A Generic Framework for Constrained Optimization Using Genetic Algorithms

Sangameswar Venkatraman and Gary G. Yen, *Senior Member, IEEE*

***Abstract*—In this paper, we propose a generic, two-phase framework for solving constrained optimization problems using genetic algorithms. In the first phase of the algorithm, the objective function is completely disregarded and the constrained optimization problem is treated as a constraint satisfaction problem. The genetic search is directed toward minimizing the constraint violation of the solutions and eventually finding a feasible solution. A linear rank-based approach is used to assign fitness values to the individuals. The solution with the least constraint violation is archived as the elite solution in the population. In the second phase, the simultaneous optimization of the objective function and the satisfaction of the constraints are treated as a biobjective optimization problem. We elaborate on how the constrained optimization problem requires a balance of exploration and exploitation under different problem scenarios and come to the conclusion that a nondominated ranking between the individuals will help the algorithm explore further, while the elitist scheme will facilitate in exploitation. We analyze the proposed algorithm under different problem scenarios using Test Case Generator-2 and demonstrate the proposed algorithm's capability to perform well independent of various problem characteristics. In addition, the proposed algorithm performs competitively with the state-of-the-art constraint optimization algorithms on 11 test cases which were widely studied benchmark functions in literature.**

***Index Terms*—Constrained optimization, constraint handling, genetic algorithm (GA), hyperheuristic.**

I. Introduction

MOST REAL-WORLD optimization problems involve constraints. Consider an optimization problem such as maximizing the profits of a particular production line. The objective function to be maximized could be a function of various manipulating variables, including but not limited to the material consumption, the labor cost, the operating hours of the machines, and many additional factors. If the raw materials, manpower, and machines can be made available without limitation then there is no limit to the profit that can be achieved. However, in face of real-world complications, they are most likely limited in the form of constraints imposed upon the optimization function. What constitute the difficulties of the constrained optimization problem are various limits on the decision variables, the constraints involved, the interference among constraints, and the interrelationship between the constraints and the objective function. Taking a numerical example, suppose we want to maximize a function $f(X) = x_1 + x_2$, where the two variables are defined by $0 \leq x_1, x_2 \leq 1$. Under the presence of no additional constraint, an optimum value of $f(X) = 2$ can be reached when $x_1 = 1$ and $x_2 = 1$. Assume that there is an equality constraint imposed on these variables described by $g(X) \equiv x_1 - x_2 = 0.5$. Considering a resolution of up to two decimal places in the discrete search space, there are only 50 feasible solutions among 10 000 possible candidates. This implies that feasible space is only 0.5% of the actual parameter space. The best objective function value that can be reached is $f(X) = 1.5$ (for $x_1 = 1$ and $x_2 = 0.5$). The problem complexity can be greatly increased by the number of constraints or the types of constraints. The general constrained continuous-parameter optimization problem as succinctly defined in [29] is to search for X so as to

$$\text{Optimize } f(X), \quad X = (x_1, x_2, \cdots, x_n) \in \Re^n \tag{1}$$

where $X \in F \subseteq S$. The objective function f is defined on the search space $S \subseteq \Re^n$, and the set $F \subseteq S$ defines the feasible region. Usually, the search space is an n-dimensional hyper box in $\Re^n$. The domains of the variables are defined by their lower and upper bounds as

$$l(i) \leq x_i \leq u(i), \quad 1 \leq i \leq n \tag{2}$$

whereas the feasible region F is restricted by a set of m additional constraints $(m \geq 0)$

$$\text{with } q \text{ inequality constraints } g_j(X) \leq 0, \quad (j = 1, \ldots, q) \tag{3}$$

and

$$m-q \text{ equality constraints } h_j(X) = 0, \quad (j = q+1, \ldots, m). \tag{4}$$

The inequality constraints that take the value of 0, i.e. $g_j(X) = 0$ at the global optimum to the problem are called the *active constraints*. In the following discussion and in the remainder of this paper, without loss of generality we shall consider the minimization of the objective function unless specified otherwise. In addressing the constrained optimization problem in the real-world scenario, we can arguably say that obtaining a feasible solution (one that is usable under the problem formulation) takes precedence over optimizing the objective function (which minimizes the cost involved). There are also problems with higher complexity in which finding a single feasible solution itself can be a monumental task. These problems are treated as constraint satisfaction problems and various evolutionary algorithms have been proposed to solve them effectively, e.g., [10]. The main challenge in constrained

Manuscript received May 19, 2004; revised November 17, 2004. This work was supported in part by the Center for Aircraft Systems/Support Infrastructure (CASI) and in part by the Oklahoma City Air Logistics Center.

The authors are with the Intelligent Systems and Control Laboratory, School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK 74078 USA (e-mail: gyen@ceat.okstate.edu).

Digital Object Identifier 10.1109/TEVC.2005.846817

optimization is simultaneously handling the constraints as well as optimization of the objective function. The constraint handling methods have primarily focused on various designs of fitness formulation for each individual in the population depending on its objective function and constraint satisfaction. Over the past decade, various constraint-handling techniques using genetic algorithms (GAs) [7], [10], [12] and benchmark test functions [24] for constrained optimization problems have been proposed.

We summarize next our motivations to propose a new constraint handling scheme to complement the existing methods in literature.

1) Reliability: From a practical point of view, it is essential that the GA used for constrained optimization produces feasible optimal solutions for every run. While this may be too much to ask, we would certainly hope that at least the feasibility criteria can be met for every run and that adequate optimization can be achieved.
2) A generic framework: There are various types of constrained optimization problems that we may encounter and it would be impractical if the GA used has to be tuned to fit for a specific problem or if it uses special operators that cannot be implemented in all problem domains. While a generic framework may not result into the most efficient design for each problem setting, it is the most advisable at the algorithm design stage, while necessary modifications can be made to tailor for particular problems (such as exploiting carefully crafted genetic operators).

The rest of the paper is organized as follows. In Section II, a thorough review of the essential features of some of the best known constraint handling schemes pertinent to this research is presented. In Section III, we introduce the proposed constraint handling scheme and elaborate it in detail. In Section IV, we analyze the distinctions between the proposed constraint-handling scheme and another scheme that favors domination of feasible solutions in tackling problems with different characteristics. Using the Test Case Generator-2 (TCG-2) proposed in [35], we demonstrate the effectiveness and efficiency of the proposed constraint handling method in Section V. In Section VI, we evaluate the performance of the proposed constraint-handling scheme on the 11 test cases proposed in [24] and provide a fair comparison with the best results from the state-of-the-art in literature. Finally, we conclude with some observations about the proposed algorithm.

II. Literature Survey

In this section, we will review some of the relevant methods proposed for constraint handling using GAs. We have broadly categorized these methods into: 1) methods based on penalty functions; 2) methods based on preference of feasible solutions over infeasible ones; and 3) methods based on multiobjective optimization. Techniques involving special operators [30], decoders [24], repair mechanisms [27], and hybrid approaches [22] are considered irrelevant to the algorithm proposed herein. We intend to present the basic ideas underlying the design of each of the above constraint-handling methods and provide the rationale to justify our proposed design framework.

1) Methods Based on Penalty Functions: Penalty functions were popularly used in the conventional methods for constrained optimization [15] and were amongst the first methods used to handle constraints with evolutionary algorithms. In these methods, the individuals are penalized based on their constraint violations. The penalty imposed on infeasible individuals can range from completely rejecting the individual to decreasing its fitness based on the degree of violation. There are different types of penalty functions based on this principle and some of them are discussed next.

In the death penalty method, the infeasible solutions are not considered for selection for the next generation and this is the greatest penalty that can be imposed on an infeasible solution. Among all penalty methods, the death penalty is the simplest to implement. This approach has the drawback of not exploiting any information from the infeasible individuals to guide the search [7]. Also, when the initial population consists of no feasible solution, the whole population has to be rejected and a new one is randomly generated. This technique may work well in problems where the feasible space is convex and covers a large part of the search space but is not generally used otherwise.

In the static penalty method, the penalty is a weighted sum of the constraint violations. The objective function is modified as

$$\mathrm{obj}(X) = f(X) + \sum_{j=1}^{m} r_j c_j(X) \tag{5}$$

$f(X)$ actual objective function value;
r_j penalty coefficient for constraint j;
$c_j(X)$ degree of violation of constraint j corresponding to the individual X;
$\mathrm{obj}(X)$ modified objective function value after adding penalty.

The success of the static penalty method depends on the proper penalty coefficients chosen for constraints. This has to be determined carefully based on the difficulties of these constraints.

In [21], a dynamic penalty method was proposed where the penalty assigned to each individual depends on the generation number and a scaling constant C in addition to its constraint violation. The authors of [21] claim that this as an important distinction which applies more selective pressure on nearly feasible solution, thus making them feasible. However, the difficulty involved in tuning many parameters for dynamic penalty method has significantly limited its applicability.

While the penalty function methods discussed so far are easy to implement, they require some degree of parameter tuning to tailor for each problem. From [33], we summarize some of the guidelines for penalty function methods: 1) penalties which are functions of the distance from feasibility perform better than those which are merely functions of the number of violated constraints; 2) for a problem having few constraints and few feasible solutions, penalties which are solely functions of the number of violated constraints are not likely to find solutions; and 3) the more accurate the penalty is estimated, the better quality of solution can be found.

A method with a self-adaptive penalty function to alleviate the difficulty of choosing penalty coefficients was proposed in [14]. The authors proposed a two-stage penalty function that requires no explicit definition of any parameters. The method was formulated to ensure that slightly infeasible solutions with a low objective function value remain fit. The first stage ensures that the worst of the infeasible solutions has a penalized objective function that is higher than or equal to that of the best solution in the population (all other solutions are penalized by a lesser amount depending on their feasibility). The second penalty increases the penalized objective value of the worst of the infeasible solutions to twice the objective value of the best solution. The method, however, required the definition of a scaling factor.

Because penalty functions combine the objective function value and the constraint violation value to decide the fitness of each individual, there is a domination relationship between the constraint violation and the objective function in deciding the fitness of the individual. In [34], the authors characterize the problem of choosing the appropriate penalty coefficient (r_j) for each constraint and describe how it affects the domination between the constraint violation and the objective function in deciding the rank of each individual. To overcome the burden of choosing an optimal r_j, the authors propose a probability factor P_f which denotes the probability of the objective function used to allocate rank to the individual. The ranking method incorporated assures that feasible solutions are ranked based only on their objective function, while the probability factor P_f determines whether objective function or constraint violation should be used to rank infeasible individuals. A P_f value of 0.45 was found to produce very good results in [34]. This implies that infeasible solutions would be ranked less often based on their objective function value (45%) and more often based on their constraint violation value (55%). While this method produced the best results for all of the problems evaluated, there was one significant drawback with the results obtained. For a particular test problem, the method could only produce feasible solutions 6 out of 30 runs. This can be attributed to the selection scheme in which constraint violation does not dominate the objective function even when ranking infeasible solutions.

2) Methods Based on Preference of Feasible Solutions Over Infeasible Solutions: In [32], the authors suggested a penalty function method in which feasible solutions would always have higher fitness than infeasible ones. A rank-based selection scheme was used and the rank was assigned based on the objective function values mapped into $(-\infty, 1)$ for feasible solutions and the constraint violation mapped into $(1,\infty)$ for infeasible solutions. Hence, in this technique, all feasible solutions dominate the infeasible ones. Infeasible solutions will be compared based on their constraint violation, while feasible solutions will be compared based on their objective function value only. This method presents some interesting properties: 1) as long as no feasible solution is found, the objective function will produce no effect on the rank of the individual; 2) once there is a combination of feasible and infeasible solutions in the population, then feasible solutions will be ranked ahead of all infeasible solutions; and 3) feasible solutions will be ranked based on their objective function values. The major drawback that we could experience in this method is a lack of diversity either explicitly defined or as part of the selection scheme. This deficiency will occur in problems with disconnected feasible components in which cases the GA may be stuck within one of the feasible components and never get to explore.

The same idea as described above formed the basis of [12], where selection was based upon the following underlying principles: 1) a feasible solution wins over any infeasible solutions; 2) two feasible solutions are compared only based on their objective function values; 3) two infeasible solutions are compared based on the amount of their constraint violations; and 4) two feasible solutions i and j are compared only if they are within a critical distance $\overline{d}$, otherwise, another solution j is checked n_f times before i is chosen as the winner. The authors of [12] also argued that real coded representation was better suited for constrained optimization problems as it affords a greater chance of maintaining feasibility. In addition, the authors used a niching scheme to maintain diversity among feasible solutions and binary tournament selection to make pairwise comparisons. The penalty approach was different in the sense that the coefficient r_j was unity for all constraints and all the constraints were normalized to allot equal importance to each constraint. This method performed very well on a variety of benchmark test problems and niching operator was incorporated to overcome the problem of stagnation discussed above. However, this method requires heuristically chosen parameters on critical distance $\overline{d}$ and n_f.

3) Methods Based on Multiobjective Optimization Techniques: Constrained optimization by multiobjective genetic algorithm (COMOGA) was proposed in [36], where the solutions are first ranked based on nondomination of their constraint violations, and then also ranked based on their objective function. A P_{cost} factor selects solutions based on objective function, while the others are selected based on constraint violation. The P_{cost} is adjusted depending on the target proportion of feasible solutions in the population. The obvious drawback of this technique is the high computational complexity, especially as the number of constraints increases. Also, there is not enough evidence to suggest that treating each constraint independently and ranking them based on Pareto domination is an efficient design.

In [6], the author proposed a subpopulation-based approach similar to VEGA by using $m + 1$ subpopulations, where m denotes the number of constraints and the first subpopulation is devoted to optimizing the objective function. The method differs from [36] in that nondominated ranking is never employed but the fitness function for each problem is changed so that initially the fitness function for each subpopulation (except the first one which is based on the objective function) depends on the violation of its constraint. If the solution evaluated does not violate the constraint corresponding to the subpopulation but is infeasible, then the subpopulation will minimize the total number of violations. Finally, once the solution becomes feasible, it will be merged with the first subpopulation and look to minimize the objective function. While the results produced were satisfactory, the choice of the size of each subpopulation remained an open question.

From our analyses of the algorithms previously proposed to solve the constrained optimization problem, we notice three common features: 1) lack of elitism; 2) choices of parameters

that require *a priori* knowledge about the problem characteristics; and 3) lack of an assurance of producing feasible solutions. Elitism can be very effective in maintaining the best feasible solution in the population and can guide the genetic search to concentrate around this solution, thus providing an exhaustive search. At the same time, elitism can lead to poor diversity in the population. The algorithm proposed herein blends the elitist approach with a nondominated ranking approach, thus providing a delicate balance between exploration and exploitation. We believe that a possible rationale that many methods fail to produce feasible solutions for every run is the simultaneous handling both the objective function and the constraints in all stages of the algorithm. This prevents the GA from moving toward the feasible regions without being distracted by the objective function landscape. Our proposed method provides a greater assurance of producing feasible solutions because the search is directed only based on the constraint violation value until feasible solutions are found.

While it is commonly accepted that no algorithm will be effective in solving all types of problems as documented in the no free lunch theorem [40], we have proposed a generic framework that requires absolutely no definition of any problem dependent parameter. This is in keeping up with the development of algorithms with an increasing level of generality by the usage of hyperheuristics. Hyperheuristics are heuristics that choose between the lower-level heuristics. Hence, with respect to GAs, hyperheuristics define an approach where the higher level is a GA that decides which heuristic to call next. That is, a GA is coded such that it represents a sequence of heuristic calls, rather than a representation of the problem itself. The lower level of the hyperheuristic is the set of heuristics which operate directly on the solution. The interested reader is referred to [2] and [3] for more reading. Our proposed algorithm is an effort to define a hyperheuristic to solve constrained optimization problems. Throughout this research, we have not made an effort to experiment with lower level heuristics. In the following, Section III describes the proposed constraint handling scheme and how it can satisfy the design requirements of reliability and problem independence.

III. Proposed Constraint Handling Scheme

GAs being a stochastic search technique can offer no guarantee of producing feasible solutions. To address this concern, we have formulated the GA in such a way that finding feasible solutions is the prioritized objective. Once a feasible solution is found, then the best one is kept in the population using the elitist scheme, thus assuring that the found feasible solution is never lost. However, preferring feasible solutions over infeasible ones could cause the GA to be stuck in one particular feasible component, where there are disconnected feasible components and the GA may never get to explore the other feasible components containing the global optimum. So exploring the search space guided by both the constraint satisfaction and the objective function optimization will be the secondary objective. The proposed constraint handling scheme consists of two phases and the algorithm switches smoothly from the first phase to the second based on a simple conditional statement.

1) *Phase one (constraint satisfaction algorithm)*: In the first phase of the algorithm, the objective function is completely disregarded and the entire search effort is directed toward finding a single feasible solution. Each individual of the population is ranked based on its constraint violation only and fitness is assigned to each individual based on its rank. The elitist strategy is used and the solution with the least constraint violation is copied to the next generation. This phase takes care of the feasibility criteria and provides a usable solution (one that satisfies all constraints). We find this technique to be especially suitable for highly constrained problems wherein finding a feasible solution may be extremely difficult. In such problems, it would be worthwhile and efficient to explore the search space based on the constraints alone without taking the objective function into consideration.
2) *Phase two (constrained optimization algorithm)*: The algorithm switches to this phase once at least one feasible solution has been identified. This phase is treated as a biobjective optimization problem, where the constraint violations and the objective functions have to be minimized simultaneously in a modified objective space that we call the "objective function—constraint violation space," or $f - v$ space for short. We have used a nondominated sorting like in [13] to rank the individuals. We save the feasible individual with the best objective function in the population as the elitist solution. We also use a niching scheme in the $f - v$ space so that sufficient diversity is maintained and the GA will continue to explore. We believe that this multiobjective evolutionary algorithm (MOEA)-based approach will search to minimize both the objective function and constraint violation simultaneously and guide the algorithm in exploring the region between the constrained and unconstrained optima and the feasible and infeasible parts of the search space. The details of implementation of the algorithm are given below.

A. Scalar Constraint Violation

From the problem formulation, we have m constraints and the constraint violation for an individual is a vector of m dimensions. Using a tolerance (δ) of 0.001 for equality constraints, the constraint violation of individual X on the jth constraint is calculated by

$$c_j(X) = \begin{cases} \max(0, g_j(X)), & j = 1, \cdots, q \\ \max(0, |h_j(X)| - \delta), & j = q+1, \cdots, m \end{cases}. \quad (6)$$

Each constraint violation is then normalized by dividing it by the largest violation of that constraint in the population. We use normalized constraint violations to treat each constraint equally. First, we find the maximum violation of each constraint in the population by using (7)

$$c\max(j) = \max_X c_j(X). \quad (7)$$

These maximum constraint violation values are used to normalize each constraint violation. The normalized constraint violations are added together to produce a scalar constraint

violation $v(\mathrm{X})$ for that individual which takes a value between 0 and 1

$$v(X) = \frac{\sum_{j=1}^{m} \frac{c_j(X)}{c\max(j)}}{m} \tag{8}$$

where $|\cdot|$ denotes the absolute operator.

B. Rank-Based Fitness Allocation

In both phases of the proposed algorithm, we allocate a fitness to each individual based on its rank in the population. In the first phase, all the individuals are ranked based on their scalar constraint violation. The rank-based fitness function is implemented from [4]. In the second phase, the individuals are sorted into different fronts based on nondomination and rank is assigned to each individual based on the front it belongs to.

C. Crowding-Distance Assignment

It is desirable to have a diverse set of solutions in the $f - v$ space to maintain the explorative power of the algorithm and, hence, a niching scheme based on the distance of the nearest neighbors to each solution is applied. To get an estimate of the density of solutions surrounding a particular individual in the second phase, we calculate the normalized average distance of two points on either side of this point along each one of the dimensions. This quantity $d(X)$ takes a value between 0 (the individual has multiple copies in the population) to 1 (the individual is not crowded). The fitness of the individual based on its rank and crowding-distance is given by

$$\mathrm{fitness}(X) = r(X) + d(X). \tag{9}$$

Note here that the elitist individual is chosen irrespective of its *fitness* but based only on the conditions for each of the two phases. The pseudocode of the algorithm is given in Fig. 1.

In the following section, we discuss the algorithm design and how we come up with it based on the difficulties associated with different problem scenarios.

IV. Constrained Optimization—Algorithm Design

As discussed before, one of the major challenges for constrained optimization is to search for optimal solutions that are feasible with respect to the constraints. One of the approaches for effectively solving the constrained optimization problem is to treat the constraints as "objectives with goals" and define preference among individuals as described in [17]. However, this can lead to an extremely high-dimensional objective space as the number of constraints grows. The computational complexity will become unmanageable. Hence, we have used a single parameter, the *scalar constraint violation* (SCV), representing normalized net violation of constraints by an individual. To analyze the proposed algorithm further, let us consider each of the two phases individually. Let us define the usage of the following terms:

F	feasible region, i.e. the domain of the search space S that is feasible;
C_i	ith disconnected feasible component in the search space $S, i = 1, \cdots, k$;
$C_i \cap C_j$	$= \phi, i \neq j, i, j = 1, \cdots, k$
$C_1 \cup C_2 \cup \cdots C_k$	$= F$

find ϕ – number of feasible solutions in the population
  if ($\phi == 0$)
    // PHASE I
    Objective ⇒ *Minimize* $v(X)$
    elite solution ⇒ *solution with least* $v(X)$
    $r(X)$ ⇒ *rank based fitness of individual based on violation* $v(X)$
    fitness$(X) = r(X)$
  else
    // PHASE II
    $f(X)$ ⇒ *given objective function*
    Objective ⇒ *Minimize* ($f(X)$,$v(X)$)
    elite solution ⇒ *feasible solution with least* $f(X)$
    $r(X)$ ⇒ *non-dominated rank based fitness of individual*
    $d(X)$ ⇒ *Crowding – distance assignment of individual* (0-1)
    fitness$(X) = r(X) + d(X)$
  end
  Apply genetic operators on current population
  generation = generation + 1
end

Fig. 1. Pseudocode of the proposed constraint handling algorithm.

A. Phase One—Constrained Satisfaction Problem

Goal: To find a feasible solution from a random initialization.

If there are m constraints and k feasible components, then a GA with selection based only on constraint violation will find a feasible solution with probability one as $t \to \infty$. This is true because in this case the scalar constraint violation is only a measure of distance from the feasible region. As selection favors minimizing this distance, a feasible solution will be eventually reached. Since there are k feasible components, the probability of the first feasible solution being found in any one of the feasible components is approximately $1/k$.

Next, we begin our analysis of the second phase of the algorithm where the actual optimization takes place. During the design of our fitness scheme, we could have chosen either one of the following two schemes: the *preference scheme* or the *nondominated scheme*. The *preference scheme* based on [31] is defined by the following.

1) Any feasible solution is better than any infeasible solution.
2) Among two feasible solutions i and j, assign greater probability of selection to the solution with the better objective function.

We shall compare this with the *nondominated scheme* in which:

1) Solutions are ranked based on the nondomination of their constraint violations and objective function values.

In analyzing these two selection schemes, we try to draw meaningful conclusions about the design of the selection scheme under different problem scenarios.

B. Phase Two—Constrained Optimization Problem

Goal: To search for the feasible global optimum after a single feasible solution is found.

We define the efficiency of a search technique by its speed (with respect to the number of function evaluations) at which it can get to the global optimum as opposed to an exhaustive brute-force search.

A major issue in solving the constrained optimization problem is the balance between the exploration and exploitation. Let us consider the $f - v$ *space*. In our algorithm, we maintain the feasible solution with the best objective function unchanged in our population and this can be regarded as an artificial way of creating a genetic drift phenomenon which helps in exploitation. At the same time, we apply a niching scheme in the $f - v$ *space* looking for a well extended and uniform Pareto front, thus helping the algorithm explore even when it is converging. The following cases illustrate why and when this property of the algorithm is essential.

Case 1: There is only one feasible component $(k = 1)$—a need for exploitation

In this case, our initial feasible solution will belong to this feasible component and the global optimum is also located within this component. Selection based on the *preference scheme* will be more efficient in converging to the global optimum than the *nondominated scheme* as: 1) there is no need to explore and 2) the infeasible solutions (which carry no useful genetic information in this case) are not encouraged in the population and this technique can lead to the global optimum in a less number of evaluations.

Case 2: If there are $k(> 1)$ *disconnected feasible components*—a need for exploration.

In this case, the *preference scheme* may not be efficient in converging to the global optimum. This is because the chances of the feasible initial solution being located in the feasible component with the global optimum is $1/k$ and becomes less as the number of disconnected components increase so there may be a need for the GA to search for solutions in the other components. This presents a need for exploration to find feasible solutions. We analyze the two methods by taking two solutions i and j from the population and evaluate the selection scheme that will increase the probability of converging to the global optimum. We also assume in our discussion that the feasible solution with the best objective function is saved as the elitist solution in the population.

1) If solution i is feasible and j is infeasible, we could:
   a) assign a greater probability of selection to i irrespective of the objective function values of i and j;
   b) check if j has a better objective function value than i and consider both i and j nondominated if it does, otherwise, assign a greater probability of selection to i.

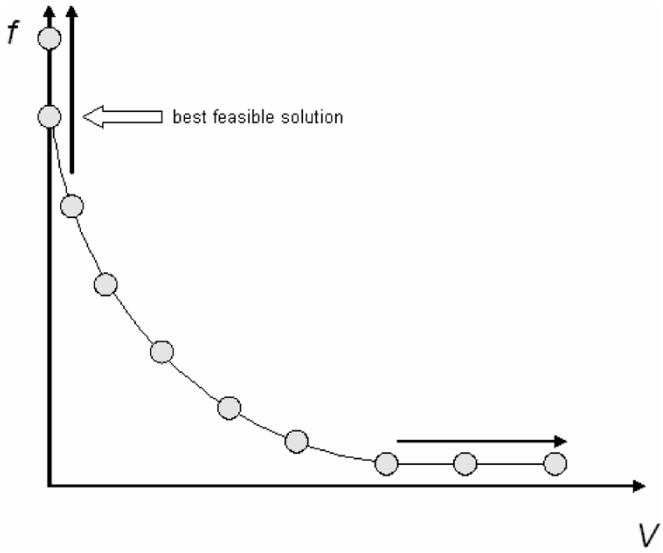

Fig. 2. Schematic of the nondominated ranking used in the GA.

In designing our algorithm, we chose option b). This is because our elitist scheme already saves the best solution in the population. This elitist solution is obviously feasible and has an objective function value that is just as good or better then i. So irrespective of whether i or j is chosen, the elitist scheme assures that a part of the genetic search proceeds along the direction of the feasible solution with the best objective function. Hence, by giving j an equal probability of selection, we are also favoring genetic search in the infeasible regions that may have good objective function values.

2) Among two feasible solutions i and j, consider i and j nondominated irrespective of the objective function values. This helps the algorithm explore more as the best feasible solution is already stored as the elitist solution. Hence, by giving both i and j an equal probability of selection we are giving the algorithm a better chance to explore.

Fig. 2 shows a nondominated set of solutions in the $f - v$ space and all these individuals are ranked one. The niching scheme assigns different fitness to these solutions based on how crowded they are. So there is a greater selective pressure on solutions at the two corners. The niching scheme tries to extend the Pareto front along the directions of the two solid (black) arrows. Since the elitist solution is saved unchanged in the population, there is a greater probability of solutions around it. Hence, we have indicated this solution in the figure using the hollowed (white) arrow.

In the next section, we introduce the TCG-2 [35] and perform actual experiments using the two selection schemes under the problem scenarios discussed above in Cases 1 and 2.

V. Selection Scheme Comparison Using TCG-2

In this section, we have used the TCG-2 to simulate different problem scenarios and evaluate the performance of the two selection schemes proposed. The TCG-2 is an enhanced version

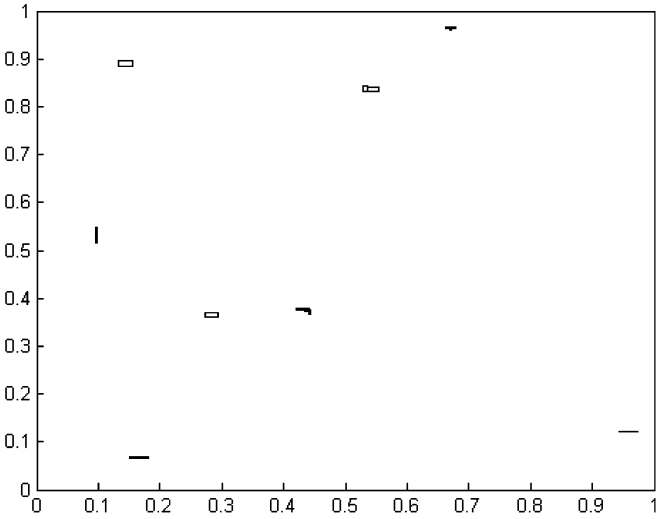

Fig. 3. Feasible components for $n = 2$, $\rho = 0.001$, $c = 0.2$, $m = 8$, and $d = 0.1$.

of the TCG proposed in [31]. The nine different tunable features of the TCG-2 are as follows:

- n dimensionality of the problem;
- m number of feasible components;
- ρ feasibility ratio of the search space;
- c complexity of the feasible search space;
- a number of active constraints;
- p number of peaks of the objective function;
- σ width of the peaks;
- α decay of height of the peaks;
- d distance between the different feasible components.

The search space is composed of an n-dimensional hypercube with each dimension ranging in the closed interval of [0,1]. The feasible regions of the search space are determined by m, ρ, c, and d. The general idea behind the TCG-2 is to randomly create m nonoverlapping boxes (or rectangular areas) in the search space. The total occupancy of the m feasible components put together is $\rho \times |S| \times (1 - c)$. Considering a two dimension search space, if the complexity c is zero, then there are m feasible components and each one of them is a perfect rectangle. New boxes are attached to the existing ones maintaining a minimum distance d between the feasible components for the remaining $\rho \times |S| \times c$ part of the search space. Fig. 3 shows the feasible components in a two-dimensional search space for $n = 2$, $\rho = 0.001$, $c = 0.2$, $m = 8$, and $d = 0.1$.

Based on the created feasible components, the constraint violation function is defined. The constraint violation value is zero inside the feasible components, while outside the feasible components the constraint violation value is the distance to the closest center of all the feasible components. The constraint violation is defined by

$$cv(X) = \begin{cases} 0, & \text{if } X \text{ is inside a feasible component} \\ |cc_{\text{feasible}} - X|, & \text{otherwise} \end{cases} \quad (10)$$

where cc_{feasible} is the closest center to nay feasible component.

The objective function is defined using a set of p randomly placed Gaussians $g_k(X)$, where h_k is the height of the peak k and $\overline{c}_k$ is the center of the peak k. In order to evaluate the objective function $f(X)$, the closest center $\overline{c}_i$ of the solution vector is found, and then the Gaussian function $g_k(X)$ is evaluated.

All centers $\overline{c}_k$ are placed randomly in the search space with the exception of the global optimum that is placed such that there are exactly a active constraints at the global optimum. All peaks heights are evenly distributed between $[\alpha, 1]$ such that the global optimum has the highest peak $h_k = 1$, while the lowest peak has $h_k = \alpha$. The global optimum is placed either inside the feasible regions (if $a = 0$) or at the borders (if $a > 0$). Hence, the global optimum always satisfies the constraints and has a value of 1. The test scenario chosen is specified by a function TCG-2 (n, m, ρ, c, a, p, σ, α, and d). We have implemented two tests to verify the results from our previous discussion regarding selection schemes for different types of problems. In each of these tests, we have defined ten levels of difficulties and the performance of each algorithm is plotted based on how it treats problems of increasing difficulty.

To perform this test, we increase our problem complexity from a problem with one feasible component and one optimum to a problem with ten disconnected feasible components and ten peaks. All the other characteristics of the problem such as the feasibility ratio and number of active constraints are kept the same. In short, we basically have ten test scenarios defined by TCG-2 (2, m, 0.005, 0.2, 1, p, 0.2, 0.5, 0.1), where m and p are varied from 1 to 10. We allow the algorithm to run a maximum of 5000 generations. In addition, since we know that the global optimum is 1, we stopped the algorithm if the best feasible objective function value crosses 0.999 borderline. When implementing both the *preference scheme* and the *nondominated* scheme, we use niching in the second phase of the both algorithms to maintain diversity. The same elitist scheme is also employed in both designs. Fig. 4(a) and (b) compares the results from the *preference scheme* method and the *nondominated scheme*.

Starting with the same number of generations required for m and p value of 1, the nondominated scheme shows a much better performance when the number of disconnected components and the number of peaks increase. Even though the increase in the number of generations required to solve the problem does not increase linearly with m and p, we can clearly see that under all scenarios the nondominated scheme performs better. Fig. 5(a) and (b) shows the mean objective function values obtained under various problem scenarios.

Again, we notice that better performance is obtained by the *nondominated scheme* for all choices of m and p greater than 1. By acknowledging that the *nondominated scheme* achieved these results with a less number of generations, as shown in Fig. 4(a) and (b), we can argue with confidence that the nondominated scheme performs more efficiently in the current design scheme. In the next section, we extend the tests to the 11 test problems from [29] used frequently in literature and provide a fair comparison with some state-of-the-art approaches.

VI. Test Results

We apply the proposed constraint handling scheme to 11 test cases from [24], as shown in Table I, using real-coded

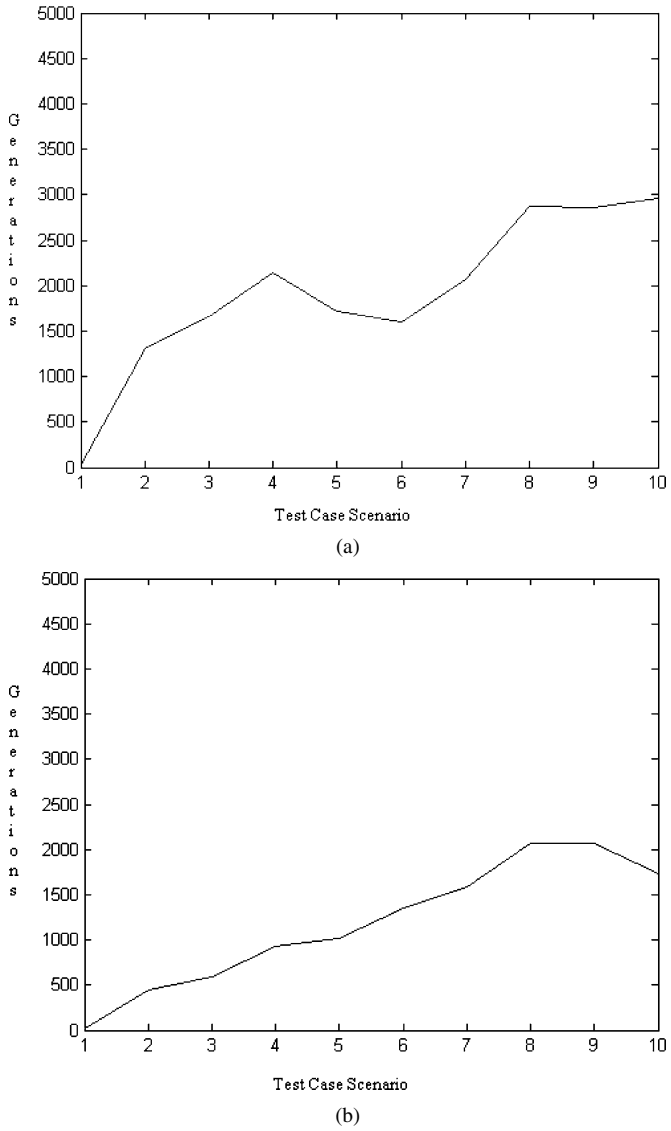

Fig. 4 (a) Number of generations used in preference scheme versus the test scenario number. (b) Number of generations used in nondominated scheme versus the test scenario number.

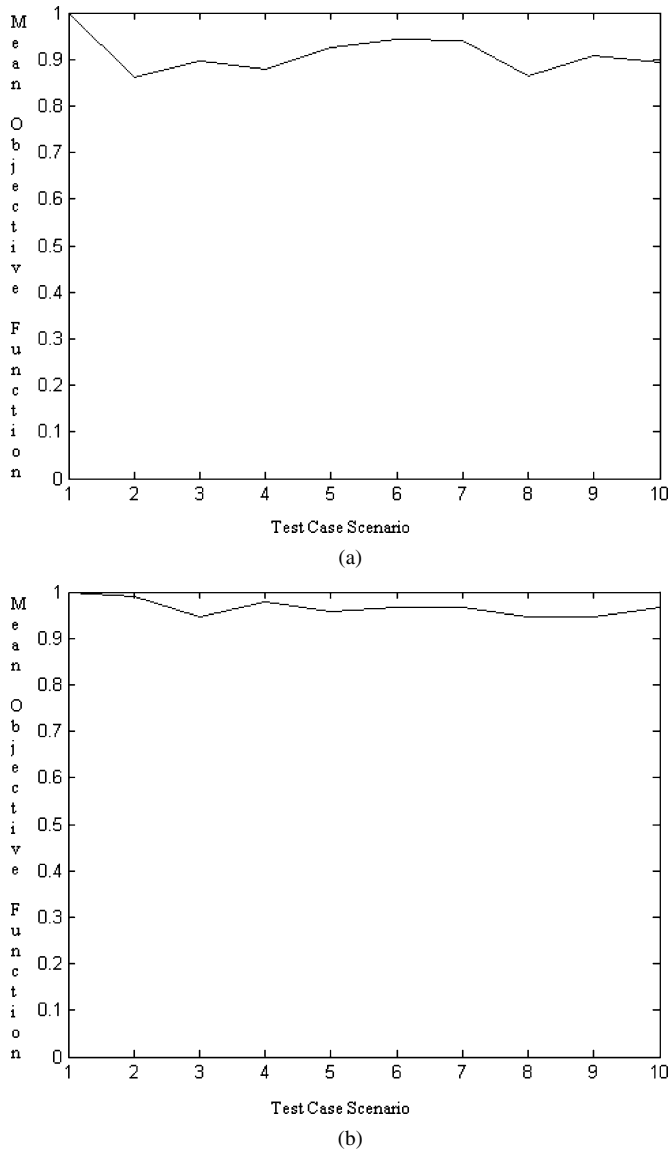

Fig. 5 (a) Mean objective function with a maximum of 5000 generations using preference scheme versus the test scenario number. (b) Mean objective function with a maximum of 5000 generations using nondominated scheme versus the test scenario number.

individuals with a probability of mutation $1/n$, where n is the number of decision variables involved. The crossover is implemented by using the binary representation of the chromosomes [4] with a probability of 0.9. For all of the 11 problems, we use a population size of only ten individuals and 10% elitism. The linear rank-based fitness assignment is adopted from [4]. We run the proposed algorithm for 5000 generations in each of 50 runs. The characteristics of these test problems are given in Table I reproduced from [24].

The feasibility ratio $\rho = |\mathbf{F} \cap \mathrm{S}|/|\mathrm{S}|$ is determined experimentally in [27] by calculating the percentage of feasible solutions among 1 000 000 randomly generated individuals. For G2 and G3, a value of $k = 50$ was used in [27] which is different from 20 used in our experiments. Also, because we treat our equality constraints by relaxing them using a threshold value, the values of ρ would be slightly different for G3, G5, and G11. From Table I, we can clearly see that we have a variety of test functions involving both maximization (Max) and minimization (Min) problems with different types of objective functions (e.g., quadratic, nonlinear, or polynomial) and constraints (e.g., linear equality or nonlinear equality). We have used the proposed constraint handling scheme *without* any modifications (on the scheme and on the parameter setting) for solving all of these 11 problems. The results are shown in Table II.

One of the first observations is that all of the 50 runs produce feasible solutions for all the test problems and this assures that in a "real-world scenario," we produce usable solutions in every run of the algorithm. This is largely credited by the first phase of our algorithm which treats the constrained optimization problem as a constraint satisfaction problem. In problems G2 and G4, a feasible solution is always found in the random initial population itself and this is because the feasible space occupy a large portion of the search space in these problems. Even though G1 contains nine constraints, they are all linear and, hence, the feasible space is convex. Thus, by minimizing the distance to the feasible regions the constraints can be satisfied effectively and a feasible solution is found early in the

TABLE I
SUMMARY OF TEST CASES

Function	*n*	*Type of f*	ρ	*LI*	*NE*	*NI*	*a*
Min. G1	13	Quadratic	0.0111%	9	0	0	6
Max. G2	*k*	Nonlinear	99.8474%	0	0	2	1
Max. G3	*k*	Polynomial	0.0000%	0	1	0	1
Min. G4	5	Quadratic	52.1230%	0	0	6	2
Min. G5	4	Cubic	0.0000%	2	3	0	3
Min.G6	2	Cubic	0.0066%	0	0	2	2
Min. G7	10	Quadratic	0.0003%	3	0	5	6
Max. G8	2	Nonlinear	0.8560%	0	0	2	0
Min. G9	7	Polynomia1	0.5152%	0	0	4	2
Min.G10	8	Linear	0.0010%	3	0	3	6
Min. G11	2	Quadratic	0.0000%	0	1	0	1

LI – Linear Inequalities, *NE*-Nonlinear Equalities, *NI*-Nonlinear Inequalities, *a*-active constraints and feasibility ratio $\rho = |F \cap S|/|S|$.

search process. G3 has only one constraint and it was relaxed slightly by using a threshold of 0.001 to help find feasible solutions. Finding feasible solutions presents the greatest challenge in G5, where the combination of nonlinear equalities and linear inequalities causes some complications in locating feasible solutions. Nonlinear inequalities again cause some appreciable delay in finding feasible solutions for G8, while a combination of linear inequalities and nonlinear inequalities introduces some delay in finding feasible solutions for G7 and G10.

Since finding feasible solutions is independent of the objective function in the first phase of our algorithm, we can draw some conclusions about how constraints affect the GA's ability in finding feasible solutions.

1) Nonlinear constraints in general introduce more difficulty in finding feasible solutions than linear constraints. This can be understood by acknowledging that GAs are stochastic search techniques that work on reinforcement learning based on the fitness values. This fitness value is a trustable indicator of how far each solution is from the feasible region when there is a linear mapping between the decision variables and the constraints. However, when the mapping becomes nonlinear, then the distance between slightly infeasible solutions and completely feasible solutions in the decision space may be disproportionate to the differences in the constraint violation values. Hence, one step toward feasibility in the constraint space may involve an exhaustive search in the decision space.
2) The feasibility ratio and the type of constraints combine together to define the degrees of difficulty in the constraint satisfaction problem. We know from Table I that G2 and G4 have a large ρ value and a feasible solution was found even in the random initial population on all 50 runs in spite of the fact that nonlinear constraints were present in both problems. At the same time, even for problems with low ρ like G1, feasible solutions could be easily found because the constraints are all linear. G5 involves a combination of nonlinear constraints and very low ρ and this probably caused the difficulty in finding feasible solutions. Also, G6 and G10 which required relatively more generations to find feasible solutions involve a combination of low ρ and nonlinear constraints.

In the results for the best values found by the algorithm from Table III, we see that the algorithm has produced results extremely close to the optimum value known for all of the 11 test problems. For G1 even though the value of -15.0 could not be reached accurately, the algorithm consistently produced -14.9999 as the optimal value. In G2, again the optimal value of 0.8031 found is fairly close to the optimum value of 0.8035. In addition to reaching the optimum in G3, the algorithm produces a result closed to the optimum in G4. In G5–G7 and G9, the best results produced by the algorithm differ in decimal places from the optimum value. G8 was a very easy problem and the optimum results were obtained for all 50 runs. In G10, again the algorithm having produced 7060.55 was quite close to the 7049.33 optimum, which apparently no GA has reached, as shown in Table III. The best value for G11 is only better than the 0.75 optimum because of the tolerance in the equality constraint used. Also, note that the standard deviation over 50 runs for all the problems other than G5 and G10 is extremely small and the median is very near the best values obtained. This implies that the algorithm is robust in obtaining consistent results. Table III reproduced from [14] compares the best results obtained from the other algorithms in literature to those obtained with the proposed constraint handling scheme in the last column.

We can see that the algorithm has performed very well for all of the test problems reaching or obtaining values extremely near the global optimum. In fact, from the table, it is obvious

TABLE II
RESULTS USING THE PROPOSED CONSTRAINT HANDLING

Function	*Optimum Value*	*Worst*	*Best*	*Median*	*Standard deviation*	*MFG*	*Infeasible Runs*
Min. G1	-15	-11.9999	-14.9999	-14.9997	0.8514	11.24	0
Max. G2	0.803553	0.672169	0.803190	0.755332	0.0327	0	0
Max. G3	1.0	0.7855820	1.00009	0.94899	0.0489	31.68	0
Min. G4	-30665.5	-30651.9595	-30665.5312	-30663.3642	3.3103	0	0
Min. G5	5126.4981	6112.2231	5126.5096	5170.5294	341.2248	1807.82	0
Min.G6	-6961.8	-6954.3186	-6961.1785	-6959.5683	1.2691	289.52	0
Min. G7	24.306	35.881930	24.410977	26.735666	2.6139	53.22	0
Max. G8	0.095825	0.095825	0.095825	0.095825	0	9.28	0
Min. G9	680.63	684.131429	680.762228	681.706290	0.7443	5.84	0
Min.G10	7049.33	12097.4078	7060.55288	7723.166720	798.68	99.86	0
Min. G11	0.75	0.8094	0.7490	0.7493	0.0093	13.32	0

MFG – Mean generation number when the first feasible solution is found.

TABLE III
COMPARISON OF BEST RESULTS

Function	*Optimum Value*	*Koziel and Michalewicz 1999* [24]	*Runarsson and Yao 2000* [34]	*Deb 2000* [12]	*Farmani and Wright 2003* [14]	*Proposed Constraint Handling Scheme*
Min. G1	**-15**	-14.7864	**-15.0000**	**-15.0000**	**-15.0000**	-14.9999
Max. G2	**0.803553**	0.799530	**0.803515**		0.802970	0.803190
Max. G3	**1.0**	0.9997	**1.0000**		**1.0000**	**1.0000**
Min. G4	**-30665.5**	-30664.900	**-30665.539**	**-30665.537**	**-30665.500**	**-30665.5312**
Min. G5	**5126.4981**		**5126.4970**		5126.9890	5126.63049
Min.G6	**-6961.8**	-6952.100	**-6961.814**		**-6961.800**	-6961.17856
Min. G7	**24.306**	24.620	**24.307**	24.373	24.480	24.410977
Max. G8	**0.095825**	**0.095825**	**0.095825**		**0.095825**	**0.095825**
Min. G9	**680.63**	680.91	**680.63**	**680.63**	680.64	680.7622
Min.G10	**7049.33**	7147.90	**7054.32**	7060.22	7061.34	7060.5528
Min. G11	**0.75**	**0.75**	**0.75**		**0.75**	**0.7490***

* - a value lower than the global minimum has been obtained due to relaxation of the equality constraints

that stochastic ranking scheme proposed in [34] has produced the best results known so far for all the test problems. But a downside to that approach as pointed out in [14] is that for G10 only 6 out of 30 runs produced feasible solutions. In [14], 17 runs out of 20 produced feasible solutions, while our algorithm produced feasible solutions in all of the 50 runs. We in a way have tackled the same problem of domination between the objective function and constraint violation in assigning the fitness to the individual brought out in [14], but have solved it using nondominated ranking as opposed to using a probability factor P_f [34].

The efficiency of each of the above algorithms can be measured by comparing the number of function evaluations used by each of the algorithms. The number of function evaluations in general is equal to the $(\text{population size}) \times (\text{number of generations})$ as each solution is evaluated once in every generation. The algorithms presented in [14] and [24] used 1 400 000 function evaluations, while the algorithm in [34] used 350 000 function evaluations. The algorithm in [12] used different number of function evaluations based upon the difficulty of the problem ranging from 50 000 to 350 000. In comparison, the proposed algorithm herein used 50 000

function evaluations to produce the results shown in Table II for all the problems.

VII. Conclusion

We have implemented a two-phase GA to solve the constrained optimization problem. This algorithm has the advantage of being problem independent and does not rely on any parameter tuning. The proposed constraint handling scheme was tested on TCG-2. We assigned various levels of difficulty based on the number of disconnected components and peaks in the decision space. We provided the rationale behind using a nondominated ranking scheme for selecting individuals in a modified objective space of the objective function plotted versus the constraint violation. This in addition to the elitist scheme helps to provide a delicate balance between exploration and exploitation. The experiment verified that the proposed algorithm is an efficient design as the number of disconnected feasible components and the number of peaks increase. We then extended the tests to the 11 test problems commonly used in literature. Apart from finding optimal solutions that are extremely close to the optimum values, the proposed algorithm also found feasible solutions in every run. We attribute this to the first phase of the algorithm, where the complete search effort is devoted to finding feasible solutions.

References

[1] T. Bäck, "Selective pressure in evolutionary algorithms: A characterization of selection mechanisms," in *Proc. 1st IEEE Conf. Evol. Comput.*, 1994, pp. 57–62.

[2] E. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg, "Hyper-heuristics: An emerging direction in modern search technology," in *Handbook of Meta-Heuristics*, F. Glover and G. Kochenberger, Eds. Norwell, MA: Kluwer, 2003, pp. 457–474.

[3] E. Burke, G. Kendall, and E. Soubeiga, "A tabu-search hyper-heuristic for timetabling and rostering," *J. Heuristics*, vol. 9, pp. 451–470, 2003.

[4] Genetic Algorithm Toolbox for Use With Matlab, A. Chipperfield, P. Fleming, H. Pohlheim, and C. Fonseca. [Online]. Available: http://www.shef.ac.uk/~gaipp/ga-toolbox

[5] C. Coello, "Use of a self-adaptive penalty approach for engineering optimization problems," *Comput. Ind.*, vol. 40, pp. 113–127, 2000.

[6] ——, "Treating constraints as objectives for single-objective evolutionary computation," *Eng. Opt.*, vol. 32, pp. 275–308, 2000.

[7] ——, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art," *Comput. Methods Appl. Mech. Eng.*, vol. 191, pp. 1245–1287, 2002.

[8] D. Coit, A. Smith, and D. Tate, "Adaptive penalty methods for genetic optimization of constrained combinatorial problems," *Proc. INFORMS J. Comput.*, vol. 8, pp. 173–182, 1996.

[9] J. Culberson, "On the futility of blind search: An algorithmic view of no free lunch," *Evol. Comput.*, vol. 6, pp. 234–242, 1998.

[10] B. Craenen, A. Eiben, and J. Van Hemert, "Comparing evolutionary algorithms on binary constraint satisfaction problems," *IEEE Trans. Evol. Comput.*, vol. 7, pp. 424–444, 2003.

[11] K. Deb and S. Agrawal, "A niched-penalty approach for constraint handling in genetic algorithms," in *Proc. Int. Conf. Artif. Neural Netw. Genetic Algorithms*, 1999, pp. 235–243.

[12] K. Deb, "An efficient constraint handling method for genetic algorithms," *Comput. Methods Appl. Mech. Eng.*, vol. 186, pp. 311–338, 2000.

[13] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, pp. 182–197, Apr. 2002.

[14] R. Farmani and J. Wright, "Self-adaptive fitness formulation for constrained optimization," *IEEE Trans. Evol. Comput.*, vol. 7, no. 5, pp. 445–455, Oct. 2003.

[15] R. Fletcher, *Practical Methods of Optimization*, 2nd ed. New York: Wiley, 1990.

[16] C. Fonseca and P. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization," in *Proc. 5th Int. Conf. Genetic Algorithms*, 1993, pp. 416–423.

[17] ——, "Multiobjective optimization and multiple constraint handling with evolutionary algorithm—Part I: A unified formulation," *IEEE Trans. Syst., Man, Cybern.—A*, vol. 28, pp. 26–37, Jan. 1998.

[18] M. Gen and R. Cheng, "A survey of penalty techniques in genetic algorithms," in *Proc. Congr. Evol. Comput.*, 1996, pp. 804–809.

[19] L. Han and G. Kendall, "Investigation of a tabu assisted hyper-heuristic genetic algorithm," in *Proc. Congr. Evol. Comput.*, 2003, pp. 2230–2237.

[20] E. Hart, P. Ross, and J. Nelson, "Solving a real-world problem using an evolving heuristically driven schedule builder," *Evol. Comput.*, vol. 6, pp. 61–80, 1998.

[21] J. Joines and C. Houck, "On the use of nonstationary penalty functions to solve nonlinear constrained optimization problems with GAs," in *Proc. Congr. Evol. Comput.*, 1994, pp. 579–584.

[22] J. Kim and H. Myung, "Evolutionary programming techniques for constrained optimization problems," *IEEE Trans. Evol. Comput.*, vol. 1, no. 2, pp. 129–140, Jul. 1997.

[23] T. Kiyota, Y. Tsuji, and E. Kondo, "Unsatisfying functions and multiobjective fuzzy satisfying design using genetic algorithms," *IEEE Trans. Syst. Man, Cybern.—B*, vol. 33, pp. 889–897, Dec. 2003.

[24] S. Koziel and Z. Michalewicz, "Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization," *Evol. Comput.*, vol. 7, pp. 19–44, 1999.

[25] A. Kuri and J. Gutierrez, "Penalty function methods for constrained optimization with genetic algorithms: A statistical analysis," in *Proc. 2nd Mexican Int. Conf. Artif. Intell.*, 2002, pp. 108–117.

[26] Z. Michalewicz and N. Attia, "Evolutionary optimization of constrained problems," in *Proc. 3rd Annu. Conf. Evol. Program.*, 1994, pp. 98–108.

[27] Z. Michalewicz and G. Nazhiyath, "GENOCOP III: A coevolutionary algorithm for numerical optimization problems with nonlinear constraints," in *Proc. Congr. Evol. Comput.*, 1995, pp. 647–651.

[28] Z. Michalewicz, "Genetic algorithms, numerical optimization and constraints," in *Proc. Int. Conf. Genetic Algorithms*, 1995, pp. 151–158.

[29] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," *Evol. Comput.*, vol. 4, pp. 1–32, 1996.

[30] Z. Michalewicz and C. Janikow, "GENOCOP: A genetic algorithm for numerical optimization problems with linear constraints," *Commun. ACM*, pp. 122–133, 1996.

[31] Z. Michalewicz, K. Deb, M. Schmidt, and T. Stidsen, "Test-case generator for nonlinear continuous parameter optimization techniques," *IEEE Trans. Evol. Comput.*, vol. 4, pp. 197–215, 2000.

[32] D. Powell and M. Skolnick, "Using genetic algorithms in engineering design optimization with nonlinear constraints," in *Proc. 5th Int. Conf. Genetic Algorithms*, 1989, pp. 424–431.

[33] J. Richardson, M. Palmar, G. Liepus, and M. Hillard, "Some guidelines for genetic algorithms with penalty functions," in *Proc. Int. Conf. Genetic Algorithms*, 1989, pp. 191–197.

[34] T. Runarsson and X. Yao, "Stochastic ranking for constrained evolutionary optimization," *IEEE Trans. Evol. Comput.*, vol. 4, pp. 344–354, Sep. 2000.

[35] M. Schmidt and Z. Michalewicz, "Test-case generator *TCG-2* for nonlinear parameter optimization," in *Proc. Parallel Problem Solving From Nature*, 2000, pp. 539–548.

[36] P. Surry, N. Radcliffe, and I. Boyd, "A multi-objective approach to constrained optimization of gas supply networks: The COMOGA method," in *Proc. Evol. Comput. AISB Workshop*, 1995, pp. 166–180.

[37] M. Schoenauer and S. Xanthakis, "Constrained GA optimization," in *Proc. Int. Conf. Genetic Algorithms*, 1993, pp. 573–580.

[38] J. van Hemert and T. Bäck, "Measuring the searched space to guide efficiency: The principle and evidence on constraint satisfaction," in *Proc. Int. Conf. Parallel Problem Solving From Nature*, 2002, pp. 23–32.

[39] D. Whitley, "The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best," in *Proc. Int. Conf. Genetic Algorithms*, 1989, pp. 116–123.

[40] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 67–82, Apr. 1997.

Sangameswar Venkatraman received the B.S. degree in electrical engineering from the University of Madras, Madras, India, in 2002 and the M.S. degree in electrical engineering from Oklahoma State University, Stillwater, in 2004.

He was a student member of the IEEE Computational Intelligence Society. His research interests include evolutionary computation and its applications in business and engineering.

Gary G. Yen (S'87–M'88–SM'97) received the Ph.D. degree in electrical and computer engineering from the University of Notre Dame, Notre Dame, IN, in 1992.

He is a Professor in the School of Electrical and Computer Engineering, Oklahoma State University (OSU), Stillwater. Before he joined OSU in 1997, he was with the Structure Control Division, U.S. Air Force Research Laboratory, Albuquerque, NM. His research is supported by the DoD, DoE, EPA, NASA, NSF, and Process Industry. His research interest includes intelligent control, computational intelligence, conditional health monitoring, signal processing and their industrial/defense applications.

Dr. Yen was an Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS and the *IEEE Control Systems Magazine* during 1994–1999. He is currently serving as an Associate Editor for the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, the IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, and *IFAC Journal on Automatica and Mechatronics*. He served as the General Chair for the 2003 IEEE International Symposium on Intelligent Control held in Houston, TX, and will be the General Chair for the 2006 IEEE World Congress on Computational Intelligence to be held in Vancouver, Canada. On behalf of IEEE Robotic and Automation Society and later IEEE Control Systems Society, he was a Representative to the IEEE Neural Network Council Administrating Committee. He is currently serving as Vice President for the Technical Activities, IEEE Computational Intelligence Society and Editor of CIS Newsletter, *IEEE Connections*.