2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

10.1109/CEC.2006.1688315

A Self Adaptive Penalty Function Based Algorithm for Constrained Optimization

Biruk Tessema, *Member, IEEE* and Gary G. Yen, *Senior Member, IEEE*

Abstract— **This paper proposes a self adaptive penalty function for solving constrained optimization problems using genetic algorithms. In the proposed method, a new fitness value, called distance value, in the normalized fitness-constraint violation space, and two penalty values are applied to infeasible individuals so that the algorithm would be able to identify the best infeasible individuals in the current population. The method aims to encourage infeasible individuals with low objective function value and low constraint violation. The number of feasible individuals in the population is used to guide the search process either toward finding more feasible solutions or toward finding the optimum solution. The proposed method is simple to implement and does not need parameter tuning. The performance of the algorithm is tested on 13 benchmark functions in the literature. The results show that the approach is able to find very good solutions comparable to other state-of-the-art designs. Furthermore, it is able to find feasible solutions in every run for all of the benchmark functions.**

I. Introduction

Genetic algorithms (GAs) have been successfully applied to solve optimization problems in the fields of science and engineering. Most real-world optimization problems involve constraints of a certain kind. What constitutes the great challenges of the constrained optimization problems are various limits on the decision variables, the constraints involved, the interference among constraints, and the interrelationship between the constraints and the objective functions. If we take maximizing the profits of a certain production line as an example, then material consumption, labor cost, operating hours of the machines and other factors often limit the profit that could have been achieved otherwise.

Generally, constrained optimization problems are difficult to solve. Due to the presence of constraints, the feasible space might be reduced to some portion of the total search space, and finding feasible solutions itself could be a daunting challenge. Consider maximizing a function $f(x) = x_1 + x_2$ where the two variables are defined by $0 \le x_1, x_2 \le 1$ as shown in Figure 1. If there is no constraint involved, the optimum value would be $f(x) = 2$ when $x_1 = 1$ and $x_2 = 1$. But if an equality constraint $g(x) = x_1 - x_2 = 0.5$ is imposed upon variables x_1 and x_2, and assuming a resolution of up to two decimal places in the discrete search space, the feasible space would only be 0.5 percent of the total search space. And the optimum value in this case would be $f(x) = 1.5$, when $x_1 = 1$ and $x_2 = 0.5$.

Biruk Tessema is with the Department of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK 74075, USA (email: biruk.tessema@okstate.edu).

Gary G. Yen is with the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK 74078-5032, USA (email:gyen@okstate.edu).

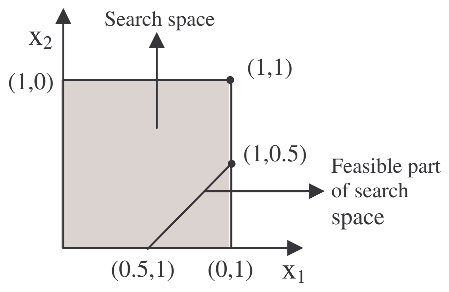

Figure 1. An example of constrained maximization problem

In general constrained optimization problem can be formulated as:

$$\text{Optimize } f(x) = f(x_1, x_2, \cdots, x_n), \tag{1}$$

subject to inequality constraints

$$g_i(x) \le 0 \quad i = 1, \cdots, k$$

and equality constraints

$$h_i(x) = 0 \quad i = k+1, \cdots, m.$$

The inequality constraints that satisfy $g_i(x) = 0$ at the global optimum solution are called active constraints. All equality constraints are active constraints. Without loss of generality, minimization problems are considered in this paper unless specified otherwise.

When dealing with constrained optimization problems, individuals that satisfy all of the constraints are called feasible individuals while individuals that do not satisfy at least one of the constraints are called infeasible individuals.

One of the major issues of constrained optimization is how to deal with the infeasible individuals throughout the search process. One way to handle infeasible individuals is to completely disregard them and continue the search process with feasible individuals only. But this has a drawback because GAs are probabilistic search methods and some of the information contained in the infeasible individuals could be unutilized. If the search space is discontinuous, then the GA can also be trapped in local minima. Therefore, different techniques have been developed to exploit the information contained in infeasible individuals.

The simplest and the earliest method of involving infeasible individuals in the search process is the static penalty function method. In this method, some penalty value is added to the fitness value of each infeasible individual so that it will be penalized for violating the constraints. Static penalty functions are popular due to their simplicity but they usually require different parameters to be defined by the user to control the amount of penalty added when multiple constraints are violated. These parameters are usually problem-dependant. To address this concerning issue, adaptive penalty functions are suggested recently where information gathered from the search process will be used to control the amount of penalty added to infeasible individuals. Adaptive penalty functions are easy to implement and they do not require users to define parameters.

In this paper, an adaptive penalty function strategy is proposed to solve constrained optimization problems using GA. Two types of penalties are added to each infeasible individual to identify the best infeasible individuals in the current population. The amount of each of the two penalties added is controlled by the number of feasible individuals currently present in the population. If there are few feasible individuals available, a higher amount of penalty is added to infeasible individuals with a higher amount of constraint violation. On the other hand, if there are sufficient numbers of feasible individuals present, then infeasible individuals with small objective function values will have small penalties added to their fitness value. These two penalties will allow the algorithm to switch between finding more feasible solutions and searching for the optimum solution at anytime during the search process. The proposed algorithm has the advantage of requiring no parameter tuning. Furthermore, the algorithm can find feasible solutions in problems having small feasible space compared to the total search space.

The remainder of this paper is organized as follows. Section II reviews related works of handling constrained optimization problems using GAs. In Section III, the proposed algorithm is presented in detail. Section IV discusses the results obtained for the benchmark functions. Finally, Section V presents some concluding remarks and relevant observations.

II. Literature Survey

Before presenting the proposed algorithm, we will first review some of the related works in this field of study. In doing so, we will not only present the basic ideas in each paper but also point out some of the weaknesses present to justify our proposed algorithm.

According to [2] constrained optimization algorithms can be categorized into five groups. Methods based on special representations and operators, repair algorithms and hybrid methods are irrelevant to the algorithm we proposed here. Therefore we have classified constrained optimization algorithms into only three groups: methods based on penalty functions, methods based on preference of feasible solutions over infeasible solutions, and methods based on multiobjective optimization techniques.

Penalty functions are by far the simplest and the most commonly used methods for handling constraints using GAs. In death penalty function methods, individuals that violate any one of the constraints are completely rejected. No information is extracted from those infeasible individuals. On the other hand, some penalty function approaches convert a constrained optimization problem into an unconstrained one by adding penalty values to individuals violating the constraints. The way the penalties are added determines the type of penalty function. If the penalties added do not depend on the current generation number and remain constant during the entire evolutionary process, then the penalty function is called static penalty function. In static penalty function methods the penalties are the weighted sum of the constraint violations. If, alternatively, the current generation number is involved in determining the penalties then the method is called dynamic penalty function method.

Although penalty functions are very simple and easy to implement they often require several parameters to be chosen heuristically by users. These parameters are problem-dependant and need prior knowledge of the degree of constraint violation present in a problem. Therefore, tuning the parameters leads to unnecessary computation for simple problems. Although dynamic penalty functions work better than static penalty functions, they require even more parameters to be tuned.

To solve the major drawback of penalty function methods, Farmani and Wright [4] proposed an adaptive penalty function strategy that takes feedback from the search process to define the penalty parameters. In their method, a two-stage dynamic penalty is imposed upon infeasible individuals to make sure that those infeasible individuals with low fitness value and low infeasibility value remain fit. In the first penalty the worst infeasible individual is penalized to have objective function value equal to or greater than the best feasible solution. The second penalty increases this value to twice the original value. All other individuals are penalized accordingly. The method requires no parameter tuning and no initial feasible solution. Although it produced good results for most of the test functions, the two-stage penalty method requires unnecessary high computation.

Lemonge and Barbosa [6] proposed a simple adaptive penalty function that also uses information from the population to tune the penalty parameters. In their approach, the average value of the objective function and the level of violation of each constraint during the evolution strategy is used to define the penalty parameters.

In [8], Runarsson and Yao introduced a stochastic ranking method to achieve a balance between objective and penalty functions stochastically. A probability factor P_f is used to determine whether the objective function value or the constraint violation value determines the rank of each individual. Although the method produced very good results for P_f = 0.45, it provided no assurance analytically that P_f = 0.45 is an optimal choice. Additionally the algorithm also failed to produce a feasible solution for 23 out of 30 runs for a particular problem.

In methods based on preference of feasible solutions over infeasible ones, feasible solutions are always considered better than infeasible ones. In [7], 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. The main drawback of the algorithm is that once there are many feasible individuals in the population, the infeasible individuals will be less used in the search process, and the algorithm will not be able to explore the search space. This may lead to the GA being stuck in a local optimum.

In [3], an algorithm based on the following ideas is proposed: 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 degrees 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 also argued that real coded representation was better suited for constrained optimization problems as it affords a greater chance of maintaining feasibility. 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. However, it requires some parameters, such as the critical distance $\overline{d}$ and n_f, to be defined by users.

Most recently, researchers have turned their attention to using multi-objective optimization techniques to solve constrained optimization problems. The main idea here is to convert the single objective optimization problem into a multi-objective optimization problem by treating the constraints as one or more objectives to be minimized. Afterward any multi-objective optimization technique can be employed to solve the problem.

In [11], a multi-objective optimization technique that uses population-based algorithm generator and infeasible solutions archiving and replacement mechanism is introduced. A given constrained optimization problem is first converted to a bi-objective optimization problem of minimizing objective function and constraint violation. By using population-based algorithm generator, an individual in the population may be replaced if it is dominated by a non dominated individual from the offspring population. By using infeasible solutions archiving and replacement mechanism, the best infeasible individual is kept in an archive and then replaced back to the population after some generations. The method plays a major role in problems where the feasible space is a small proportion of the search space, but it is computationally expensive and requires some parameters to be chosen *ad hoc*.

In [12], the authors suggested a two-phase algorithm that is based on multi-objective optimization technique. In the first phase of the algorithm, the objective function is completely disregarded and the constraint optimization problem is treated as a constraint satisfaction problem. The search is directed toward finding a single feasible solution. Once a feasible solution is found, the algorithm switches to Phase 2 where both satisfying the constraint violation and optimizing objective function are treated as bi-objective optimization problem. In this case, nondominated ranking is used to rank individuals and niching scheme is used to preserve diversity. The algorithm has an advantage in that it can always find feasible solutions for all problems. But its major drawback is that the algorithm switches to Phase 2 once a single feasible solution is found in Phase 1; and if the number of feasible individuals starts to decline while the algorithm is in Phase 2 there is no design the algorithm can switch back to Phase 1 to find more feasible individuals.

In [1], the authors proposed an algorithm that combines penalty function approach and multiobjective optimization technique for solving constrained optimization problems. The algorithm has a similar structure as penalty-based approach but borrows the ranking scheme from multiobjective optimization techniques. Initially, the *m* constraints are treated as *m* objectives to be optimized. Each individual will be ranked in two ways. First, it will be ranked with respect to its value in the original objective function. Then it will be ranked based on its non-dominance with respect to the *m* constraints. Finally, a new rank is assigned to each individual, which is the sum of the two ranks. However, the main problem with this method is that it did not perform well for problems containing equality constraints.

From the papers we have reviewed, we can observe that each type of algorithm possesses certain advantages and disadvantages. Penalty function methods are easy to implement but usually require some kind of parameter tuning. This makes them unreliable because the optimal values of these parameters cannot be known in advance, and they are often problem-dependent. Methods based on preference of feasible solutions over infeasible solutions, on the other hand, are also easy to implement but they do not properly exploit the information contained in infeasible individuals as they tend to rely solely on feasible individuals. Multi-objective optimization based algorithms require no specific parameter tuning, but they are often complex and computationally expensive. The algorithm we presented here aims to preserve the merits of the above algorithms while providing some solutions to the drawbacks observed. An ideal constrained optimization algorithm would be easy to implement, free of parameter tuning, and could find good solutions for every problem at every run. Adaptive penalty functions satisfy at least two of these requirements, as they can be easily programmed and do not require any parameter definition. The proposed algorithm is an adaptive penalty function based approach. In the following section, we will present the algorithm in detail.

III. Proposed Algorithm

The simplicity of penalty functions has made them the most commonly used methods for solving constrained optimization problems. In penalty functions, infeasible individuals will be penalized for violation of the constraints by adding some value to their original fitness value. Adding a penalty value will decrease the probability of an infeasible individual being selected for recombination. Generally, penalty functions have the following form:

$$F(x) = f(x) + \sum_{j=1}^{m} r_j c_j(x), \tag{2}$$

where $F(x)$ is the updated fitness value,
$f(x)$ is the original fitness value,
r_j is the j^{th} penalty coefficient,
$c_j(x)$ is the j^{th} constraint violation, and
m is the total number of constraint violations.

The major drawback of penalty function approach is determining the penalty coefficients. Usually, a prior knowledge of the problem is needed or repeated experiments should be done to determine the proper coefficients. In this paper, a self adaptive penalty function method is proposed. Two penalty values are added to infeasible individuals to make sure that if there is no feasible individual present in the population, then highly infeasible individuals are more penalized; and if there are few infeasible individuals, then those with high objective function value are more penalized. The two penalty values enable the algorithm to achieve the two main objectives of a constrained optimization algorithm: finding feasible solutions if there aren't any, and then searching for the optimum solution.

In constrained optimization, the main purpose of involving infeasible individuals in the search process is to exploit the information they might carry. Since GAs are stochastic search techniques, omitting infeasible individuals might lead to the GA being stuck in a local optima, especially in problems having discontinuous search space. In addition, in some highly constrained problems, finding a single feasible individual by itself might be difficult, and the search has to begin with all infeasible individuals.

The major difference among constraint handling algorithms is, therefore, their choice of the infeasible individuals to be involved in the search process. Figure 2 shows the types of infeasible individuals one might encounter during the search process. The figure shows a two-dimensional space with the y-axis being the objective function value and the x-axis being the sum of all constraint violations (to be defined in Equation (6)). Individual "D" has very low constraint violation but high objective value, while individual "A" has very low objective value but high constraint violation. On the other hand, individuals "B" and "C" have relatively low value of both objective function and constraint violation. Some algorithms prefer individuals such as "D" with low constraint violation [10], while others prefer individuals such as "A" with low objective function value [11], [12].

All of the individuals shown in the figure are important but at different stages and under different situations during the search process. For example, if the number of feasible individuals in the current population is very small or next to zero, then giving a higher probability of recombination to individuals like "D" will help in finding more feasible individuals. On the other hand, if we have many feasible individuals in the current population, then the main effort should be devoted to finding the global optimum solution. In this case, individuals like "D" will give us little information in finding the global optimum value, and instead we should place higher priority on individuals such as "A". This will help the algorithm to explore the entire search space. In other situations, individuals "B" and "C" should be preferred as they are close to the feasible region and also have relatively low objective value.

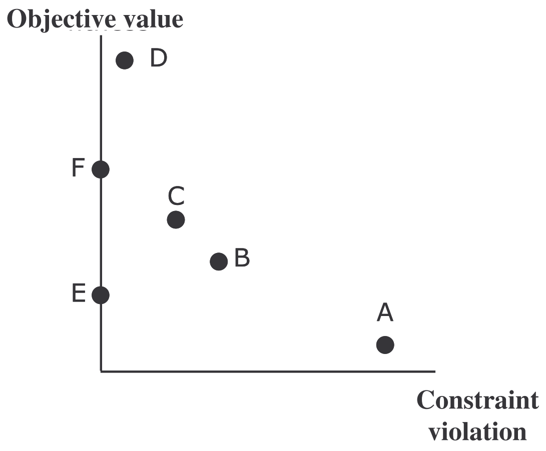

Figure 2. Possible combination of feasible and infeasible individuals in a population

Our algorithm is designed to achieve the above idea using a simple adaptive penalty function strategy. The algorithm keeps track of the number of feasible individuals in each population to determine the amount of penalty added to infeasible individuals. A new fitness value that we call "distance" value is introduced and the sum of this value and the penalty added will determine the rank of each individual.

A. Distance Value

To calculate the distance value the objective function value of all individuals will be calculated first and the smallest and the largest values will be identified as $f_{\min}$ and $f_{\max}$ respectively.

$$f_{\min} = \min_x f(x), \text{ and } f_{\max} = \max_x f(x). \tag{3}$$

Then each individual's fitness value will be updated by subtracting $f_{\min}$,

$$f'(x) = f(x) - f_{\min}, \tag{4}$$

where $f(x)$ is the objective function value and $f'(x)$ denotes the new fitness value. Then the fitness value will be normalized as follows:

$$f''(x) = \frac{f'(x)}{f_{\max} - f_{\min}}, \tag{5}$$

where $f''(x)$ is the updated normalized fitness value. After the above two transformations, each individual's fitness value will lie between 0 and 1 with 0 corresponding to the individual with the smallest fitness and 1 to the individual with the highest fitness.

Constraint violation of each infeasible individual is then calculated as the sum of the normalized violation of each constraint divided by the total number of constraints,

$$v(x) = \frac{1}{m}\sum_{j=1}^{m}\frac{c_j(x)}{c\max_j}, \tag{6}$$

where $c_j(x) = \begin{cases} \max(0, g_j(x)) & j = 1, \cdots, k \\ \max(0, |h_j(x)| - \delta) & j = k+1, \cdots, m \end{cases}$

$c\max_j = \max_x c_j(x)$,

$v(x)$ is constraint violation function,

δ is tolerance value (usually 0.001 or 0.0001), and

m is total number of constraints.

Then a "distance" value is formulated as follows:

$$d(x) = \begin{cases} v(x), & \text{if } r_f = 0 \\ \sqrt{f''(x)^2 + v(x)^2}, & \text{otherwise} \end{cases} \tag{7}$$

where

$$r_f = \frac{\text{number of feasible individuals}}{\text{population size}}.$$

Figure 3 shows the distance value of the individuals in Figure 2. From the above equation, we can observe that if there is no feasible individual in the current population, then the distance value will be equal to the constraint violation of the individuals. In this case, according to the distance value, an infeasible individual with small constraint violation will be considered better than another infeasible individual with higher constraint violation irrespective of their objective function value. This is the best way of comparing infeasible individuals in the absence of feasible individuals, and it will help us in approaching the feasible space very quickly. For example, in Figure 3, if the feasible individuals "E" and "F" were not present, then individual "D" would have the smallest distance value.

On the other hand, if there are one or more feasible solutions available, then the distance value will have the following properties that we summarized here

1. For feasible individuals, the distance value is equal to $f''(x)$. This implies that if we compare the distance value of two feasible individuals, then the individual with small objective function value will have small distance value.

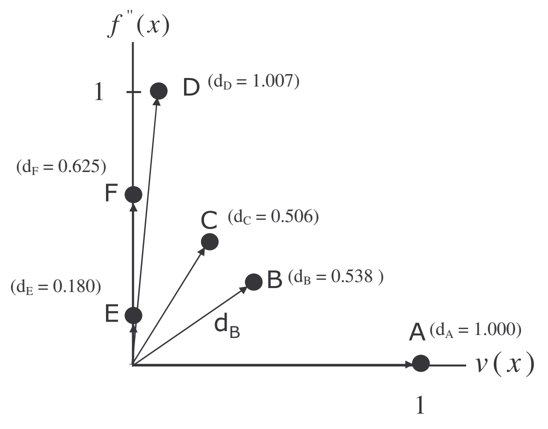

Figure 3. Distance value for individuals in Figure 2

2. For infeasible individuals, the distance value is the measure of the objective function value and the constraint violation. As can be seen in Figure 3, individuals near the origin (in the f''-v space) have lower distance value than those farther away from the origin. Individuals near the origin have both small objective function value and small constraint violation. Therefore, if we compare two infeasible individuals based on their distance value, then the one that has both low objective function value and low constraint violation will be considered better-fit.
3. If we compare the distance values of a feasible individual and an infeasible individual, then either one can have smaller value. But if the two individuals have the same objective value, then the feasible individual will have smaller distance value.

B. Two Penalties

From the property of the distance value, we can notice that it is another form of penalizing infeasible individuals for their constraint violation. This is because an infeasible individual having a certain objective value will have larger distance value than a feasible individual having the same objective value. In addition to the penalty imposed upon infeasible individuals this way, two other penalties will also be added. These penalties have two major purposes: 1) to further decrease the fitness of infeasible individuals and 2) to identify the best infeasible individuals based upon the number of feasible individuals in the population. The two penalties are formulated as follows:

$$p(x) = (1 - r_f)X(x) + r_f Y(x), \tag{8}$$

where

$X(x) = \begin{cases} 0, & \text{if } r_f = 0 \\ v(x), & \text{otherwise} \end{cases}$, and

$Y(x) = \begin{cases} 0, & \text{if } x \text{ is a feasible individual} \\ f''(x), & \text{if } x \text{ is an infeasible individual} \end{cases}.$

From the penalty function definition in Equation (8), we can observe that if the number of feasible solutions in the population is small (but not zero), then the first penalty

($X(x)$) will have more impact than the second penalty ($Y(x)$). Since the first penalty is formulated to have large value for individuals with large amount of constraint violation, in the case where there are few feasible individuals present in the population, individuals with high constraint violation will be more penalized. On the other hand if there are many feasible solutions in the population, the second penalty will have more effect than the first one in which case individuals with large objective function value will be more penalized than individuals with small objective function value.

C. *Final Fitness Formulation*

The final fitness value against which individuals will be compared or ranked is formulated as the sum of the distance value and the penalty value,

$$F(x) = d(x) + p(x)\,. \tag{9}$$

This fitness formulation is very flexible and will allow us to utilize infeasible individuals efficiently. Most constraint optimization algorithms are "rigid" in a sense that they always prefer certain types of infeasible individuals. For example, they might always give priority to those individuals with small constraint violation. But according to our new fitness formulation, the infeasible individuals that are considered valuable are not always similar. Here are some of the interesting properties of this fitness formulation:

1. If there is no feasible individual in the current population, $d(x)$ will be equal to the constraint violation ($v(x)$) and $p(x)$ will be zero. In this case the objective value of the individuals will be totally disregarded, and all individuals will be given a fitness value based on their constraint violation only. This will help us in finding feasible individuals before we try to search for the optimum value.
2. If there are feasible individuals in the population, then $d(x)$ will mainly determine which individuals are better fit. An individual with smaller distance value will be better than an individual with larger distance value; or stated in a different way, individuals with both low objective function value and low constraint violation value will be better than individuals that have high objective function value or high constraint violation or both.
3. If two individuals have equal or very close distance value, then the penalty value ($p(x)$) will determine which one is better. According to our penalty function, if the feasibility ratio (r_f) in the population is small, then the individual closer to the feasible space will be considered better. Otherwise the individual with smaller objective function value will be better.
4. If there is no infeasible individual in the population ($r_f = 1$), then individuals will be compared based on their objective function value alone.

IV. Experimental Results

The proposed algorithm was tested on 13 test functions from [9]. Table I shows some of the characteristics of these test functions. From the table we can observe that the test functions involve various types of problems. Some are maximization problems while others are minimization problems. The functions also vary from linear, nonlinear, quadratic, cubic, to polynomial. The number and the type of the constraints (LI- linear inequality, NE- nonlinear equality, and NI- nonlinear inequality) are also different. The feasibility ratio ρ is an estimate of the ratio of the feasible space to that of the entire search space. The number of active constraints is represented by a. n denotes the number of decision variables involved.

For each test function we applied the algorithm described here using a real coded GA. Rank based selection was used with BLX-α crossover operator (α = 0.5). For all test functions the population size was fixed at 100 and for all of the 50 independent runs performed the generation size was set to 5,000 (the total function evaluation is therefore 500 000). The test results of the algorithm are summarized in Table II. The best, the worst, the mean and the median results of each test function are reported. The numbers of infeasible runs from the 50 trials for each test function are also found in the same table. In addition we have highlighted the best values we found that are identical to the already know optimum values.

The first thing we notice from Table II is that our algorithm was able to find feasible solutions for all the 50 runs of all of the 13 test functions it was tested upon. For most of the test functions the feasibility ratio is less than 1 percent and the algorithm had to start with all infeasible individuals in the initial population. In addition nonlinear constraints posed some difficulty on finding feasible solutions. However the flexibility of our fitness formulation allowed the algorithm to find feasible individuals before proceeding to finding the global optimum value.

Our algorithm was also able to produce very good results for all of the benchmark functions. In fact, as can be seen in Table II, the proposed algorithm found the known optimum values in 6 of the 13 functions. In g01, g03, g08 and g12 values exactly equal to the already known optimum values are found. In test functions g11 and g13 however values less than the known optimum values are found because all equality constraints were relaxed by a threshold value of 0.0001, which is very small compared with others [12] where a threshold of 0.001 is used. In the rest of the test functions the algorithm found very good results close to the known optimum values. In g02 and g04, the optimum values could not be found but the best results found are very close. In g05 due to the low feasibility ratio and nonlinear constraint present in the test function the algorithm spent a lot of time searching for feasible individuals and the optimal result could not be found. In g07 and g09 the results our algorithm produced are again very close to the known optimum values. Nonlinear equality and nonlinear inequality constraints again imposed difficulty in finding the optimum value in g10. In fact the worst result produced by our algorithm is for this test function.

TABLE I

Summary of main characteristics of the benchmark problems

Test function		n	Type of function	ρ	LI	NE	NI	a
g01	Min	13	Quadratic	0.011%	9	0	0	6
g02	Max	20	Nonlinear	99.990%	1	0	1	1
g03	Max	10	Nonlinear	0.002%	0	1	0	1
g04	Min	5	Quadratic	52.123%	0	0	6	2
g05	Min	4	Nonlinear	0.000%	2	3	0	3
g06	Min	2	Nonlinear	0.006%	0	0	2	2
g07	Min	10	Quadratic	0.000%	3	0	5	6
g08	Max	2	Nonlinear	0.856%	0	0	2	0
g09	Min	7	Nonlinear	0.521%	0	0	4	2
g10	Min	8	Linear	0.001%	3	0	3	3
g11	Min	2	Quadratic	0.000%	0	1	0	1
g12	Max	3	Quadratic	4.779%	0	0	9^3	0
g13	Min	5	Nonlinear	0.000%	0	3	0	3

TABLE II

Results found using the proposed algorithm

Test Function	Optimum value	Best	Worst	Mean	Median	Standard Deviation	Infeasible runs
g01	-15.000	**-15.000**	-13.097	-14.552	-14.966	0.700	0
g02	0.803619	0.803202	0.745712	0.755798	0.789398	0.133210	0
g03	1.000	**1.000**	0.887	0.964	0.971	0.301	0
g04	-30665.539	-30665.401	-30656.471	306659.221	-30663.921	2.043	0
g05	5126.498	5126.907	5564.642	5214.232	5208.897	247.476	0
g06	-6961.814	-6961.046	-6943.304	-6953.061	-6953.823	5.876	0
g07	24.306	24.838	33.095	27.328	25.415	2.172	0
g08	0.095825	**0.095825**	0.092697	0.095635	0.095825	0.001055	0
g09	680.630	680.773	682.081	681.246	681.235	0.322	0
g10	7049.331	7069.981	7489.406	7238.964	7201.017	137.773	0
g11	0.750	**0.749**	0.757	0.751	0.750	0.002	0
g12	1.000000	**1.000000**	0.999548	0.999940	1.000000	0.000141	0
g13	0.053950	**0.053941**	0.885276	0.286270	0.054713	0.275463	0

TABLE III
Comparison of best results

Test Function	Optimum value	Koziel and Michalewiez [5]	Runarsson and Yao [8]	Deb [3]	Farmani and Wright 2003 [4]	Venkatraman and Yen [12]	Proposed Algorithm
g01	-15.0000	-14.7860	-15.0000	-15.0000	-15.0000	-14.9999	**-15.0000**
g02	0.803619	0.799530	0.803515	NA	0.802970	0.803190	0.803202
g03	1.0000	0.9997	1.0000	NA	1.0000	1.0000	**1.0000**
g04	-30665.539	-30664.900	-30665.539	-30665.537	-30665.500	-30665.531	-30665.401
g05	5126.498	NA	5126.497	NA	5126.989	5126.630	5126.907
g06	-6961.814	-6952.100	-6961.814	NA	-6961.800	-6961.179	-6961.046
g07	24.306	24.620	24.307	24.373	24.480	24.411	24.838
g08	0.095825	0.095825	0.095825	NA	0.095825	0.095825	**0.095825**
g09	680.630	680.910	680.630	680.630	680.640	680.762	680.773
g10	7049.331	7147.900	7054.316	7060.220	7061.340	7060.553	7069.981
g11	0.750	0.750	0.750	NA	0.75	0.749	**0.749**
g12	1.000000	NA	1.000000	NA	NA	NA	**1.000000**
g13	0.053950	NA	0.053957	NA	NA	NA	**0.053941**

Table III compares the best results of the proposed algorithm with 5 other related algorithms. Of these algorithms stochastic ranking [8] had produced the best results. However this algorithm produced feasible solutions in only 6 out of the 30 runs performed in test function g10. The same case occurred in [4] where only 17 out of 20 runs produced feasible solutions. For the same algorithm feasible solutions could be found only for 9 out of 20 runs for g05. But our algorithm produced feasible solutions in all runs for both test functions. In [12] the algorithm was able to find feasible solutions for every run in all test functions, however the algorithm could find the optimum solutions only for four test functions. In general we can say that our algorithm can perform as good as or in some cases better than other algorithms in the related field.

V. Conclusion

In this paper, we have developed a self adaptive constrained optimization algorithm that is free of any parameter tuning. The algorithm can find feasible solutions in every run for problems having small feasible space compared to the search space. The main objective of the algorithm is to exploit the information hidden in infeasible individuals efficiently by selecting the proper individuals at different stages of the evolutionary process and under different conditions. To achieve this objective, a new fitness value, called distance value, and two penalty functions are introduced. The number of feasible individuals in the population determines the values of the distance value and the two penalties. The performance of our algorithm was tested on 13 benchmark functions, and the algorithm was able to find competitive results with all well-regarded algorithms in literature. Furthermore, the algorithm was able to find feasible, quality solutions in all of the runs in the test functions.

References

[1]. J. Aidanpaa, J. Anderson, and A. Angantyr, “Constrained optimization based on a multiobjective evolutionary algorithm,” in *Proceedings of Congress on Evolutionary Computation*, Canberra, Australia, pp. 1560-1567, 2003.

[2]. C.A.C. Coello, “Theoretical and numerical constraint handling techniques used with evolutionary algorithms: a survey of state of the art,” *Computer Methods in Applied Mechanics and Engineering*, vol. 191, pp. 1245-1287, 2002.

[3]. K. Deb, “An efficient constraint handling methods for genetic algorithms,” *Computer Methods in Applied Mechanics and Engineering*, vol. 186, pp. 311-338, 2000.

[4]. R. Farmani and J. Wright, “Self-adaptive fitness formulation for constrained optimization,” *IEEE Transaction on Evolutionary Computation*, vol. 7, no. 5, pp. 445-455, 2003.

[5]. S. Koziel and Z. Michalewicz, “Evolutionary algorithms, homorphous mappings, and constrained parameter optimization,” *Evolutionary Computation,* vol. 7, pp. 19-44, 1999.

[6]. A.C.C. Lemonge and H.J.C. Barbosa, “An adaptive penalty scheme in genetic algorithms for constrained optimization problems,” in *Proceedings of Genetic and Evolutionary Computation Conference*, New York, NY, pp. 287-294, 2002.

[7]. D. Powell and M. Skolnick, “Using genetic algorithms in engineering design optimization with nonlinear constraints,” in *Proceedings of the International Conference on Genetic Algorithms*, Urbana-Champaign, IL, pp. 424-431, 1993.

[8]. T.P. Runarsson and X. Yao, “Stochastic ranking for constraint evolutionary optimization,” *IEEE Transaction on Evolutionary Computation,* vol. 4, pp. 344-354, 2000.

[9]. T.P. Runarsson and X. Yao, “Search biases in constrained evolutionary optimization,” *IEEE Transaction on Systems, and Cybernetics*, *Part C*, vol. 35, no. 2, pp. 233-2443, 2005.

[10]. M. Sheng-jing, S. Hong-Ye, C. Jian, and W. Yue-Xuan, “An infeasibility degree selection based genetic algorithms for constrained optimization problems,” *IEEE International Conference on Systems, Man and Cybernetics*, pp. 1950-1954, 2003.

[11]. Y. Wang and Z. Cai, “A multiobjective optimization based evolutionary algorithm for constrained optimization,” in *Proceedings of Congress on Evolutionary Computation*, Edinburgh, UK, pp. 1081-1087, 2005.

[12]. S. Venkatraman and G.G. Yen, “A generic framework for constrained optimization using genetic algorithms,” *IEEE Transaction on Evolutionary Computation*, vol. 9, no. 4, pp. 424-435, 2005.