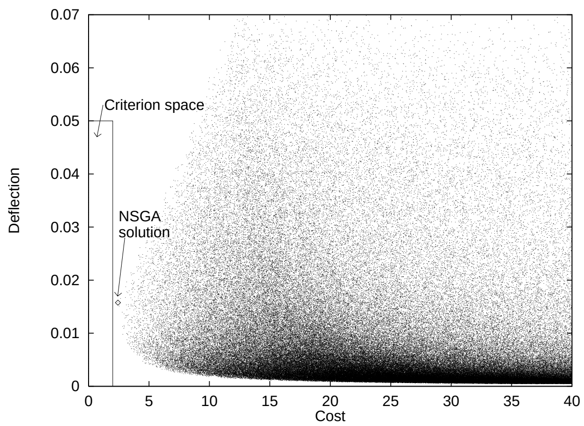

Figure 14: NSGA solution is shown on objective function space on the modified problem.

figure. The figure shows that there exists no feasible solution in the criterion space, meaning thereby that the solution to the above goal programming problem will have to violate at least of one of goals. A real-parameter NSGA with 100 population members and SBX operator with $\eta_c = 30$ and polynomial mutation operator with $\eta_m = 100$ is used. We also use σ_{share} of 0.281 (refer to equation 7 with $P = 4$ and $q = 10$). Figure 13 shows the solutions (each marked with a 'diamond') obtained after 500 generations. The existence of multiple solutions is accounted for the fact that no knowledge on weight factor of each goal is assumed here and each solution can be accounted for a different combination of weight factors of cost and deflection quantities. We present three such solutions in Table 2. If cost is more important than deflection the weight factor for cost will be more and a solution like the first solution will be chosen. On the other hand, if deflection is important, the weight factor for deflection will be more and a solution like the third solution will be chosen. Depending on the relative weights of cost and deflection, other solutions such as the one presented in the the second row in the table will be chosen. In the last two columns, relative weight factors are computed for each solution using equation 11. There are a total of 100 such solutions found by the NSGA in one run.

Table 2: Three solutions of welded beam goal programming problem.

Cost	Defl.	h	ℓ	t	b	w_1	w_2
5.70	0.0033	0.627	1.644	9.996	0.662	0.94	0.06
10.02	0.0018	0.778	1.274	9.999	1.247	0.43	0.57
16.76	0.0010	1.032	0.890	9.998	2.194	0.00	1.00

When all such solutions are available with a designer, usually a higher-level decision making (with further goals) procedure or compromise programming[25] can be used to choose one solution. However, the procedure adopted here show how many such solutions can be obtained simultaneously in one single run.

We construct another problem by changing the targets to $t_1 = 2.0$ and $t_2 = 0.05$. Since there exists no solution with cost smaller than 2 units, and the deflection of 0.05 inch is also large enough, the resulting solution is supposed to be the minimum-cost solution. Figure 14 shows that NSGA with identical parameter setting converges to one solution:

$$h = 0.222, \quad \ell = 7.024, \quad t = 8.295, \quad b = 0.244.$$

This solution has a cost of 2.431 units and deflection of 0.0157 inch. Figure 14 shows that this solution is very near to the minimum-cost solution.

7 Conclusions

Classical methods used for solving goal programming problems require users to provide a weight factor for each goal. The resulting solution, therefore, depends on the chosen set of weight factors. Moreover, the popular weighted goal programming problem has the difficulty of finding important solutions in problems having non-convex feasible decision space. In this paper, we reformulate the goal programming problem into a multi-objective optimization problem and suggest using a multi-objective GA to find desired solutions. Since multi-objective GAs can find multiple Pareto-optimal solutions in one single run, the proposed technique is capable of finding multiple solutions to the goal programming problem, each corresponding to a different set of weight factors. This makes the proposed approach free from using any weight factor and also allows to find solutions to goal-programming problems having non-convex decision space. The efficacy of the proposed technique has been shown by solving five test problems and one engineering design problem. The results are encouraging and suggest the use of the proposed approach to more complex and real-world engineering goal-programming problems.

Acknowledgments

The author acknowledges the support provided by Alexander von Humboldt Foundation, Germany during the course of this study.

References

[1] Charnes, A., Cooper, W., and Ferguson, R. (1955). Optimal estimation of executive compensation by linear programming. *Management Science*, **1**: 138–151.

[2] Clayton, E. R., Weber, W. E., and Taylor, B. W. (1982). A goal programming approach to the optimization of multiresponse simulation models. *IIE Transactions*, **14**(4): 282–287.

[3] Sayyouth, M. H. (1981). Goal programming: A new tool for optimization in petroleum reservior history matching. *Applied Mathematics Modelling*, **5**: 223–226.

[4] Ignizio, J. P. (1976). *Goal programming and extensions.* Lexington, MA: Lexington Books.

[5] Ignizio, J. P. (1978). A review of goal programming: A tool for multiobjective analysis. *Journal of Operations Research Society*, **29**(11): 1109–1119.

[6] Romero, C. (1991). *Handbook of critical issues in goal programming.* Oxford: Pergamon Press.

[7] Steuer, R. E. (1986). *Multiple criteria optimization: Theory, computation, and application*. New York: Wiley.

[8] Fonseca, C. M. and Fleming, P. J. (1993). Genetic algorithms for multi-objective optimization: Formulaton, discussion and generalization. *Proceedings of the Fifth International Conference on Genetic Algorithms*. 416–423.

[9] Srinivas, N. and Deb, K. (1994). Multi-Objective function optimization using non-dominated sorting genetic algorithms, *Evolutionary Computation*, **2**(3): 221–248.

[10] Cunha, A. G., Oliveira, P., and Covas, J. A. (1997). Use of genetic algorithms in multicriteria optimization to solve industrial problems. *Proceedings of the Seventh International Conference on Genetic Algorithms*. 682–688.

[11] Eheart, J. W., Cieniawski, S. E., and Ranjithan, S. (1993). Genetic-algorithm-based design of groundwater quality monitoring system. *WRC Research Report No. 218*. Urbana: Department of Civil Engineering, The University of Illinois at Urbana-Champaign.

[12] Parks, G. T. and Miller, I. (1998). Selective breeding in a multi-objective genetic algorithm. *Proceedings of the Parallel Problem Solving from Nature, V*, 250–259.

[13] Weile, D. S., Michielssen, E., and Goldberg, D. E. (1996). Genetic algorithm design of Pareto-optimal broad band microwave absorbers. *IEEE Transactions on Electromagnetic Compatibility*, **38**(4).

[14] Horn, J. and Nafploitis, N., and Goldberg, D. E. (1994). A niched Pareto genetic algorithm for multiobjective optimization. *Proceedings of the First IEEE Conference on Evolutionary Computation*. 82–87.

[15] Laumanns, M., Rudolph, G., and Schwefel, H.-P. (1998). A spatial predator-prey approach to multiobjective optimization: A preliminary study. *Proceedings of the Parallel Problem Solving from Nature, V*. 241–249.

[16] Zitzler, E. and Thiele, L. (1998a). Multiobjective optimization using evolutionary algorithms—A comparative case study. *Parallel Problem Solving from Nature, V*, 292–301.

[17] Lee, S. M. (1972). *Goal programming for decision analysis.* Philadelphia: Auerbach publishers.

[18] Deb, K. (1995). *Optimization for engineering design: Algorithms and examples.* New Delhi: Prentice-Hall.

[19] Reklaitis, G. V., Ravindran, A. and Ragsdell, K. M. (1983). *Engineering optimization methods and applications*. New York: Wiley.

[20] Zeleney, M. (1982). *Multiple criteria decision making*. New York: McGraw-Hill.

[21] Deb, K. and Goldberg, D. E. (1989). An investigation of niche and species formation in genetic function optimization. *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 42–50).

[22] Goldberg, D. E. (1989). *Genetic algorithms for search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

[23] Deb, K. and Agrawal, R. B. (1995). Simulated binary crossover for continuous search space. *Complex Systems*, **9**: 115–148.

[24] Deb, K. and Goyal, M. (1997). A robust optimization procedure for mechanical component design based on genetic adaptive search. *ASME Journal of Mechanical Design*.

[25] Zeleney, M. (1973). Compromise programming. In J. L. Cochrane and M. Zeleney (Eds.) *Multiple criteria decision making*, (pp. 262–301).

[26] Deb, K. (in press). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*.