A SELF-ADAPTIVE GENETIC ALGORITHM FOR CONSTRAINED OPTIMIZATION

By

BIRUK GIRMA TESSEMA

Bachelor of Science in Electrical Engineering

Bahir Dar University

Bahir Dar, Ethiopia

2004

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2006

A SELF-ADAPTIVE GENETIC ALGORITHM FOR CONSTRAINED OPTIMIZATION

Thesis Approved:

Dr Gary G. Yen, Chairman

Dr Guoliang Fan

Dr Louis G. Johnson

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

I am deeply indebted to my advisor, Dr. Gary Yen, for he has constantly supported, guided and motivated me for the past one and a half year. He has greatly enriched my knowledge with his exceptional insights into engineering. Without his help, this work would not have been possible.

I am also thankful to the fellow graduate students; Wenf Leong, Moayed Daneshyari and Yonas Woldesenbet for their valuable assistance and encouragement.

I am grateful that I have very special parents. Mom and Dad I thank you for raising me the way you did. You have always been there for me, shown me the right way, supported and encouraged me all my life.

Aryam and Eden thank you for being my role models. I have always followed your foot steps and you have always led me to the right way. Misgana and Yabets you are the best brothers.

I would also like to thank Dr Guoliang Fan and Dr Louis G. Johnson for attending my defense.

Above all, I cannot express my full gratitude to my Father God who is the source of all good things.

Biruk Tessema

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Overview

In our day to day lives we usually encounter situations where we have to make decisions. The decision could be as simple as choosing a route to follow from one place to another. Or it could be a more difficult one that requires days or months to analyze (e.g. buying a house). In most of the decisions we make, with or without conscious, we intend to minimize (e.g. our energy and expenses) or maximize (e.g. pleasure and efficiency) certain aspects. This process of minimizing or maximizing a specific problem is technically termed as *optimization*.

When dealing with optimization problems, various things should be analyzed in order to come to the conclusion that one solution is better than another. For example if we consider an optimization problem of maximizing the profits of a certain production line, then we should take into consideration raw materials, labor, machinery and other additional factors in order to determine how we can achieve profit maximization. In mathematical terminology these determining factors are called *decision variables*. An *objective function* based on the decision variables can then be used to determine what combination of material, labor and machinery would give a maximum profit.

An optimization problem could be a straight forward problem where there are no conditions to be met. In more realistic problems, however, there are certain constraints imposed on the decision variables. In our previous example of optimizing the profit of a production line, in real world we can only have a limited amount of raw materials, a fixed number of human labor, and we can not run our machines indefinitely. These kinds of complex optimization problems where there are constraints involved are called *constrained optimization* problems.

In Figure 1.1 an optimization of a simple problem expressed mathematically as $f(x) = x_1 + x_2$ is shown. For this problem our decision variables are x_1 and x_2 and they are defined by $0 \le x_1$ and $x_2 \le 1$. If there are no constraint involved, the maximum value attained by the function would be $f(x) = 2$, which occurs when x_1=1 and x_2=1. But if we impose a hard constraint, which is defined by the function, $g(x) = x_1 - x_2 = 0.5$, on the decision variables x_1 and x_2 the final result would be changed. The optimum value in this case would be $f(x)$=1.5, when x_1=1 and x_2=0.5.

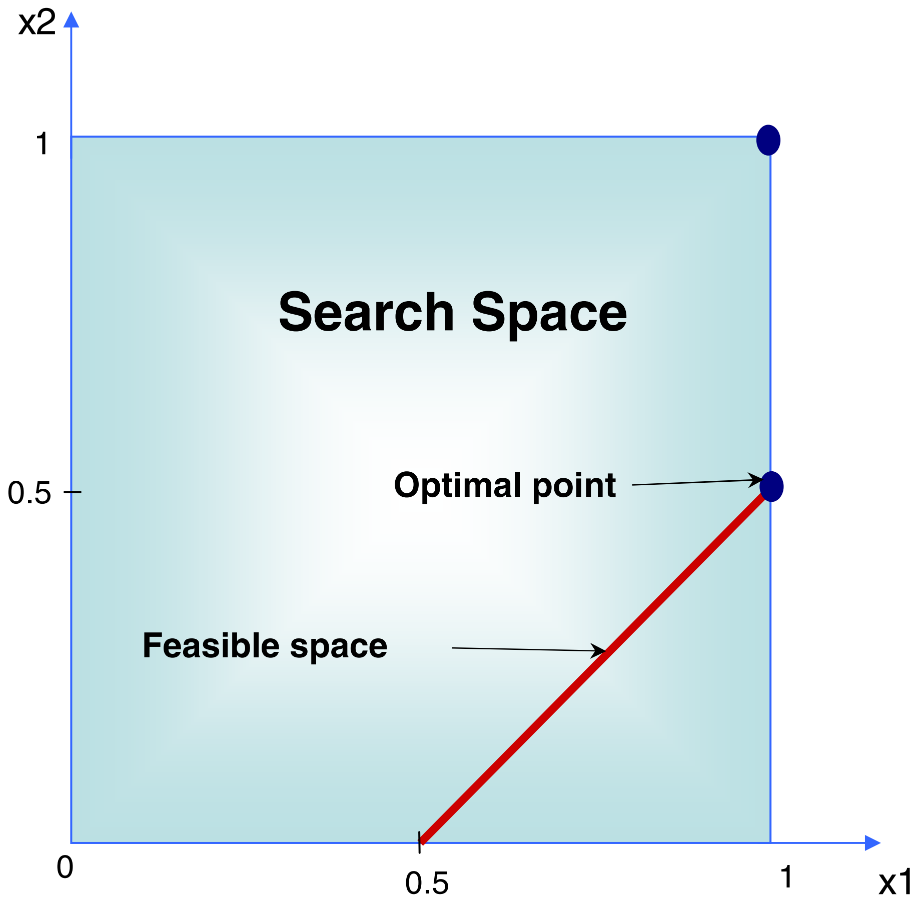


Figure 1.1 An example of constrained maximization problem

1.2 Problem Definition

In general, constrained optimization problem, as defined in [28], can be represented mathematically as follows:

$$\text{Optimize } f(\vec{x}) = f(x_1, \cdots, x_n) \tag{1.1}$$

Subject to

$$g_i(\vec{x}) \le 0 \qquad i = 1, \cdots, k \tag{1.2}$$

$$h_i(\vec{x}) = 0 \qquad i = k+1, \cdots, m. \tag{1.3}$$

The objective function $f(\vec{x})$ is defined on a search space $S \subseteq \Re^n$. Usually the search space is an n-dimensional hyperbox in $\Re^n$. The domains of the decision variables, x_i, $i = 1,...,n$ are defined by their own lower and upper bounds as shown below.

$$l(x_i) \le x_i \le u(x_i). \tag{1.4}$$

In Equation (1.2) $g_i(\vec{x})$ corresponds to the i^{th} inequality constraint and in (1.3) $h_i(\vec{x})$ corresponds to the i^{th} equality constraint. There are m number of constraints totally. The inequality constraints that satisfy $g_i(\vec{x}) = 0$ at the global optimum solution are called *active constraints*. All equality constraints are therefore considered as active constraints. The presence of equality and inequality constraints will restrict our search space to a *feasible space* $F \subseteq S$, where a usable solution could be found. In Figure 1.1 we can see that due to the presence of the equality constraint, the feasible space would only be 0.5 percent of the total search space (assuming a resolution up to two decimal places in the discrete search space).

The goal of constrained optimization is finding the decision variables that would give an extremum (maximum or minimum) value for the objective function and that would satisfy the equality and inequality constraints. An extremum value can be *local* or *global*. A local extremum is an optimum in a finite neighborhood .The global extremum is the actual highest or lowest functional value. In multi-modal optimization we are interested in both local and global optimum values. But here we focus only on global optimal values. In addition, without loss of generality we will consider only minimization problems. Maximization problems can easily be converted to minimization problems by multiplying them with -1.

1.3 Constraint Handling Techniques

Generally speaking there are two classes of algorithms that can be used to solve constrained optimization problems [12]. *Specific* methods (e.g., cutting plane method, the reduced gradient method, and the gradient projection method [11], [31]) exploit the mathematical structure of the constraint (e.g. gradient) and are applicable to a special type of constraints. These methods can be applied to problems having convex feasible regions only or to problems having few variables as a large number of variables will cause computational complexity.

On the other hand *generic* methods do not exploit the mathematical structure (i.e. linear or non-linear) of the constraint and they can be easily applied to any problem without much change in the algorithm. Therefore generic constraint handling techniques, such as penalty functions, can be used with generic search methods for solving complex constrained optimization problems. *Genetic algorithms (GAs),* which are a particular class of evolutionary algorithms (EAs) that use techniques inspired by evolutionary biology, are stochastic search methods that are recently very popular for solving constrained optimization problems.

1.4 Research Approach and Goal

This research focuses on developing an adaptive penalty function strategy for solving constrained optimization problems using EAs. The research has two major goals. The first goal is designing a reliable algorithm that would always guarantee finding a very good usable solution. A good solution refers to a close approximation of the global optimum solution. Most of the existing EA based constraint optimization algorithms either fail to produce usable solutions in every run of the algorithm, or they fail to produce good solutions every time. The second goal is designing an algorithm that is free of any parameter tuning. As we may encounter various types of constrained optimization problems in real world, tuning the algorithm for each type of problem would make the algorithm impractical.

To achieve the two goals, the algorithm is designed to encourage infeasible individuals with low constraint violation and better objective function value so as to facilitate finding feasible individuals in each run as well as producing quality results. The commonly used single penalty function strategy is also modified to a two penalty function method, where one penalty function will assist in finding feasible individuals while the other will assist in finding the optimal solution. Furthermore priority is imposed to finding feasible individuals before looking for optimal solutions so that the algorithm can find feasible solutions in problems having small feasible space compared to the search space.

1.5 Document Organization

The remainder of the thesis is organized as follows. Chapter II presents an introduction to EAs. This chapter will give a highlight into the common operations in a typical evolutionary algorithm followed by a brief introduction to constraint handling using EA. Chapter III provides an overview of related works of handling constrained optimization problems using EAs. In Chapter IV, the proposed algorithm is presented in detail. In the proposed method a new fitness, called distance, will be assigned to all solutions and two penalties will be applied to infeasible solutions in order to efficiently utilize the information constrained in infeasible solutions. Chapter V discusses the numerical analysis performed and the results obtained for some selected testing functions. Chapter VI presents some concluding remarks and relevant observations. The Appendix part contains mathematical formulation of the test suites used for the experimental analysis part.

CHAPTER II

EVOLUTIONARY ALGORITHMS

2.1 Introduction

According to [40] global search and optimization techniques can be broadly classified into three categories: enumerative, deterministic, and stochastic. Enumerative techniques are the simplest search schemes where within some defined search space each possible solution is evaluated. However these techniques become inefficient as the size of the search space become very large. Moreover finding acceptable solutions within a reasonable time becomes difficult as most of the real world problems are computationally expensive.

Deterministic algorithms, which are usually based upon graph or tree search algorithms, try to solve this problem by incorporating some type of problem knowledge. Greedy algorithms and hill climbing algorithms are the most common deterministic algorithms. These algorithms work by repeatedly expanding a node, examining all possible successors and then by further expanding the most promising nodes. Although deterministic algorithm have been successfully applied for solving a variety of problems, they are usually ineffective when applied to NP-Complete problems or high dimensional problems. This is because they require problem specific knowledge to direct the search process and finding this information in very large search spaces is often difficult.

Since most real world problems are high dimensional, multimodal, discontinuous and/or NP-Complete, enumerative and deterministic method can not be effectively used as optimizers. Stochastic techniques are developed for solving these kinds of difficult problems. Stochastic algorithms work with a group of randomly chosen solutions where fitness value is assigned to the solutions based on their performance. These techniques can not guarantee finding an optimal solution every time; however they generally provide good solutions for a wide variety of problems.

Evolutionary algorithms (EAs) are stochastic search methods based on the evolutionary ideas of natural selection and genetic. They are zero order methods that require only values of the function to optimize. This allows EAs to tackle optimization problems for which standard gradient based optimization methods that require the existence and computation of derivatives are not applicable.

In the past few decades, EAs have received significant attention regarding their potential as global optimization techniques and EA based algorithms have been successfully applied to solve optimization problems in the fields of science and engineering [17]. Over the years EAs have been subject to extensive experimentation and theoretical analysis but the basic concept of EA is designed to simulate processes in natural system necessary for evolution, specifically those that follow the principles of survival of the fittest coined by Charles Darwin [13].

2.2 EA Implementation

Generally EA is implemented as computer program where a population of abstract representations (called *chromosomes* or *genotypes*) of candidate solutions (called *individuals* or *phenotypes*) to an optimization problem evolves from *generation* to generation toward finding better solutions. The pseudocode for a typical EA is shown in Figure 2.1 [35] and the corresponding flow chart is shown in Figure 2.2. The evolution process starts from a *population* of randomly generated individuals which is called the *initial population*. Then in each *generation*, if the termination condition is not satisfied, the *fitness* of every individual in the population is evaluated to determine the better-fit individuals. The fitness of an individual measures how well the individual satisfies the optimality condition. Based on their fitness values, multiple individuals are selected from the current population to be modified to form a new population. The selected individuals are then modified by applying genetic operators. There are two kinds of modification operations that are commonly used: crossover and mutation. In *crossover* operation, two *parent individuals* will mate (or recombine) to produce an offspring individual. Crossover aims at swapping portions of genetic material between two individuals. In *mutation* operation a parent individual will be modified to create an offspring. The new population created by applying genetic operations (i.e. the *offspring* population) will then *replace* some of the solutions in the original population. This process is repeated until the termination condition (which is usually the *maximum generation* number) is reached; in which case the result obtained will be reported as the optimal solution.

```
Procedure for Evolutionary Algorithm
Begin
    $gen \leftarrow 0$
    initialize population
    While (not termination -condition) do
    Begin
        $gen \leftarrow gen + 1$
        select individuals for reproduction
        apply operators
        evaluate newborn offspring
        replace some parents by some offspring
    End
End
```

Figure 2.1 The pseudocode of an evolutionary algorithm

2.2.1 Representation: - There are two different ways of representing solutions in EAs. Traditionally, a solution is represented as a bit string of length *m* where each individual represents one sample point in binary space of size 2^m [3]. This way of representing solutions with binary strings of 0s and 1s is called *binary representation*. Recently *real representation* of solutions is employed where the solutions are represented in actual real numbers.

The original Genetic algorithm (GA), which is a branch of EA, was based on binary coding. This coding was similar to the chromosome structure of biological genes and therefore it was easy to explain in biology genetic theory. In addition various genetic operators could be easily utilized that are similar to the actual biological operators. However this representation has some drawbacks when applied to multidimensional, high-precision numerical problems when compared to real representation. First, binary representation doesn't provide adequate precision to solutions unlike the real representation. In addition with the same number of digits real representation has larger range. Moreover the real coded genes have the ability to exploit the gradualness of continuous variables (i.e. small changes in the variables correspond to small changes in the objective function). Therefore most algorithms prefer to employ real representation of solutions.

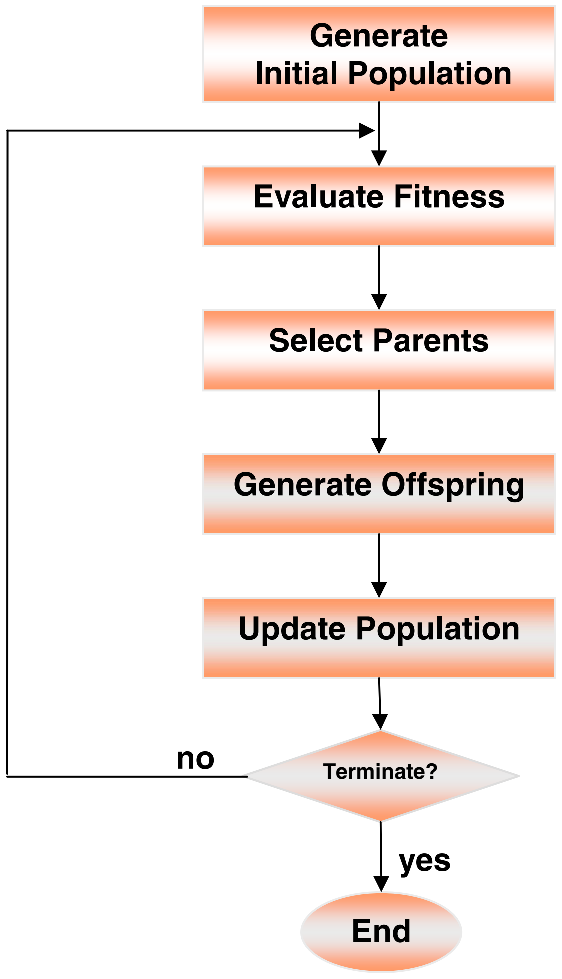


Figure 2.2 Flow chart of an evolutionary algorithm

2.2.2 Selection: - Selection is the process of selecting parents for producing offspring for successive generations. The selection operator plays an important role in directing the population toward the optimal solution. In most selection policies the best individuals are favored to become parents for the next generation. The three commonly used selection strategies are the *roulette-wheel*, *rank based selection* and *tournament selection* [3].

In *roulette-wheel* selection strategy, individuals with higher fitness values will be given high probability of selection. In this method individuals are mapped on a line segment such that each individual segment is equal to its proportion. The proportion of an individual is the ratio of the objective function of an individual to the sum of the value of the objective function of all of the individuals in the population. Afterwards a random number is generated and the individual whose segment spans the random number will be selected. This process will be repeated until the mating pool is filled with the required number of parents. The main drawback of this selection strategy is that it favors the most fit individuals and is unevenly biased based on the objective value of individuals.

On the other hand, in *rank* based selection, probabilities will be assigned to individuals based on their relative ranking, ignoring their absolute fitness values. First the population will be sorted based on the objective function. Then the fitness value is assigned based on their rank in the population. This ranking method overcomes the scaling problems of the roulette-wheel selection and prevents stagnation of the population. Since the reproductive range is limited, no individual will be allowed to generate an excessive number of offspring and hence this ranking introduces a uniform scaling at the same time maintains the selection pressure to have better diversity.

In *tournament* selection some number of individuals will be chosen randomly from the population and the best individual out of them will be selected. This process will be repeated until the mating pool is filled. The size of the tournament, which is equal to the number of individuals chosen randomly from the population, can vary based on the need. Generally by increasing the tournament size the selection pressure can be increased since larger tournament winner will have better fitness value than a small tournament winner.

2.2.3 Crossover operator: - Crossover operators can be categorized as binary coded and real coded crossovers in order to represent the kind of crossovers used in binary represented EA and real represented EA respectively. The three commonly used binary crossover operators are *one-point*, *two point* and *uniform* crossovers [17].

In one-point crossover, given two parent binary strings of equal length, a crossover point will be chosen randomly in those two strings and portions of the bit strings after this point will be swapped to generate two offspring individuals. Two point crossover is almost as simple as one point crossover. In this scheme, two crossover points are picked within the binary strings of the two parents at random with the stipulation that there must be at least one bit in between the two crossover points. Then the strings between the two crossover points in each parent are swapped to create two children. Examples of the types of binary crossovers described above are shown in Figure 2.3.

Uniform crossover makes use of two parents and attempts to create a crossover point between every two bits within a bit string. Crossovers are made with a probability of 0.5. The result of uniform crossover is two children with much recombinant information interspersed within them.

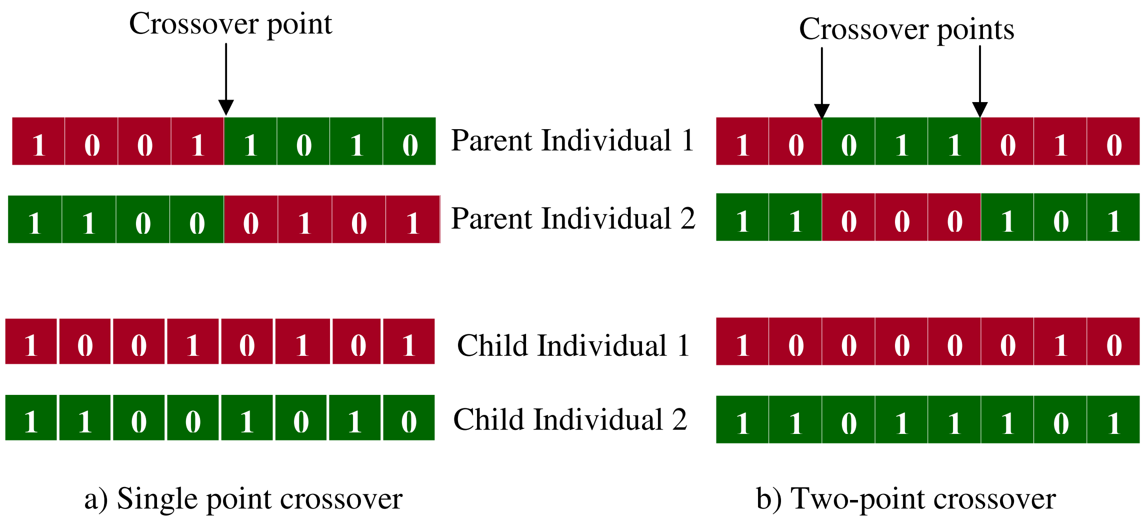


Figure 2.3 Binary crossover

One of the simplest real coded crossover operators is the *arithmetic crossover* [13]. This operator makes use of two parents and averages the two up with respect to a bias. The following two formulae will be used to generate two child individuals:

$$p_1 \times bias + p_2 \times (1 - bias) \text{ and,}$$
$$p_1 \times (1 - bias) + p_2 \times bias\,. \tag{2.1}$$

where p_1 *and* p_2 are the parent values. The bias can be selected initially, at random, or via a heuristic. Although fast and efficient, this operator has a drawback because it has a bias toward the center of the range of the two parents, and therefore can lead to a loss of diversity.

The *blend crossover* (BLX-α) operator [13] is another fairly simple real coded operator. In BLX-α, offspring are generated in two steps as follows:

(1) Choose two parent vectors $\vec{x}^1$ and $\vec{x}^2$ randomly from the population,

(2) Generate the i^{th} element x_i^o of the offspring vector $\vec{x}^o$ randomly from the interval [X_i^1, X_i^2] given as follows:

$$\begin{aligned} X_i^1 &= \min(x_i^1, x_i^2) - \alpha d_i \\ X_i^2 &= \min(x_i^1, x_i^2) + \alpha d_i\,. \\ d_i &= \left|x_i^1 - x_i^2\right| \end{aligned} \tag{2.2}$$

where α is a positive number between 0 and 1. This formulation allows the BLX-α crossover operator to choose a child on a baseline that extends some distance beyond the two parents. This allows for better placement of children than the arithmetic crossover; however, the children will still be biased to lie on baselines between sets of parents.

The Unimodal Normally Distributed Crossover (UNDX) is slightly more complicated crossover operator. It first selects three parents at random from the population. Next, it finds the midpoint of the first two parents and calls it x^p. Then, it finds the difference vector of the first two parents as $d = x^1 - x^2$. The line containing the first two parents is called the primary search line and the value *D* is computed as the distance from the third parent to the primary search line. These terms are then combined to form a child x^c via the following equation:

$$x^c = m + d\xi + D\sum_{i=1}^{n-1} \eta_i e_i$$
$$\xi \sim (0, \sigma_\xi^2)\,. \tag{2.3}$$

$$\eta_i \sim (0, \sigma_\eta^2)$$

where e_i are the set of vectors orthogonal to the primary search space and σ_ξ and σ_η are standard deviations determined empirically. The operation of this crossover is elaborated in Figure 2.4. This crossover has the advantage of preserving the mean vector and covariance matrix of the parent population thereby maintaining a similar distribution to the parent population in child generations. The population still, however, remains on search lines near the parent population. Therefore it does not cover the search space very well.

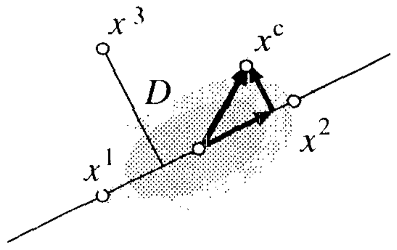


Figure 2.4 Unimodal Normally Distributed Crossover

2.2.4 Mutation operator: - Similar to the crossover operator the mutation operator can be divided into two categories: binary and real mutation. In the simplest binary mutation, the *bitwise mutation*, one bit value of a parent individual is changed from ‘0’ to ‘1’ or from ‘1’ to ‘0’ in order to generate an offspring individual. The operation of this crossover is shown in Figure 2.5.

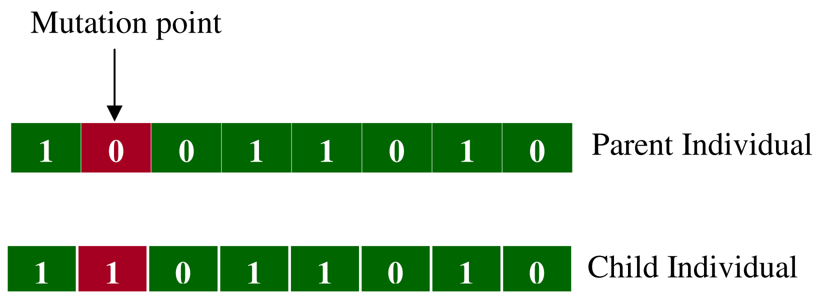


Figure 2.5 Bitwise mutation

The simplest real coded mutation operator is the *uniform mutation*. In uniform mutation a gene x_i is replaced by a uniform random number from the interval $[l(x_i), u(x_i)]$, where $l(x_i)$ and $u(x_i)$ are defined as in Equation (1.4). This mutation provides values which are not included in the initial population and it is important to use this mutation operator when the diversity of the initial population is low.

However the most population real coded mutation operator is the *Gaussian mutation*, which modifies all components of a solution by adding a random noise:

$$x_i^m = x_i + N(0, \vec{\sigma}) \tag{2.4}$$

where $N(0, \vec{\sigma})$ is a Gaussian random number with mean zero and standard deviation $\vec{\sigma}$. Unlike boundary mutation and uniform mutation which are used to search solutions globally, Gaussian mutation is used to search locally.

2.3 Classification of EAs

EAs are generally classified into three major branches:

1. *Evolution strategies (ES)* usually work with a single individual or with a population where individuals are represented as real numbers. The main operator used with ES is the mutation operator. In each generation a parent individual will be mutated to generate an offspring. The replacement step is deterministic where the fittest from the parent and the offspring becomes the parent of the next generation.
2. *Evolutionary Programming (EP)* emphasizes the phenotype space. As in ES, each individual in the population will generate one offspring. Moreover the only evolution operator used is mutation. Their major difference from ES is that in each generation the best *P* individuals among parents and offspring become the parent of the next generation.
3. *Genetic algorithms (GAs)* were originally designed to work with individuals represented in binary. But in most recent works, real valued representations are also used. Based on the selection scheme used some individuals will be

selected in each generation from the whole population as parents. Here both crossover and mutation operators are used to modify the parent individuals. In replacement phase offspring individuals will replace some or all of parent individuals. If elite strategy is used, then the best individual in the generation will always be included in the next generation.

2.4 Advantages of EAs

EAs are preferred as global optimization techniques from traditional search techniques for the following reasons [6]:

1. EAs don't require prior knowledge of the problem in order to carry out the search. Instead of using previously known domain-specific information to guide the search they make random changes to their candidate solutions and then use the fitness function to determine whether those changes produce an improvement.
2. EAs use stochastic instead of deterministic operators and appear to be robust in problems where the fitness function is complex, discontinuous, noisy, time varying, or has many local optima.
3. EAs operate on multiple solutions simultaneously, gathering information from a population of search points to direct subsequent search effort. This will make EAs less susceptible to the problems of local maxima and noise. Most algorithms can only explore the solution space to a problem in only one direction at a time; and if the solution they discover is suboptimal the search should be done again.
4. Due to their parallelism, EAs can evaluate many solutions at once. Therefore they are particularly well-suited to solving nonlinear problems where the search space is very large.

Also it is worthwhile to note that the process of problem solving is usually preceded by problem modeling. And in most real world problems, the model must be simplified to allow classical methods to be applied. For example, most nonlinear problems are often approximated by linear cost functions since, in a general case, no algorithm will guarantee global solution for nonlinear cost functions. Therefore instead of using an approximate or simplified model of a real problem and then finding its precise solution often it is better to use an exact model of the problem and find its approximate solution using EAs. However the price to pay when using EAs is twofold. First because of their stochastic nature, EAs can not guarantee finding the optimal solution in every run. And second the computational cost associated with EAs is generally very high, and a large number of function evaluations must be performed for a satisfying result to be found. Therefore it is usually advised not to use EAs whenever some quality deterministic optimization method is applicable.

CHAPTER III

LITERATURE REVIEW

3.1 Introduction

Due to their success as global optimization techniques, EAs have recently attracted the attentions from different researchers for solving constrained optimization problems [8], [10], [12], [26]. When dealing with constrained optimization problems with EA, individuals that satisfy all of the constraints involved are called *feasible individuals.* On the other hand, individuals that do not satisfy at least one of the constraints are called *infeasible individuals*. Figure 3.1 shows a group of feasible and infeasible individuals for in a two dimensional search space.

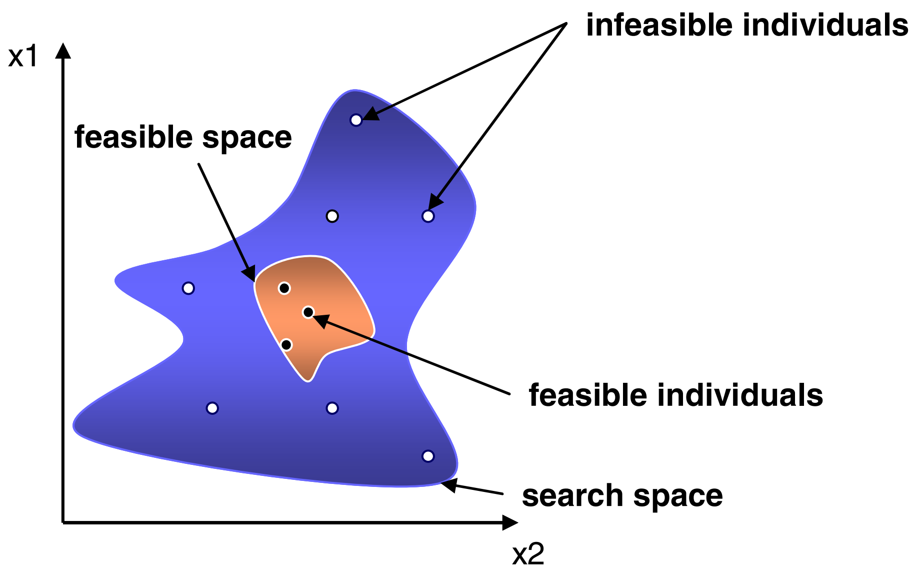


Figure 3.1 Feasible and infeasible individuals

Since EAs are developed as unconstrained search techniques, when they are used to solve constrained problems, an additional mechanism is required to be incorporated into the fitness function evaluation in order to guide the search direction properly. Hence, a variety of approaches have been proposed to achieve this objective. Most of the approaches address one of the major issues of constrained optimization: how to deal with 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 EAs are stochastic search methods and some of the information contained in the infeasible individuals could be unutilized. Moreover if the search space is discontinuous, the EA can also be trapped in local minima. Therefore most constraint optimization techniques are designed to exploit the information contained in infeasible individuals. In this chapter some of the related works in constrained optimization will be reviewed. In doing so, the basic ideas in each paper will be presented. In addition some of the weaknesses that should be improved will be pointed out.

3.2 Classification of Constraint Handling Algorithms Using EA

Different researchers have proposed different ways of classifying constraint handling techniques and there is no generally accepted classification at present. In [8] Coello categorized constrained optimization algorithms into the following five broad groups:

1) Penalty Functions,
2) Special Representations and Operators,
3) Repair Algorithms,
4) Separations of Objectives and Constraints, and
5) Hybrid Methods

Out of the five categories, methods based on special representations and operators [29], repair algorithms [27], and hybrid methods [20] are irrelevant to the algorithm proposed here. Special representations and operators are used to tackle certain particularly difficult problems for which a generic representation used in EAs might not be appropriate. These techniques preserve feasibility at all times and use decoders that transform the shape of the search space. Repair algorithms are normally used in combinatorial optimization problems in which the traditional genetic operators tend to generate infeasible solutions all the time. Repair refers to making infeasible individual feasible through the application of heuristic procedures. Hybrid methods are hybrids with other techniques like Lagrangian multipliers or fuzzy logic.

In [38] Takahama and Sakai classified constrained optimization algorithms into four groups:

1) Penalty functions,
2) Methods based on preference of feasible solutions over infeasible solutions,
3) Methods that use constraint violations and objective functions separately and,
4) Methods based on multiobjective optimization techniques

This is a better way of classifying constraint handling techniques because all of the categories can be implemented using standard EA. Therefore this way of classification will be employed and in the following the characteristics of each category will be presented in detail along with some specific examples.

3.3 Penalty Functions

Penalty functions are the simplest and the most commonly used methods for handling constraints using EAs. They were popularly used in the conventional methods for constrained optimization [16] and were the first methods used to handle constraints with evolutionary algorithms [41]. In these techniques a constrained optimization problem is transformed into an unconstrained one by adding a *penalty* value to the fitness of infeasible individuals so that they will be penalized for violating the constraints. This is intended to make sure that an infeasible individual with a certain objective function value is less likely to be selected for reproduction than a feasible individual having similar objective function value.

In classical optimization, two kinds of penalty functions are considered: *interior* and *exterior* [8]. In interior penalty, the penalty value will have small value at points away from the constraint boundaries and will be very large as the constraint boundaries are approached. Therefore if we start from a feasible point, points generated afterwards will always lie inside the feasible region since the constraint boundaries act as barriers during the optimization process. Interior penalty has the following general form [23]:

$$F(\vec{x}) = f(\vec{x}) + \frac{1}{k}(B(\vec{x})) \tag{3.1}$$

where k is a penalty coefficient and $B(\vec{x})$ is barrier function.

In exterior penalties, one will start with an infeasible solution and then move towards the feasible region. The most commonly used penalty approach in EA is the exterior penalty. It is usually preferred because there is no requirement of an initial feasible solution. The general formulation of the exterior penalty function is [4]:

$$F(\vec{x}) = f(\vec{x}) + \left[\sum_{i=1}^{n} r_i c_j(\vec{x})\right] \tag{3.2}$$

where $F(\vec{x})$ is the new objective function to be optimized, r_i are positive constants (i.e. penalty factors), and $c_j(\vec{x})$ are functions of the inequality and equality constraints which are given as follows:

$$c_j(\vec{x}) = \begin{cases} \max\left[0, g_j(\vec{x})\right]^{\beta} & j = 1, \cdots, k \\ \max |h_j(\vec{x})|^{\gamma} & j = k+1, \cdots, m \end{cases} \tag{3.3}$$

where k is the total number of inequality constraints, m is the total number of constraints, and β and γ are normally 1 or 2.

Penalty functions can also be classified based on the way the penalties are added. In the earliest penalty function method, the *death penalty*, individuals that violate any one of the constraints are completely rejected. In this method no information is extracted from infeasible individuals to guide the search. 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. The general mathematical expression for static penalty functions is similar to the one shown in Equation (3.2).

If, alternatively, the current generation number is considered in determining the penalties, then the method is called *dynamic penalty* function method [19]. Dynamic penalty functions have the following general form:

$$F(\vec{x}) = f(\vec{x}) + (C \times t)^{\alpha} \times \left[\sum_{i=1}^{n} r_i c_j(\vec{x}) \right] \tag{3.4}$$

where C is a constant, t is the generation number and α is usually 2.

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 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 heuristically as in the case of static penalty methods.

In [5] Bean and Alouane developed an adaptive penalty function strategy that takes feedback from the search process to define the penalty parameters. The fitness assignment has the following general form:

$$F(\vec{x}) = f(\vec{x}) + \lambda(t) \left[\sum_{i=1}^{k} g_i^2(\vec{x}) + \sum_{j=1}^{m-k} \left| h_j(\vec{x}) \right| \right] \tag{3.5}$$

where the penalty parameter $\lambda(t)$ is updated at every generation t in the following manner:

$$\lambda(t+1) = \begin{cases} (\frac{1}{\beta_1})\lambda(t) & \text{if } b^i \in \text{(feasible region) for all } t-g+1 \le i \le t \\ \beta_2 \lambda(t) & \text{if } b^i \notin \text{(feasible region) for all } t-g+1 \le i \le t \\ \lambda(t) & \text{otherwise} \end{cases} \tag{3.6}$$

where b^i is the best element at generation i, $\beta_1 \ne \beta_2$ and $\beta_1, \beta_2 > 1$.

Farmani and Wright [14] also proposed an adaptive penalty function strategy that takes feedback from the search process. 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 [23] proposed a simple adaptive penalty function that 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 process is used to define the penalty parameters. Their fitness formulation has the following form:

$$F(\vec{x}) = \begin{cases} f(\vec{x}) & \text{if } x \text{ is feasible} \\ f(\vec{x}) + \sum_{j=1}^{m} k_j c_j(\vec{x}) & \text{otherwise} \end{cases} \tag{3.7}$$

The penalty parameter k_j is given by:

$$k_j = \left| \langle f(\vec{x}) \rangle \right| \frac{\langle c_j(\vec{x}) \rangle}{\sum_{l=1}^{m} \left[\langle c_l(\vec{x}) \rangle \right]^2} \tag{3.8}$$

where $\langle f(\vec{x}) \rangle$ is the average of the objective function values in the current population and $\langle c_l(\vec{x}) \rangle$ is the violation of the l^{th} constraint averaged over the current population.

3.4 Methods based on Preference of Feasible over Infeasible Solutions

In methods based on preference of feasible solutions over infeasible ones, feasible solutions are always considered better than infeasible ones. If a feasible individual is compared with an infeasible individual, the feasible individual will always be preferred for reproduction regardless of their objective function values.

In [30], the authors suggested a technique 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 alone. This method has the following characteristics: 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 main drawback 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 EA being stuck in a local optimum.

In [12], 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 allow equal importance to each constraint. This method performed very well on a variety of benchmark test problems. However, it requires heuristic choices of some parameters, such as the critical distance $\overline{d}$ and n_f.

In [25] a differential evolution (DE) based approach is used for solving constraint optimization problems. In order to increase the probability of a parent generating better offspring, each solution is allowed to generate more than one offspring. In order to select the best individual out of the generated offspring, three selection criteria based on feasibility are used. Between two feasible solutions, the one with the highest fitness value wins. If one solution is feasible and the other one is infeasible, the feasible solution wins. And if both solutions are infeasible, the one with the lowest sum of constraint violations is preferred. In addition a diversity mechanism is used to maintain infeasible individuals located in promising areas of the search space. But a user defined parameter, S_r, is required to determine the probability to select between parent and offspring based only on the objection function value. In addition the algorithm struggled in problems having higher dimensionality and higher number of nonlinear equality constraints.

3.5 Methods that Use Constraint Violations and Objective Functions Separately

In the third group of methods, constraint violation and objective function are optimized separately. These methods adopt a lexicographic order, in which the constraint violation usually precedes the objective function.

In [32], 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 [36] Takahama and Sakai proposed constrained optimization technique by applying the α constrained method to the nonlinear simplex method. The α constrained method is a transformation technique where satisfaction level for constraints is introduced which indicates how well a search point satisfied the constraints. Moreover α level comparison is used to compare individuals based on their satisfaction level of the constraints. The α constrained method can convert an algorithm for unconstrained problems into an algorithm for constrained problems by replacing ordinary comparisons with the α level comparison.

In [37] the same authors proposed the εDE algorithm by applying the ε constrained method to DE, which is similar to the α constrained algorithm. In this method, the ε

level comparison is used to compare individuals based on their constraint violation and objective function values. Based on the value of ε, the comparison switches from the case where constraint violation precedes objective function ($\varepsilon = 0$) to the case where objective function precedes constraint violation ($\varepsilon = \infty$). In addition a gradient-based mutation that finds feasible point using the gradient of constraints at an infeasible point is used. Although the algorithm is shown to be robust to multi modal problems and effective for problems with many equality constraints, controlling the value of ε requires extra computation and parameter definition.

3.6 Methods based on Multiobjective Optimization Techniques

Some researchers have also used multiobjective optimization techniques to solve constrained optimization problems. The main idea here is to convert the single objective optimization problem into a multiobjective optimization problem by treating the constraints as one or more objectives to be minimized. Afterward, any multiobjective optimization technique (e.g. Multiobjective Evolutionary Algorithm (MOEA)) can be employed to solve the problem. A multiobjective optimization technique aims to find a set of trade-off solutions which are considered good in all the objectives to be optimized. But in constrained optimization, feasibility of solutions should take precedence than optimality. Therefore some changes are usually done to the original multiobjective optimization techniques in order to adapt them for solving constrained optimization problems.

In [7] Coello proposed a sub-population based approach similar to Vector Evaluated Genetic Algorithm (VEGA) [34] to treat each constraint as objective. In each generation the population is divided into $m+1$ sub-populations with equal size. m is the total number of constraints and the first sub-population is devoted to optimizing the objective function. In this approach, initially the fitness function for each sub-population (except for the first one) depends on the violation of its constraint. If the solution evaluated does not violate the constraint related to the sub-population but is otherwise infeasible, then the sub-population will minimize the total number of violations. Once the solution becomes feasible it will be combined with the first sub-population and will be used to minimize the objective function. The major drawback of this approach is determining the size of each sub-population.

In [9] the authors proposed a version of the Niched Pareto Genetic Algorithm (NPGA) [18]. To control the diversity of the population, this approach uses a parameter called selection ratio (S_r). This parameter corresponds to the minimum number of individuals that are selected through dominance based tournament selection. The remaining ($S_r - 1$) individuals are selected probabilistically. Four comparison rules are used for the dominance based tournament selection: 1) if two individuals are feasible, the individual with better fitness will be preferred, 2) any feasible solution is considered better fit than any infeasible individual, 3) if two individuals are infeasible, the nondominated individual is preferred only if the other individual is dominated; and 4) if two individuals are infeasible and both are either dominated or nondominated, the individual with the lowest amount of constraint violation will be considered better fit. The main drawback of this approach is that it is difficult to maintain a reasonable proportion of infeasible and feasible solutions in the population.

In [1] the authors proposed the Inverted-shrinkable Pareto Archived Evolutionary Strategy (IS-PAES) which is an extension of the PAES [21]. In this approach the search space is shrunk during the search process so that the search space will be focused onto specific areas of the feasible region. Moreover the size of the search space will be very small and the solutions obtained will be competitive in the end. In addition, an adaptive grid is used to store the solutions found. The main problem with this technique is the difficulty associated with its implementation. In addition, if the search space is shrunk towards a false direction, there is a possibility that the algorithm will be trapped in a local optimum.

In [39], a multiobjective 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 nondominated individual from the offspring population. By using infeasible solutions archiving and replacement mechanism, the best infeasible individual is kept in an archive and then reintroduced 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 [41], the authors suggested a two-phase algorithm that is based on multiobjective 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 [2], 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 involving equality constraints.

3.7 Summary

From the literature review, we can observe that each type of algorithm possesses certain advantages and disadvantages. Dynamic penalty functions, static penalty functions, and methods that use constraint violations and objective functions separately are easy to implement, but usually require some kinds 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. Multiobjective optimization based algorithms require no specific parameter tuning but they are often complex and computationally expensive. The algorithm presented here aims to preserve the merits of the above algorithms while addressing some concerning issues to the drawbacks observed. An ideal constrained optimization algorithm should be *easy to implement*, *free of parameter tuning*, and *guarantee to find good solutions for every problem at every run*. The proposed algorithm, which is based on adaptive penalty function, meets all three goals for solving constrained optimization problems efficiently and effectively.

CHAPTER IV

THE PROPOSED ALGORITHM

4.1 Introduction

One of the major distinctions among constraint handling algorithms is their choice of the infeasible individuals to be involved in the search process. The main purpose of involving infeasible individuals in the search process of constrained optimization is to exploit the information they might carry. There are two kinds of information that an infeasible individual might carry: information about the feasible region and information about the optimal solution. Since EAs are stochastic search techniques, omitting infeasible individuals might lead to the EA being stuck in 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.

Figure 4.1 shows the types of feasible and infeasible individuals one might encounter during the search process. The figure shows a two-dimensional space with the y-axis being the original fitness value of the individual in a population and the x-axis being the sum of all constraint violations (to be defined in Equation (4.4)). There are six individuals shown in the figure. In a typical evolutionary search process there will be more than six but for explanation purpose only six individuals are used. Individual “A” has very low fitness value but high constraint violation, while individual “D” has very low constraint violation but high fitness value. On the other hand, individuals “B” and “C” have relatively low value of both fitness and constraint violation. Individuals “E” and “F” are feasible individuals and have zero constraint violation with E having the lower fitness.

Using these six individuals first the differences between infeasible individual selection strategies of different constrained optimization algorithms will be explained.

Some algorithms prefer individuals such as “D” with low constraint violation [36] only. These types of algorithms usually have a certain threshold value to compare the constraint violation of an infeasible individual. If the individual has constraint violation less than the threshold value, then it can be involved in the reproduction process (i.e. crossover and mutation). Otherwise the individual can not be involved in reproduction. Other algorithms favor individuals such as “A” with low fitness value [39] only. Here an infeasible individual with low fitness value can compete with feasible individuals without considering its constraint violation value. On the other hand in [41] individuals like “D” are preferred only at the beginning of the search process and individuals like “A” are preferred only at the later stages.

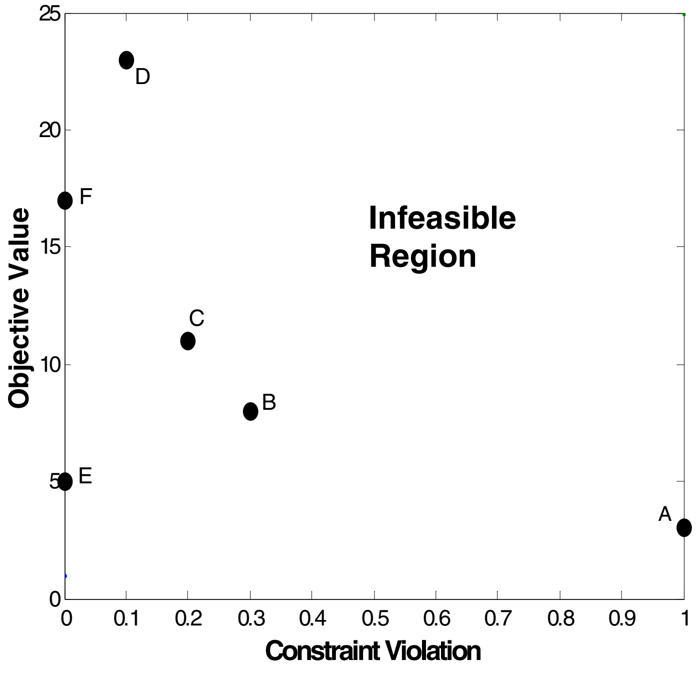


Figure 4.1 Possible combinations of feasible and infeasible individuals in a population

The algorithm presented here is based on the following idea: 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, 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 constraint violation.

4.2 Adaptive Penalty Functions

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 an infeasible individual’s probability of being selected for recombination. Generally static penalty functions have the following form:

$$F(\vec{x}) = f(\vec{x}) + \sum_{j=1}^{m} r_j c_j(\vec{x}) \tag{4.1}$$

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

Although very simple to implement, penalty function approach has its own drawback. The major drawback of penalty function approach is determining the penalty coefficients. Usually, a prior knowledge about the problem is needed or repeated experiment should be done to determine the proper coefficients. Adaptive penalty functions are designed to solve the need of determining penalty coefficients. In these methods the penalty coefficients will be determined adaptively from information gathered from the search process.

In this research, a self-adaptive penalty function method is proposed. In the method two penalty values are added to infeasible individuals in order to achieve the two main goals of constrained optimization algorithm: finding feasible solutions (if they aren't any available) and then searching for the optimum solution. The two penalties are designed so that if there is no feasible individual present in the population, highly infeasible individuals are more penalized and if there are few infeasible individuals; those with high objective function are more penalized. In order to determine the amount of penalty added to infeasible individuals, the algorithm keeps track of the number of feasible individuals in the population in each generation. Moreover, a new fitness value called "distance" is introduced and the sum of this value and the penalty added will determine the rank of each individual. In the next two sections the formulation of the distance value and the two penalties will be explained in detail. In the end the derivation of the final fitness according to which individuals will be ranked will be thoroughly discussed.

4.3 Distance Value

The first part of the algorithm involves assigning the distance value to all of the individuals in the population. The distance value has two major purposes. First if there are no feasible individuals in the population at the current generation, then the main focus of the search should be finding feasible individuals. Hence, in this case, the distance value is formulated so that infeasible individuals with low constraint violation are better fit. On the other hand if there are feasible individuals in the current population, then the search should be directed to finding the optimal solution. The infeasible individuals that carry good information about the optimal solution are the ones that have both low constraint violation as well as low fitness value. Therefore, in this case distance value is formulated so that infeasible individuals with low constraint violation and low fitness value will be better fit. Feasible individual, on the other hand, will have distance value equal to their original fitness value.

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

$$f_{\min} = \min_x f(\vec{x}), \text{ and } f_{\max} = \max_x f(\vec{x}). \tag{4.2}$$

Then each individual's fitness value will be normalized by,

$$\tilde{f}(\vec{x}) = \frac{f(\vec{x}) - f_{\min}}{f_{\max} - f_{\min}}, \tag{4.3}$$

where $\tilde{f}(x)$ is the normalized fitness value.

After the above transformations, each individual's fitness value will lie between 0 and 1 with 0 corresponding to the individual with the smallest fitness value and 1 to the individual with the highest fitness.

Constraint violation, $v(\vec{x})$, of each infeasible individual is then calculated as the sum of the normalized violation of each constraint divided by the total number of constraints,

$$v(\vec{x}) = \frac{1}{m}\sum_{j=1}^{m} \frac{c_j(\vec{x})}{c\max_j}, \tag{4.4}$$

where

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

$c\max_j = \max_x c_j(\vec{x})$ and

δ is tolerance value (usually 0.001 or 0.0001) .

Then the "distance" value, $d(\vec{x})$, is formulated as follows:

$$d(\vec{x}) = \begin{cases} v(\vec{x}), & \text{if } r_f = 0 \\ \sqrt{\tilde{f}(\vec{x})^2 + v(\vec{x})^2}, & \text{otherwise} \end{cases} \tag{4.5}$$

where

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

Figure 4.2 shows the distance values of the individuals in Figure 4.1. From Equation (4.5), 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 4.2, if individuals "E" and "F" were not present, then individual "D" would have the smallest distance value.

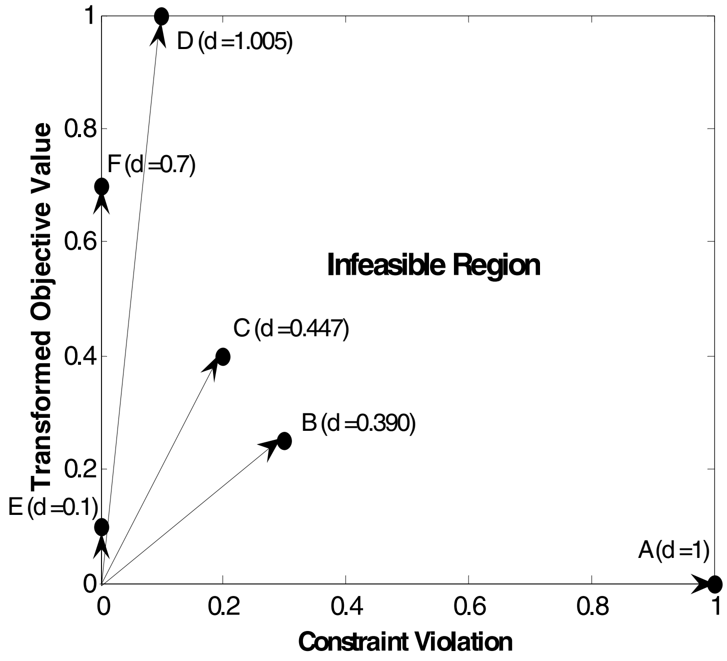


Figure 4.2 Distance value of individuals in Figure 4.1

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

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

2. For infeasible individuals, the distance value is the measure of the objective function value and the constraint violation. As can be seen from the previous figure, individuals near the origin (in the $\tilde{f}(\vec{x}) - v(\vec{x})$ space) would have lower distance value than those farther away from the origin. Therefore, if we compare two infeasible individuals based on their distance value, then the one that has both low objective function value and low violation will be considered a 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 function value, the feasible individual will have smaller distance value.

```
Input: $f(\vec{x})_i$, $v(\vec{x})_i$ $\forall i, i = 1, \ldots, Population\ Size$,
       $f_{\min}$, $f_{\max}$, $r_f$
Output: $d(\vec{x})_i$ $\forall i, i = 1, \ldots, Population\ Size$

Begin
    If $r_f = 0$ then
        For $i = 0$ to $Population\ Size$ Do
            $d(\vec{x})_i \leftarrow v(\vec{x})_i$
        End For
    Else
        For $i = 0$ to $Population\ Size$ Do
            $\tilde{f}(\vec{x})_i \leftarrow \frac{f(\vec{x})_i - f_{\min}}{f_{\max} - f_{\min}}$
            $d(\vec{x})_i \leftarrow$ Sqrt ( $\tilde{f}(\vec{x})_i^2 + v(\vec{x})_i^2$ )
        End For
    End If
End
```

Figure 4.3 Pseudocode for finding distance value

The pseudocode for finding the distance value of individuals is shown in Figure 4.3. The algorithm takes as input the objective function values and the constraint violation values of each individual in the population, the minimum and maximum values of the objective function, and the feasibility ratio of the current population. The value of feasibility ratio is first checked. If it is zero, then the distance value of each individual will be equal to its constraint violation value. Otherwise, the distance value will be assigned as in Equation (4.5).

4.4 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. Adding these penalties have two major aims: 1) to further decrease the fitness of infeasible individuals as the penalty imposed upon infeasible individuals by the distance formulation is very small, and 2) to identify the best infeasible individuals in the population by adding different amount of penalty to each infeasible individual's fitness.

The number of feasible individuals in the population is used to determine the amount of penalty added to an infeasible individual. If there are few feasible individuals in the population we would want infeasible individuals with low constraint violation to be less penalized than those with high constraint violation. And if there are many feasible individuals in the population we would want infeasible individuals with better objective function value to be less penalized.

The two penalties are formulated accordingly as follows:

$$p(\vec{x}) = (1 - r_f)X(\vec{x}) + r_f Y(\vec{x}), \tag{4.6}$$

where

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

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

From the penalty function definition in Equation (4.6), we can observe that if the feasibility ratio (r_f) of the population is small (but not zero), then the first penalty ($X(x)$) will have more impact than the second penalty ($Y(x)$). The first penalty is formulated to have large value for individuals with large amount of constraint violation. Hence in the case when there are few feasible individuals present in the population (r_f is small), infeasible individuals with high constraint violation will be more penalized than those with low constraint violation. On the other hand, if there are many feasible solutions in the population (r_f is large), the second penalty will have more effect than the first one. In this case infeasible individuals with large objective function value will be more penalized than infeasible individuals with small objective function value. If there are no feasible individuals in the population ($r_f = 0$), both penalties will be zero.

```
Input: $f(\vec{x})_i$, $v(\vec{x})_i$ $\forall i, i = 1,\ldots,$ Population Size, $r_f$
Output: $p(\vec{x})_i$ $\forall i, i = 1,\ldots,$ Population Size

Begin
    For $i = 0$ to Population Size Do
        If $r_f = 0$ then
            $X(\vec{x})_i \leftarrow 0$
        Else
            $X(\vec{x})_i \leftarrow v(\vec{x})_i$
        End If
        If $v(\vec{x})_i = 0$ then
            $Y(\vec{x})_i \leftarrow 0$
        Else
            $Y(\vec{x})_i \leftarrow \tilde{f}(\vec{x})_i$
        End If
        $p(\vec{x})_i \leftarrow (1 - r_f)X(\vec{x})_i + r_f Y(\vec{x})_i$
    End For
End
```

Figure 4.4 Pseudocode for finding penalty value

The pseudocode for finding the penalty is shown in Figure 4.4. The objective function value and the constraint violation value of each individual in the population are inputs to the algorithm. The algorithm will iterate through each individual first by checking whether the feasibility ratio of the current population is zero. If that is the case, then the value of the first penalty will be set to zero for that specific individual, otherwise it will be equal to the individual's constraint violation. Next the constraint violation of the individual will be checked. If it is zero, then the second penalty will be zero. Otherwise the second penalty will be assigned according to Equation (4.6).

4.5 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,

$$F(x) = d(x) + p(x) \,. \tag{4.7}$$

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 only or those individuals with low fitness value only. But according to the new fitness formulation, the infeasible individuals that are considered valuable are not always similar. Instead they vary based on the current situation of the search process. 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 alone. This will help us to find 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 the 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.

Furthermore the best feasible individual in each generation is archived as *elite solution*. In the case where there is no feasible individual available, the infeasible individual with the least constraint violation will be archived until a feasible individual is found. The archived individual will be allowed to participate in the recombination process in each generation until it is replaced by a better fit individual. This will allow the algorithm to preserve the already found good solution until a better solution is found so that it will not be lost during the search.

4.6 Overall Algorithm

In Figure 4.5 the flow chart of the overall algorithm is shown while Figure 4.6 shows the corresponding pseudocode. The algorithm has two parts: *constraint handling* algorithm and a *search algorithm*. Every constraint optimization algorithm consists of these two separate components. The constraint handling algorithm is used for handling constraints in problems having one or more constraints. The search algorithm, on the other hand, can be any algorithm that is used for solving unconstrained optimization problems.

The algorithm begins by randomly generating the initial population of size *Population_Size*. Then in each generation (G) the constraint handling algorithm and the search algorithm will be executed until the termination condition (i.e. *Maximum*

Generation) is satisfied. The constraint handling algorithm begins by evaluating the objective function ($f(\vec{x})_i^G$) and the constraint violation ($v(\vec{x})_i^G$) of individuals in the population. Based on the constraint violation of individuals in the current population, the feasibility ratio ($r_f^{\,G}$) will be calculated next. If the feasibility ratio is not zero (i.e. if there are some feasible individuals in the current population), the minimum and maximum values of the objective function (i.e. $f_{\min}^{\,G}$ & $f_{\max}^{\,G}$ respectively) will be calculated. Then the distance ($d(\vec{x})_i^G$) and penalty ($p(\vec{x})_i^G$) values of individual i will be evaluated and will be used for determining the fitness value ($F(\vec{x})_i^G$) of each individual. The first part of the algorithm (i.e. the constraint handling algorithm) stops here after assigning a fitness value to each individual. This fitness value contains information about the objective function value as well as the constraint violation of the individual.

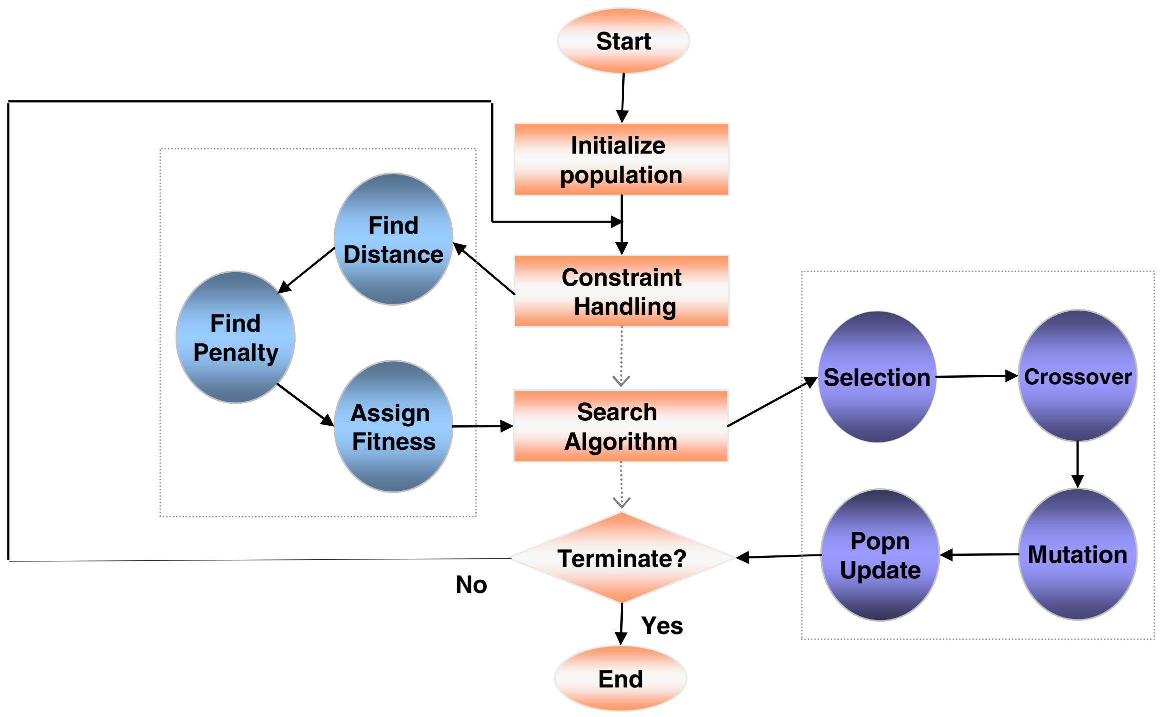


Figure 4.5 Flow chart of proposed algorithm

The second part of the algorithm part is the search process. The focus of the research has been on designing the constraint handling algorithm which can be used with any search engine (i.e. GA based or DE based) for solving constrained optimization problems. Therefore in Figure 4.5, a typical GA based search engine is adopted. The algorithm has four parts: *selection*, *crossover*, *mutation* and *population update*. Any one of the commonly used selection mechanisms (i.e. roulette wheel selection, tournament selection or rank based selection) can be used. There are different techniques of performing crossover (e.g. BLX-α and SPX) and mutation (e.g. uniform mutation and Gaussian mutation) that can be employed. A little modification has been made on the population update part where the best solution in the current population will always be kept as an elite solution. This will make sure that the information about the best location reached (either towards the feasible region or towards the optimal solution) so far will not be lost during the search.

```
Procedure for the Proposed Algorithm

Begin
    Randomly generate initial population $\vec{x}_i \ \forall i, i = 1,\ldots, Population\ Size$
    For $G = 1$ to Maximum Generation Do
        // Part 1: Constraint Handling
        For $i = 0$ to Population Size Do
            Evaluate $f(\vec{x})_i^G$
            Evaluate $v(\vec{x})_i^G$
        End For
        Find $r_f^G$
        If $r_f^G \neq 0$ then
            $f_{\min}^G \leftarrow \min(f(\vec{x})^G)$
            $f_{\max}^G \leftarrow \max(f(\vec{x})^G)$
        End If
        Evaluate $d(\vec{x})_i^G \ \forall i, i = 1,\ldots, Population\ Size$
        Evaluate $p(\vec{x})_i^G \ \forall i, i = 1,\ldots, Population\ Size$
        For $i = 0$ to Population Size Do
            $F(\vec{x})_i^G \leftarrow d(\vec{x})_i^G + p(\vec{x})_i^G$
        End For
        // Part 2: Search Algorithm
        Perform selection
        For $i = 0$ to Population Size Do
            Perform crossover
            Perform mutation
        End For
        Keep Elite solution
        Update population
        $G \leftarrow G + 1$
    End For
End
```

Figure 4.6 Pseudocode of proposed algorithm

CHAPTER V

EXPERIMENTAL RESULTS AND DISCUSSION

5.1 Benchmark Functions

The proposed algorithm was tested on 22 test functions. The detailed formulation of the problems [24] can be found in Appendix A. These test functions are extensions to the commonly used 11 test functions in [22]. Some of the characteristics of the benchmark functions are summarized in Table 5.1. As can be seen from the table, these functions represent a diverse set of functions that will help to evaluate the performance of different constraint handling algorithms. They involve linear, nonlinear, quadratic, cubic and polynomial problems. In addition the number of decision variables, which is shown in the second column, varies for each problem. The numbers of constraints as well as their types (i.e. linear inequality (LI), nonlinear inequality (NI), linear equality (LE), and nonlinear equality (NE)) are shown in columns 5, 6, 7 and 8. The number of the inequality constraints that are active is shown in the last column. Each test function has a different feasibility ratio "ρ" which is determined experimentally by calculating the percentage of feasible solutions among 1,000,000 randomly generated individuals [19]. It is an estimate of the ratio of the feasible space to that of the entire search space. This is shown in the fourth column. Most of the test functions have feasibility ratio less than 1% and finding feasible solutions in these functions is challenging.

5.2 Algorithm Implementation

The algorithm proposed is applied to the 22 benchmark problems using a *real coded GA* as the search algorithm. In real coded GA, optimization is performed in real-valued search spaces. This approach allows for greater precision and complexity than the comparatively restricted method of using binary numbers only and often “is intuitively closer to the problem space” [15].

TABLE 5.1

Summary of main characteristics of the benchmark problems

Prob.	n	Type of function	ρ	LI	NI	LE	NE	A
g01	13	Quadratic	0.0111%	9	0	0	0	6
g02	20	Nonlinear	99.9971%	1	1	0	0	1
g03	10	Nonlinear	0.0000%	0	0	0	1	1
g04	5	Quadratic	52.1230%	0	6	0	0	2
g05	4	Nonlinear	0.0000%	2	0	0	3	3
g06	2	Nonlinear	0.0066%	0	2	0	0	2
g07	10	Quadratic	0.0003%	3	5	0	0	6
g08	2	Nonlinear	0.8560%	0	2	0	0	0
g09	7	Nonlinear	0.5121%	0	4	0	0	2
g10	8	Linear	0.0010%	3	3	0	0	3
g11	2	Quadratic	0.0000%	0	0	0	1	1
g12	3	Quadratic	4.7713%	0	1	0	0	0
g13	5	Nonlinear	0.0000%	0	0	0	3	3
g14	10	Nonlinear	0.0000%	0	0	3	0	3
g15	3	Quadratic	0.0000%	0	0	1	1	2
g16	5	Nonlinear	0.0204%	4	34	0	0	4
g17	6	Nonlinear	0.0000%	0	0	0	4	4
g18	9	Quadratic	0.0000%	0	13	0	0	6
g19	15	Nonlinear	33.4761%	0	5	0	0	0
g21	7	Linear	0.0000%	0	1	0	5	6
g23	9	Linear	0.0000%	0	2	3	1	6
g24	2	Linear	79.6556%	0	2	0	0	2

In all of the problems *linear rank based selection* [3], [4] was used as the selection strategy. In this selection method the individuals in the population will be ranked according to their fitness value. The probability that an individual will be selected is proportional to its rank in this sorted list. The advantage of this method is that it can prevent very fit individuals from gaining dominance in early generations at the expense of less fit individuals which would reduce the population's genetic diversity.

Crossover is implemented using the *blend crossover* (BLX-α) operator which was discussed in section 2.2. In BLX-α, offspring are generated in two steps as follows:

(3) Choose two parent vectors $\vec{x}^1$ and $\vec{x}^2$ randomly from the population,

(4) Generate the i^{th} element x_i^o of the offspring vector $\vec{x}^o$ randomly from the interval $[X_i^1, X_i^2]$ given as follows:

$$\begin{aligned} X_i^1 &= \min(x_i^1, x_i^2) - \alpha d_i \\ X_i^2 &= \min(x_i^1, x_i^2) + \alpha d_i \\ d_i &= \left| x_i^1 - x_i^2 \right| \end{aligned} \tag{5.1}$$

where α is a positive number between 0 and 1. The BLX-α crossover has an advantage in generating diverse offspring as well as being simple to implement.

As in [37] three mutation operators- *boundary mutation*, *uniform mutation* and *Gaussian mutation*- are adopted. In boundary mutation when a gene x_i of a vector $\vec{x}$ is mutated, it is replaced by either $l(x_i')$ or $u(x_i')$ with equal probability, where $l(x_i')$ and $u(x_i')$ are defined as follows:

$$\begin{aligned} l(x_i') &= \min(x_i') \\ u(x_i') &= \max(x_i') \end{aligned} \tag{5.2}$$

where x_i' is a feasible solution. Boundary is an effective mutation operation in constrained optimization as optimal solutions often exist in the neighborhood of the boundary of the feasible region.

In uniform mutation, a gene x_i is replaced by a uniform random number from the interval $[l(x_i), u(x_i)]$, where $l(x_i)$ and $u(x_i)$ are defined in Equation (1.4). Uniform mutation produces values which are not included in the initial population and is important to use this mutation operator when the diversity of the initial population is low.

Unlike boundary mutation and uniform mutation which are used to search solutions globally, Gaussian mutation is used to search locally. In Gaussian mutation, x_i is mutated to give x_i^m as shown below:

$$x_i^m = x_i + \delta \tag{5.3}$$

where δ_i is a Gaussian random number with the normal distribution $N(0, \beta_G^2 (u(x_i) - l(x_i))^2)$. The standard deviation is proportional to the spread between the upper and the lower bound of x_i (i.e. $(u(x_i) - l(x_i))$) and β_G is a parameter to be defined.

The parameters used for algorithm implementation are given as follows:
Population Size: 100,
Maximum Generation: 5,000,
Maximum Fitness Evaluation (FES): 500,000,
Crossover rate: 0.9,
Alpha value: 0.5-0.8,
Boundary mutation rate: 0.01,
Uniform mutation rate: 0.01,
Gaussian mutation rate: 0.1.
Gaussian mutation parameter β_G: 0.01-0.05

5.3 Test Results

The test results of the algorithm are summarized in Tables 5.2, 5.3, 5.4 5.5 and 5.6 according to the guidelines given in [24]. Each test problem is run for 50 independent trials. For each trial the following procedure is followed:

1) The function error, or the difference between the best value found and the optimal value (i.e. $f(\vec{x})$ - $f(\vec{x}^*)$), after 5×10^3, 5×10^4 and 5×10^5 number of function evaluations (FES) is identified.
2) The function errors for the 50 trials are then compared and the best, the median, the worst, the mean and the standard deviation values are reported in

Tables 5.2 to 5.5. The numbers in parenthesis after the error values correspond to the number of violated constraints at the corresponding values. For example if the value in the parenthesis for the median run is 4, then it implies that the best individual for this run has violated 4 constraints. A value of zero indicates that the run has produced a feasible solution.

TABLE 5.2

Error values achieved when FES =5×10^3, FES =5×10^4 or FES = 5×10^5 for testing functions g01-g06

FES		g01	g02	g03	g04	g05	g06
5×10^3	Best	4.3547(0)	0.0284(0)	0.9961(0)	3.4730(0)	152.8019(0)	530.4634(0)
	Median	4.3547(0)	0.0284(0)	0.9961(1)	3.4730(0)	387.5015(0)	530.4634(0)
	Worst	1.1116(4)	0.0325(0)	0.9968(0)	6.0174(0)	387.5015(0)	557.7571(0)
	$\bar{v}$	0.0000	0.0000	0.0911	0.0000	0.0000	0.0000
	Mean	2.7980	0.0293	0.9962	4.2363	274.8457	531.5551
	Std	1.6202	0.0016	0.0003	1.1659	117.2558	5.3484
5×10^4	Best	0.0309(0)	0.0028(0)	0.2836(0)	0.5178(0)	3.6177(0)	4.5157(0)
	Median	0.6929(0)	0.0172(0)	0.3591(0)	1.1853(0)	4.6911(0)	8.3433(0)
	Worst	1.4087(0)	0.0304(0)	0.8519(0)	2.3214(0)	19.4236(0)	12.1043(0)
	$\bar{v}$	0.000	0.0000	0.0000	0.0000	0.0000	0.0000
	Mean	0.6303	0.0190	0.4350	1.0859	8.9762	7.8102
	Std	0.5555	0.0089	0.2110	0.4962	7.1801	3.1004
5×10^5	Best	-0.0001(0)	0.0000(0)	0.0001(0)	-0.1211 (0)	0.0000(0)	-0.2285(0)
	Median	-0.0001(0)	0.0001(0)	0.0001(0)	-0.1131(0)	0.0000(0)	-0.1790(0)
	Worst	0.0000(0)	0.0001(0)	0.0001(0)	0.0000(0)	3.5431(0)	0.0001(0)
	$\bar{v}$	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Mean	-0.0000	0.0001	0.0001	-0.0106	1.0453	-0.0013
	Std	0.0000	0.0001	0.0000	0.0014	1.4324	0.2321

3) The mean value of constraint violations (i.e. $v(x)$ in Equation (4.4)) for the best solution in the median trial is shown as $\bar{v}$ in Tables 5.2 to 5.5.

4) In Table 5.6 the best, the median, the worst, the mean and the standard deviation of the number of FES to achieve a fixed accuracy level (i.e. ($f(\vec{x})$ - $f(\vec{x}^*) \le 0.0001$) are shown. In addition the *Feasible Rate* (rate of runs where at least one feasible solution is found), the *Success Rate* (rate of runs where the required accuracy is met) and the *Success Performance* (the mean FES of the successful runs multiplied by the total number of runs and divided by the total number of successful runs) are also reported.

TABLE 5.3

Error values achieved when FES = 5×10^3, FES = 5×10^4 or FES = 5×10^5 for testing functions g07-g12

FES		g07	g08	g09	g10	g11	g12
5×10^3	Best	119.2733(0)	0.0020(0)	19.0029(0)	1400.3890(0)	0.0009(0)	0.0001(0)
	Median	896.6370(0)	0.0020(0)	28.9447(0)	1400.3890(0)	0.0011(0)	0.0002(0)
	Worst	896.6370(0)	0.0033(0)	28.9447(0)	1400.3890(0)	0.0011(0)	0.0082(0)
	$\bar{v}$	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Mean	818.9006	0.0025	27.3540	1400.3890	0.0011	0.0038
	Std	233.2091	0.0006	3.6447	0.0000	0.0001	0.0037
5×10^4	Best	0.2653(0)	0.0000(0)	0.1547(0)	490.5240(0)	0.0001(0)	0.0000(0)
	Median	4.9832(0)	0.0001(0)	0.4131(0)	1097.0097(0)	0.0008(0)	0.0001(0)
	Worst	937.1398(0)	0.0004(0)	0.6197(0)	1097.0097(0)	0.0031(0)	0.0003(0)
	$\bar{v}$	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Mean	246.9571	0.0002	0.3956	854.4154	0.0013	0.0001
	Std	379.7854	0.0001	0.1382	297.1160	0.0012	0.0001
5×10^5	Best	0.0000(0)	0.0000(0)	0.0000(0)	0.0000(0)	0.0000(0)	0.0001(0)
	Median	0.0000(0)	0.0000(0)	0.0000(0)	12.4641(0)	0.0000(0)	0.0001(0)
	Worst	0.0001(0)	0.0000(0)	0.0000(0)	140.3215(0)	0.0000(0)	0.0001(0)
	$\bar{v}$	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Mean	0.0000	0.0000	0.0000	28.4341	0.0000	0.0001
	Std	0.0000	0.0000	0.0000	51.2404	0.0000	0.0000

TABLE 5.4

Error values achieved when FES =5×10^3, FES =5×10^4 or FES = 5×10^5 for testing functions g13-g18

FES		g13	g14	g15	g16	g17	g18
5×10^3	Best	0.9503(0)	4.3012(0)	0.0327(0)	0.0231(0)	185.7021(1)	0.2903(0)
	Median	0.9503(0)	4.7716(0)	0.0327(0)	0.0231(0)	185.7021(1)	0.2903(0)
	Worst	0.9025(1)	4.7716(0)	0.2531(0)	0.0238(0)	270.7656(4)	0.3026(0)
	$\bar{v}$	0.0000	0.0000	0.0000	0.0000	0.3485	0.0000
	Mean	0.9446	4.6963	0.0415	0.0233	206.9679	0.2962
	Std	0.0155	0.1724	0.0431	0.0001	36.8335	0.0061
5×10^4	Best	0.3458(0)	0.7374(0)	0.0000(0)	0.0011(0)	71.6903(1)	0.0013(0)
	Median	0.8830(0)	2.5243(0)	0.0014(0)	0.0011(0)	256.5883(4)	0.0076(0)
	Worst	0.6228(1)	4.0504(0)	0.0095(0)	0.0033(0)	316.9424(4)	0.2547(0)
	$\bar{v}$	0.0000	0.0000	0.0000	0.0000	0.0441	0.0000
	Mean	2.6931	2.3165	0.0024	0.0021	171.6836	0.0456
	Std	4.0419	0.9680	0.0031	0.0001	101.6865	0.0935
5×10^5	Best	0.0000(0)	0.0001(0)	0.0000(0)	0.0000(0)	0.0001(0)	0.0000(0)
	Median	0.0001(0)	0.0001(0)	0.0000(0)	0.0000(0)	56.7392(0)	0.0001(0)
	Worst	0.0001(0)	0.0002(0)	0.0000(0)	0.0001(0)	58.8482(0)	0.0001(0)
	$\bar{v}$	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Mean	0.0001	0.0001	0.0000	0.0001	34.9479	0.0001
	Std	0.0000	0.0001	0.0000	0.0000	29.0347	0.0000

The first thing we notice from Tables 5.2, 5.3, 5.4 and 5.5 is that the algorithm was able to find very good feasible solutions for all the 50 runs and for all of the 22 testing functions. In addition, we also notice from the tables that only 5×10^3 FES is sufficient for the algorithm to produce at least one feasible solution except for g17, g21 and g23. For most of the testing functions the feasibility ratio is less than 1% and the algorithm had to start with all infeasible individuals in the initial population. Only g02, g04, g12, g19 and g24 have feasibility ratio greater than 1 percent in which case finding feasible solutions is not difficult. In fact, the initial population in these test functions always consisted of feasible solutions. Although g11 has zero feasibility ratio, relaxing the only available equality constraint led to always finding feasible individuals in the initial population. Actually all equality constraints were relaxed by a threshold value of 0.0001.

TABLE 5.5

Error values achieved when FES =5×10^3, FES =5×10^4 or FES = 5×10^5 for testing functions g19, g21, g23 and g24

FES		g19	g21	g23	g24
5×10^3	Best	24.0635(0)	55.5527(6)	3.9117(5)	0.0781(0)
	Median	24.0635(0)	91.1871(6)	3.9117(5)	0.0781(0)
	Worst	34.6545(0)	91.1871(6)	12.1624(6)	0.0781(0)
	$\bar{v}$	0.0000	0.0923	0.1063	0.0000
	Mean	28.2999	89.7617	4.9018	0.0781
	Std	5.1885	6.9828	2.6811	0.0000
5×10^4	Best	0.3505(0)	7.2581(5)	4.5320(6)	0.0001(0)
	Median	0.8940(0)	71.1181(5)	47.6524(6)	0.0003(0)
	Worst	1.2213(0)	103.6376(5)	78.6462(6)	0.0007(0)
	$\bar{v}$	0.0000	0.0895	0.0876	0.0000
	Mean	0.8975	60.5697	44.4369	0.0004
	Std	0.2139	33.1987	23.9404	0.0002
5×10^5	Best	0.0000(0)	2.9085(0)	0.2927(0)	-0.0000(0)
	Median	0.0000(0)	4.8113(0)	3.4647(0)	0.0000(0)
	Worst	0.0001(0)	11.6892(0)	16.0222(0)	0.0000(0)
	$\bar{v}$	0.0000	0.0000	0.0000	0.0000
	Mean	0.0000	5.7913	5.2924	0.0000
	Std	0.0000	2.3565	3.8656	0.0000

In addition to the feasibility ratio, nonlinear constraints also pose some difficulty on finding feasible solutions. However the flexibility of the fitness formulation allowed the algorithm to search for feasible individuals before proceeding to finding the global optimum value. Whenever there is no feasible individual the main objective of the algorithm will be finding feasible individuals. And hence infeasible individuals that will help to achieve this will be given precedence. Generally, finding feasible individuals in every run is very important because in real world applications infeasible individuals will have no usable value.

TABLE 5.6

Number of FES to achieve the fixed accuracy level $((f(\vec{x}) - f(\vec{x}^*)) \leq 0.0001)$, Success Rate, Feasible Rate and Success Performance

Prob.	Best	Median	Worst	Mean	Std	Feasible Rate	Success Rate	Success Performance
g01	255200	301400	453300	419692.0000	258887.8412	100%	100%	419692.0000
g02	240900	441100	479800	410266.6666	52042.2904	100%	60%	683777.7777
g03	58500	289100	495100	232505.2631	103159.8426	100%	100%	232505.2631
g04	166900	194100	283500	207400.0000	37914.4651	100%	100%	259250.0000
g05	280432	310678	324568	312654.6789	654.6787	100%	55%	568463.0525
g06	100600	124100	178400	128887.5000	22378.6693	100%	100%	201386.7187
g07	60000	176600	259300	215300.0000	81755.2852	100%	100%	215300.0000
g08	2000	46000	163600	57076.0000	43738.2146	100%	100%	57076.0000
g09	18200	34600	291699	53031.8181	63102.6752	100%	100%	53031.8181
g10	163000	190000	266000	203710.4285	34390.9810	100%	70%	291020.0408
g11	5500	209400	408900	217650.0000	118376.8664	100%	100%	217650.0000
g12	17600	78200	195900	108180.0000	66366.6904	100%	100%	108180.0000
g13	48500	135000	110900	77970.0000	51225.0236	100%	70%	111385.7142
g14	89300	158600	214900	150750.0000	44900.5846	100%	64%	235546.8750
g15	7400	10900	43700	18460.0000	13539.3648	100%	100%	18460.0000
g16	45765	48765	55675	48578.6769	189.7678	100%	80%	48578.6769
g17	100700	108900	154500	115800.0000	15936.6802	100%	36%	32166.6666
g18	137600	182100	355500	210033.3333	63437.8435	100%	96%	218784.7221
g19	22300	25400	34800	26100.0000	1598.8714	100%	100%	26100.0000
g21	-	-	-	-	-	100%	-	-
g23	-	-	-	-	-	100%	-	-
g24	26400	157000	459900	196023.5294	139329.4763	100%	100%	196023.5294

Also in the same tables the function error values are reported after 5×10^3, 5×10^4 and 5×10^5 number of FES. From Tables 5.2 and 5.5 we can notice that negative error values are reported for test functions g01, g04, g06 and g24. This is because the algorithm was able to find better solutions than the already reported optimal solutions for these problems. For g01 a new optimal value of -15.000136 is found at x^* = (0.999999, 0.999998, 0.999999, 1.000000, 1.000000, 1.000000, 0.999998, 1.000000, 1.000000, 3.000052, 3.000053, 3.000051, 1.000000). For g04 the new optimal value is -30665.659868 which is at x^* = (78.000000, 33.000001, 29.994777, 45.000000, 36.776315). For g06 the optimal value found is -6962.042447 which is located at x^* = (14.094902, 0.842758). And for g24 a new optimal value of -5.508113 is found at x^* = (2.329520, 3.178593). In addition for g08, g11, g12, g15 and g24 a solution satisfying the required accuracy level (i.e. $(f(\vec{x}) - f(\vec{x}^*) \leq 0.0001)$ was found only using 5×10^4 FES. Actually for g08, g11 and g12 only 5×10^3 FES was enough for finding error value less than 0.002.

The high rate of finding feasible solutions is also shown in Table 5.6 where the feasible rate is 100% for all of the test functions. In addition the success performance in the same table shows that the algorithm achieved 100% success for finding the required accuracy level for 12 of the 22 test problems. For the rest test problems either the low feasibility ratio or the presence of nonlinear constraints prevented the algorithm to reach the desired accuracy level every time. This is because the algorithm has to spend more time in finding feasible solutions than in locating the optimal value. A success rate less than 50% is achieved only for three test functions (g17, g21 and g23) where each test problem has a zero feasibility ratio and one or more nonlinear equality constraints. On the other hand except for g01 and g02, a FES less than 2×10^5 is enough for finding the best individual. For test functions g21 and g23, although the required error value could not be reached repeatedly, the best result found is not very far from the true optimal value as can be seen in Table 5.7.

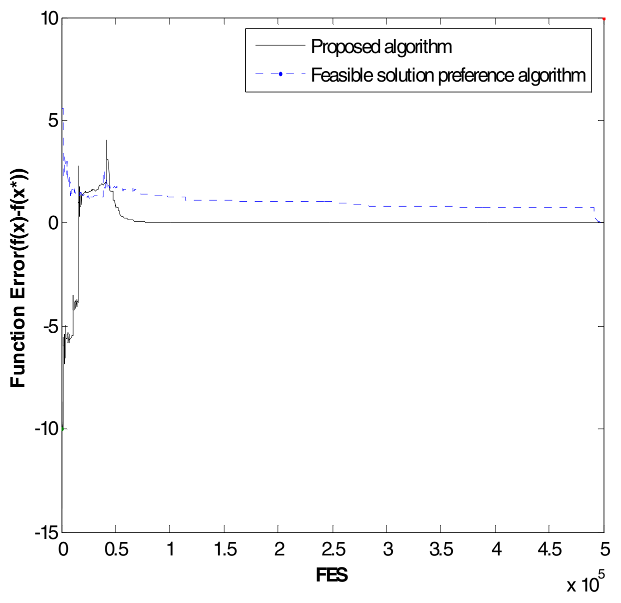
TABLE 5.7

Comparison of best results

Test Function	Optimum value	Proposed Algorithm	Farmani & Wright [14]	Runarsson & Yao [33]	Venkatraman & Yen [41]
g01	-15.000000	**-15.000136**	-15.0000	-15.0000	-14.9999
g02	-0.803619	-0.803601	-0.802970	-0.803619	-0.803190
g03	-1.000500	-1.000493	-1.0000	-1.0001	-1.0000
g04	-30665.538671	**-30665.659868**	-30665.500	-30665.539	-30665.531
g05	5126.496714	5126.496797	5126.989	5126.497	5126.630
g06	-6961.813875	**-6962.042447**	-6961.800	-6961.814	-6961.179
g07	24.306209	24.306217	24.480	24.306	24.411
g08	-0.095825	-0.095825	-0.095825	-0.095825	-0.095825
g09	680.630057	680.630058	680.640	680.630	680.762
g10	7049.248020	7049.248021	7061.340	7049.248	7060.553
g11	0.749900	0.749918	0.75	0.750	0.749
g12	-1.000000	-0.999999	-	-1.000000	-
g13	0.053941	0.053941	-	0.053942	-

5.4 Result Comparison

In Table 5.7 the best results found by the proposed algorithm for test functions g01 to g13 are compared with other related algorithms in the literature. From the table we can observe that the proposed algorithm performed better than the rest by finding the true optimal solution to 0.0001 accuracy level for all of the 13 testing functions. In addition the algorithm has found results better than the optimal results reported so far for test functions g01, g04, and g06 which are shown in bold. Of the other algorithms stochastic ranking [33] (with $\gamma = 0.85$) has produced the better results. However this algorithm needs a parameter (i.e. γ) to be defined by users. On the other hand in [14] only 17 out of 20 runs produced feasible solutions for problem g10. For the same algorithm feasible solutions could be found only for 9 out of 20 runs for g05. But the proposed algorithm produced feasible solutions in all runs for all testing functions. In [41] 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.



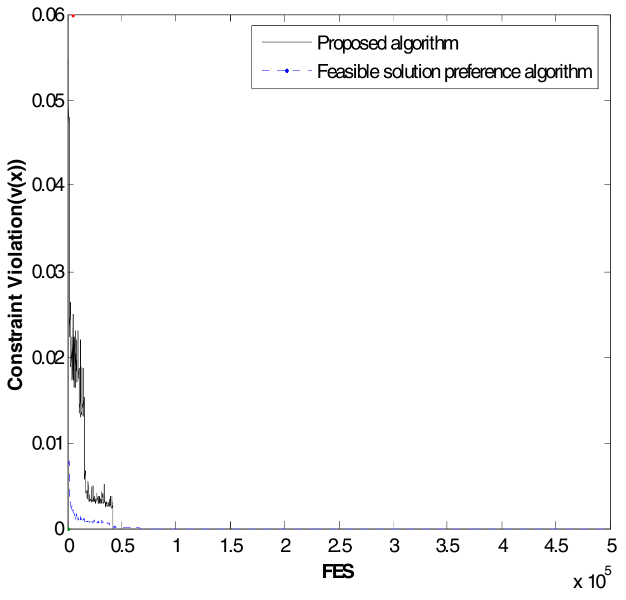


Figure 5.1 Convergence graph for g01

To show why the proposed algorithm performed better than other algorithms, the convergence graphs for the testing functions are shown in Figures 5.1 to 5.3. The figures show the function error (i.e. $f(\vec{x}) - f(\vec{x}^*)$) vs FES and constraint violation (i.e. $v(x)$) vs

FES for the best individual in the best runs for those three problems. The convergence graphs are slightly modified than those in [16]; instead of showing the logarithmic values of the function error and constraint violation, their exact values are shown as the function error can be negative.

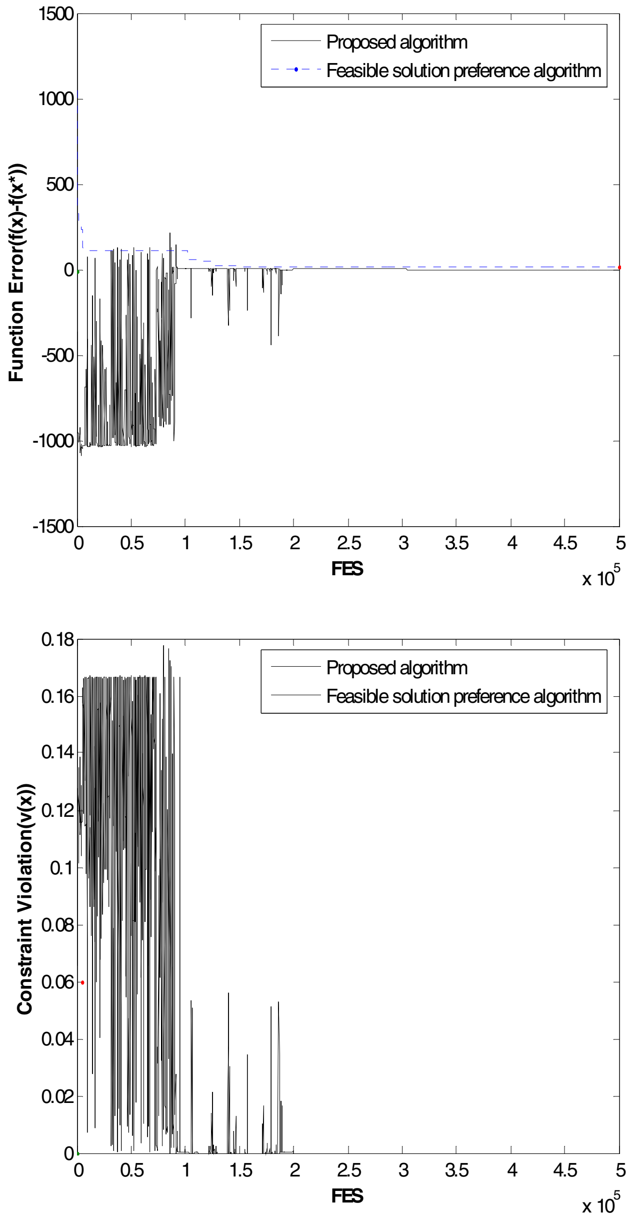


Figure 5.2 Convergence graph for g04

For comparison purpose in addition to the fitness assignment using the proposed algorithm another fitness assignment using preference of feasible solutions over infeasible ones is adopted. In this method, feasible solutions will always be better than infeasible solutions; between feasible solutions the one with better objective value is better; and between infeasible solutions the one with lower constraint violation is better. This method is chosen for comparison because most of the algorithms in the literature use this technique for constraint handling.

In Figures 5.1 we can see that the error values found by the proposed algorithm in the initial FES are negative which increase to a positive value as the FES increases. After a certain FES value is reached, the error starts to decrease quickly to a value slightly less than zero. This is because for g01 the feasibility ratio is very small and in the initial generations the population consisted of infeasible individuals only. Infeasible individuals can have negative error value as they do not satisfy the constraints. In the same figure, we can see that the constraint violation of the best individual decreases as the error value increases from negative to positive. On the other hand, for the preference of feasible individuals over infeasible individuals method, we can see that the error value is always positive which decreases very slowly to a value slightly greater than zero. This is due to the fact that only feasible individuals are given precedence in this method. The proposed algorithm will always produce better result because infeasible individuals with low constraint violation and very low fitness value (i.e. negative error value) are also given precedence, which in turn can lead us to finding feasible individuals with better fitness value.

In Figures 5.2 and 5.3 we can notice that for the initial few FES the error found by the proposed algorithm alternates between negative and positive values which is unlike the error found using the feasible solution preference method. In g04 the initial population always consisted of feasible solutions because the feasibility ratio is very large. Therefore instead of giving priority to feasible individuals as always, the algorithm tries to gather information from infeasible individuals as well by giving them better fitness value. Also for g06, which has very low feasibility ratio, as many infeasible individuals are better fitted as feasible individuals in the initial FES. This is in order to utilize the information contained in infeasible individuals.

From Figures 5.1 to 5.3 we can notice that the proposed algorithm produced better result than most of the algorithms in the literature because unlike most of them search for the optimal solution is done from two directions. One direction is minimizing the function error value of feasible individuals until an individual with minimum error is found. This is the search method that most algorithms, including preference of feasible solutions, employ. But the second search direction, which is equally important, is minimizing the constraint violation of infeasible individuals with very low or negative error value until zero constraint violation is reached. This search direction will assist the algorithm both in finding feasible solutions and also in finding the optimal solution.

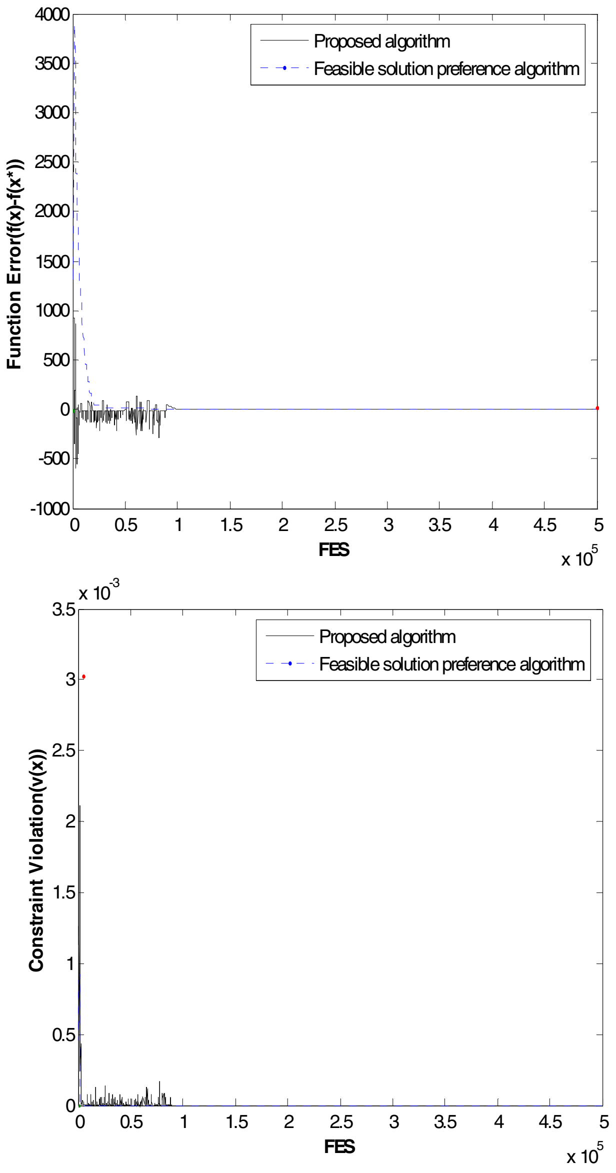


Figure 5.3 Convergence graph for g06

CHAPTER VI

CONCLUSIONS

In this thesis report a self-adaptive constraint handling algorithm is proposed that can be used with any generic search algorithm for solving constrained optimization problems. In developing the algorithm the main objective has been solving some of the drawbacks of previously designed algorithms for constrained optimization using evolutionary algorithms (EAs). One drawback, which is common with penalty function based constraint handling techniques, is the necessity of defining problem specific parameters which will make these types of methods impractical as the values of the parameters is problem dependant. Lack of reliability in finding feasible solutions for every run of the algorithm is also a drawback observed in most methods. In addition most algorithms, especially multiobjective optimization based constraint handling methods, are complex and computationally expensive.

As solution to these problems the algorithm is designed to be reliable, free of any parameter tuning and easy to implement. In addition it is designed to work well in problems having very small feasible space compared with the search space. The algorithm aims at exploiting the information hidden in infeasible individuals efficiently by selecting the proper individuals at different stages of the evolutionary process and under different conditions. Infeasible individuals carry two types of information. They carry information about the location of the feasible space and the location of the optimal solution. Therefore the main objective of the algorithm is to give priority to infeasible individuals with low constraint violation value whenever the need is locating the feasible space; and to infeasible individuals with better objective function value whenever the requirement is finding the optimal solution.

To achieve this objective, a new fitness called distance value and two penalty functions are introduced. The distance value, which is a measure of the objective function value and the constraint violation, will be assigned to every individual in a given population. The penalties, on the other hand, will be applied to infeasible individuals in order decrease their fitness compared to feasible individuals. The number of feasible individuals in the population adaptively determines the values of the distance and the two penalties. This will avoid the need of parameter tuning usually present in most of the constraint handling techniques in the literature. And finally the sum of the distance and the penalties will be used to rank and compare individuals in each generation of the search. This fitness formulation is a very flexible formulation that identifies the best infeasible individuals adaptively.

The performance of the algorithm is tested on 22 benchmark functions that resemble real world optimization problems and that are commonly used by different researchers for comparison. From the results produced it is observed that the algorithm is capable of finding feasible, quality solutions in all of the runs in the test functions using only 5×10^3 FES indicating the fact that the algorithm is computationally efficient. In particular the algorithm is faster and more accurate for low dimensional problems with inequality constraints only. This is because in these problems feasible solutions can be found easily and more focus can be given to finding the optimal solution. In addition the results of the algorithm are compared with some of the well-regarded algorithms in literature. The comparison results indicate that the proposed algorithm can perform as good as these algorithms. In fact the algorithm found better results than the already reported optimal results in four of the benchmark functions.

Although the algorithm is designed to work with any heuristic search algorithm, its performance for the test functions has been evaluated using GA as the main search algorithm. Therefore, as a future design further investigation should be done to check the performance of the algorithm using a different search algorithm, particularly differential evolution (DE). DE is a stochastic direct search method that is fast and robust to non-convex and multi-modal problems [38]. In addition further work is needed to test the algorithm in real world application problems. Moreover the algorithm can also be extended to solve muliobjective optimization problems with multiple constraints.

REFERENCES

[1]. A. H. Aguirre, S. B. Rionda, C. A. C. Coello, G. L. Lizaraga and E. Mezura-Montes, "Handling constraints using multiobjective optimization concepts," *International Journal for Numerical Methods in Engineering*, vol. 59, pp. 1989-2017, 2004.

[2]. 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.

[3]. T. Bäck and F. Hoffmeister "Extended selection mechanisms in genetic algorithms," *in Proceedings of* Fourth *International Conference on Genetic Algorithms,* San Diego, CA, pp 92–99, 1991.

[4]. J. Baker "Adaptive selection methods for genetic algorithms," *in Proceedings of First International Conference on Genetic Algorithms and Their Applications*, Hillsdale, NJ: Lawrence Erlbaum, pp 101–111, 1984.

[5]. J. C. Bean, A. B. Alouane, "A dual genetic algorithm for bounded integer programs," Technical Report TR 92-53, Department of Industrial and Operations Engineering, The University of Michigan, 1992.

[6]. B. P. Buckles and F. E. Petry, *Genetic Algorithms*, Technology Series, IEEE Computer Society Press, 1992.

[7]. C. A. C. Coello, "Treating constraints as objectives for single-objective evolutionary optimization," *Engineering Optimization*, vol. 32, pp. 275-308, 2000.

[8]. 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.

[9]. C. A. C. Coello and E. Mezura-Montes, "Constraint-handling in genetic algorithms through the use of dominance-based tournament selection," *Advanced Engineering Informatics*, vol. 16, pp. 193-203, 2002.

[10]. B. Craenen, A. Eiben, and J. Van Hemert, “Comparing evolutionary algorithms on binary constraint satisfaction problems,” *IEEE Transaction on Evolutionary Computation*, vol. 7, pp. 424-444, 2003.

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

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

[13]. L. Eshelman and J. Schaffer “Real-coded genetic algorithms and interval-schemata,” *Foundation of Genetic Algorithms*, pp 187–202, 1991.

[14]. R. Farmani and J. Wright, “Self-adaptive fitness formulation for constrained optimization,” *IEEE Transaction on Evolutionary Computation*, vol. 7, pp. 445-455, 2003.

[15]. P. Fleming and R.C. Purshouse. "Evolutionary algorithms in control systems engineering: a survey." *Control Engineering Practice*, vol.10, p.1223-1241, 2002.

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

[17]. D.E. Goldberg, *Genetic algorithms in search, optimization, and machine learning,* Addison-Wesley, Reading, 1989.

[18]. J. Horn, N. Nafpliotis and D.E. Goldberg, “A niched Pareto genetic algorithm for multiobjective optimization,” in *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, Piscataway, NJ, pp. 82-87, 1994.

[19]. J. Joines and C. Houck, “On the use of nonstationary penalty functions to solve nonlinear constrained optimization problems with GAs,” in *Proceedings of the First IEEE Congress on Evolutionary Computation*, Orlando, FL, pp. 579–584, 1994.

[20]. J. Kim and H. Myung, “Evolutionary programming techniques for constrained optimization problems,” *IEEE Transaction on Evolutionary Computation*, vol. 1, pp. 129–140, 1997.

[21]. J. D. Knowles and D. W. Corne, “Approximating the nondominated front using the Pareto archived evolution strategy,” *Evolutionary Computation*, vol. 8, pp. 149-172, 2000.

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

[23]. 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.

[24]. J. J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. N. Suganthan, C. A. C. Coello, and K. Deb, "Problem definitions and evaluation criteria for the CEC2006 special session on constrained real-parameter optimization," 2006.[Online] Available: http://www.ntu.edu.sg/home/EPNSugan/index_files/CEC-06/CEC06.htm .

[25]. E. Mezura-Montes, J. Velazquez-Reyes, and C.A.C. Coello, "Modified differential evolution for constrained optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation*, Vancouver, Canada, pp. 332-336 , 2006.

[26]. Z. Michalewicz, "A survey of constraint handling techniques in evolutionary computation methods," in *Proceedings of the 4th Annual Conference on Evolutionary Programming*, Cambridge, MA, pp. 135–155, 1995.

[27]. Z. Michalewicz and G. Nazhiyath, "GENOCOP III: A coevolutionary algorithm for numerical optimization problems with nonlinear constraints," in *Proceedings of Congress on Evolutionary Computation*, Perth, WA, Australia, pp. 647–651, 1995.

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

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

[30]. 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.

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

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

[33]. T.P. Runarsson and X. Yao, “Search biases in constrained evolutionary optimization,” *IEEE Transaction on Systems, Man and Cybernetics*, *Part C*, vol. 35, pp. 233-2443, 2005.

[34]. J. D. Schaffer, “Multiple objective optimization with vector evaluated genetic algorithms”, in *Proceedings of the 1st International Conference in Genetic Algorithms and their Applications* , Hillsdale, NJ, pp. 93-100, 1985.

[35]. M. Schoenauer and Z. Michalewicz, “Evolutionary Computation”, *Control and Cybernetics*, vo1. 26, pp.307-338, 1997.

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

[37]. T. Takahama and S. Sakai, “Constrained optimization by applying the α constrained method to the nonlinear simplex method with mutations,” *IEEE Transactions on Evolutionary Computation*, vol. 9, pp. 437-451, 2005.

[38]. T. Takahama and S. Sakai, “Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elite,” in *Proceedings of the IEEE Congress on Evolutionary Computation*, Vancouver, Canada, pp. 308-315 , 2006.

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

[40]. D. A. Van Veldhuizen, “Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations”, Ph. D. Thesis, Dayton, OH: Air Force Institute of Technology, 1999.

[41]. S. Venkatraman and G.G. Yen, “A generic framework for constrained optimization using genetic algorithms,” *IEEE Transaction on Evolutionary Computation*, vol. 9, pp. 424-435, 2005.

APPENDIX

BENCHMARK FUNCTIONS

1) g01

Minimize: $$f(\vec{x}) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^{\,2} - \sum_{i=5}^{13} x_i$$

subject to:

$$g1(\vec{x}) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \le 0$$

$$g2(\vec{x}) = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \le 0$$

$$g3(\vec{x}) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \le 0$$

$$g4(\vec{x}) = -8x_1 + x_{10} \le 0$$

$$g5(\vec{x}) = -8x_2 + x_{11} \le 0$$

$$g6(\vec{x}) = -8x_3 + x_{12} \le 0$$

$$g7(\vec{x}) = -2x_4 - x_5 + x_{10} \le 0$$

$$g8(\vec{x}) = -2x_6 - x_7 + x_{11} \le 0$$

$$g9(\vec{x}) = -2x_8 - x_9 + x_{12} \le 0$$

where the bounds are $0 \le x_i \le 1$ (i = 1,…, 9), $0 \le x_i \le 100$ (i = 10, 11, 12) and $0 \le x_{13} \le 1$. The global minimum is at $\vec{x}^*$ = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1) where six constraints are active (g1, g2, g3, g7, g8 and g9) and $f(\vec{x}^*) = -15$.

2) g02

Minimize: $$f(\vec{x}) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

subject to:

$$g1(\vec{x}) = 0.75 - \prod_{i=1}^{n} x_i \le 0$$

$$g2(\vec{x}) = \sum_{i=1}^{n} x_i - 7.5n \le 0$$

where n = 20 and $0 \le x_i \le 10$ (i = 1, . . . , n). The global minimum $\vec{x}^*$ = (3.16246061572185, 3.12833142812967, 3.09479212988791, 3.06145059523469, 3.02792915885555, 2.99382606701730, 2.95866871765285, 2.92184227312450, 0.49482511456933, 0.48835711005490, 0.48231642711865, 0.47664475092742, 0.47129550835493, 0.46623099264167, 0.46142004984199, 0.45683664767217, 0.45245876903267, 0.44826762241853, 0.44424700958760, 0.44038285956317), the best found is $f(\vec{x}^*)$ = -0.80361910412559, constraint g1 is close to being active.

3) g03

Minimize: $$f(\vec{x}) = -(\sqrt{n})^n \prod_{i=1}^{n} x_i$$

subject to:

$$h(\vec{x}) = \sum_{i=1}^{n} x_i^2 - 1 = 0$$

where n = 10 and $0 \le x_i \le 1$ (i = 1, . . . , n). The global minimum is at $\vec{x}^*$ = (0.31624357647283069, 0.316243577414338339, 0.316243578012345927, 0.316243575664017895, 0.316243578205526066, 0.31624357738855069, 0.316243575472949512, 0.316243577164883938, 0.316243578155920302, 0.316243576147374916) where $f(\vec{x}^*)$ = -1.00050010001000.

4) g04

Minimize: $$f(\vec{x}) = 5.3578547\, x_3^2 + 0.8356891\, x_1 x_5 + 37.293239\, x_1 - 40792.141$$

subject to.

$$g1(\vec{x}) = 85.334407 + 0.0056858\, x_2 x_5 + 0.0006262\, x_1 x_4 -$$
$$0.0022053\, x_3 x_5 - 92 \le 0$$

$$g2(\vec{x}) = -85.334407 - 0.0056858\, x_2\, x_5 - 0.0006262\, x_1\, x_4 +$$
$$0.0022053\, x_3\, x_5 \leq 0$$

$$g3(\vec{x}) = 80.51249 + 0.0071317\, x_2\, x_5 + 0.0029955\, x_1\, x_2 + 0.0021813\, x_2$$
$$3 - 110 \leq 0$$

$$g4(\vec{x}) = -80.51249 - 0.0071317\, x_2\, x_5 - 0.0029955\, x_1\, x_2 - 0.0021813\, x_2$$
$$3 + 90 \leq 0$$

$$g5(\vec{x}) = 9.300961 + 0.0047026\, x_3\, x_5 + 0.0012547\, x_1\, x_3 + 0.0019085\, x_3\, x_4$$
$$- 25 \leq 0$$

$$g6(\vec{x}) = -9.300961 - 0.0047026\, x_3\, x_5 - 0.0012547\, x_1\, x_3 - 0.0019085\, x_3\, x_4$$
$$+ 20 \leq 0$$

where $78 \leq x_1 \leq 102$, $33 \leq x_2 \leq 45$ and $27 \leq x_i \leq 45$ ($i = 3, 4, 5$). The optimum solution is $\vec{x}^* = (78, 33, 29.9952560256815985, 45, 36.7758129057882073)$ where $f(\vec{x}^*) = -3.066553867178332e + 004$. Two constraints are active (g1 and g6).

5) g05

Minimize: $f(\vec{x}) = 3\, x_1 + 0.000001\, x_1^3 + 2\, x_2 + (0.000002/3)\, x_2^3$

subject to.

$$g1(\vec{x}) = -x_4 + x_3 - 0.55 \leq 0$$

$$g2(\vec{x}) = -x_3 + x_4 - 0.55 \leq 0$$

$$h3(\vec{x}) = 1000 \sin(-x_3 - 0.25) + 1000 \sin(-x_4 - 0.25) + 894.8 - x_1 = 0$$

$$h4(\vec{x}) = 1000 \sin(x_3 - 0.25) + 1000 \sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0$$

$$h5(\vec{x}) = 1000 \sin(x_4 - 0.25) + 1000 \sin(x_4 - x_3 - 0.25) + 1294.8 = 0$$

where $0 \leq x_1 \leq 1200$, $0 \leq x_2 \leq 1200$, $-0.55 \leq x_3 \leq 0.55$ and $-0.55 \leq x_4 \leq 0.55$. The best known solution $\vec{x}^* = (679.945148297028709, 1026.06697600004691, 0.118876369094410433, -0.39623348521517826)$ where $f(\vec{x}^*) = 5126.4967140071$.

6) g06

Minimize: $f(\vec{x}) = (x_1 - 10)^3 + (x_2 - 20)^2$

subject to:

$$g1(\vec{x}) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0$$

$$g2(\vec{x}) = (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \leq 0$$

where $13 \leq x_1 \leq 100$ and $0 \leq x_2 \leq 100$. The optimum solution is $\vec{x}^* =$ (14.09500000000000064, 0.8429607892154795668) where $f(\vec{x}^*) = -6961.81387558015$. Both constraints are active.

7) g07

Minimize: $f(\vec{x}) = x_1^2 + x_2^2 + x_1 x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45$

subject to:

$$g1(\vec{x}) = -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \leq 0$$

$$g2(\vec{x}) = 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0$$

$$g3(\vec{x}) = -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0$$

$$g4(\vec{x}) = 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0$$

$$g5(\vec{x}) = 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \leq 0$$

$$g6(\vec{x}) = x_1^2 + 2(x_2 - 2)^2 - 2x_1 x_2 + 14x_5 - 6x_6 \leq 0$$

$$g7(\vec{x}) = 0.5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 \leq 0$$

$$g8(\vec{x}) = -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \leq 0$$

where $-10 \leq x_i \leq 10$ ($i = 1, \ldots, 10$). The optimum solution is $\vec{x}^* =$ (2.17199634142692, 2.3636830416034, 8.77392573913157, 5.09598443745173, 0.990654756560493, 1.43057392853463, 1.32164415364306, 9.82872576524495, 8.2800915887356, 8.3759266477347) where g07($\vec{x}$) = 24.30620906818 (The recorded results may suffer from rounding errors which may cause slight infeasibility sometimes in the best given solutions). Six constraints are active (g1, g2, g3, g4, g5 and g6).

8) g08

Minimize: $$f(\vec{x}) = -\frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1 + x_2)}$$

subject to:

$$g1(\vec{x}) = x_1^2 - x_2 + 1 \leq 0$$

$$g2(\vec{x}) = 1 - x_1 + (x_2 - 4)^2 \leq 0$$

where $0 \leq x_1 \leq 10$ and $0 \leq x_2 \leq 10$. The optimum is located at $\vec{x}^* =$ (1.22797135260752599, 4.24537336612274885) where $f(\vec{x}^*) = -0.0958250414180359$.

9) g09

Minimize: $$f(\vec{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$

subject to:

$$g1(\vec{x}) = -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0$$

$$g2(\vec{x}) = -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \leq 0$$

$$g3(\vec{x}) = -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0$$

$$g4(\vec{x}) = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0$$

where $-10 \leq x_i \leq 10$ for ($i = 1, \ldots, 7$). The optimum solution is $\vec{x}^* =$ (2.33049935147405174, 1.95137236847114592,-0.477541399510615805, 4.36572624923625874,-0.624486959100388983, 1.03813099410962173, 1.5942266780671519) where $f(\vec{x}^*) = 680.630057374402$. Two constraints are active (g1 and g4).

10) g10

Minimize: $$f(\vec{x}) = x_1 + x_2 + x_3$$

subject to:

$$g1(\vec{x}) = -1 + 0.0025(x_4 + x_6) \leq 0$$

$$g2(\vec{x}) = -1 + 0.0025(x_5 + x_7 - x_4) \leq 0$$

$$g3(\vec{x}) = -1 + 0.01(x_8 - x_5) \leq 0$$

$$g4(\vec{x}) = -x_1 x_6 + 833.33252 x_4 + 100 x_1 - 83333.333 \leq 0$$

$$g5(\vec{x}) = -x_2 x_7 + 1250 x_5 + x_2 x_4 - 1250 x_4 \leq 0$$

$$g6(\vec{x}) = -x_3 x_8 + 1250000 + x_3 x_5 - 2500 x_5 \leq 0$$

where 100 $\leq x_1 \leq$ 10000, 1000 $\leq x_i \leq$ 10000 (i = 2, 3) and 10 $\leq x_i \leq$ 1000 (i = 4, . . . , 8). The optimum solution is $\vec{x}^*$ = (579.306685017979589, 1359.97067807935605, 5109.97065743133317, 182.01769963061534, 295.601173702746792, 17.982300369384632, 286.41652592786852, 395.601173702746735), where $f(\vec{x}^*)$ = 7049.24802052867. All constraints are active (g1, g2 and g3).

11) g11

Minimize: $$f(\vec{x}) = x_1^2 + (x_2 - 1)^2$$

subject to:

$$h(\vec{x}) = x_2 - x_1^2 = 0$$

where -1 $\leq x_1 \leq$ 1 and -1 $\leq x_2 \leq$ 1. The optimum solution is $\vec{x}^*$ = (-0.707036070037170616, 0.500000004333606807) where $f(\vec{x}^*)$ = 0.7499.

12) g12

Minimize: $$f(\vec{x}) = -(100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2) = 100$$

subject to:

$$g(\vec{x}) = (x_1 - p)2 + (x_2 - q)2 + (x_3 - r)2 - 0.0625 \leq 0$$

where 0 $\leq x_i \leq$ 10 (i = 1, 2, 3) and p, q, r = 1, 2, . . . , 9. The feasible region of the search space consists of 93 disjointed spheres. A point (x_1, x_2, x_3) is feasible if and only if there exist *p, q, r* such that the above inequality holds. The optimum is located at $\vec{x}^*$ = (5, 5, 5) where $f(\vec{x}^*)$ = -1. The solution lies within the feasible region.

13) g13

Minimize: $$f(\vec{x}) = e^{x_1 x_2 x_3 x_4 x_5}$$

subject to:

$$h1(\vec{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_4^2 - 10 = 0$$

$$h2(\vec{x}) = x_2 x_3 - 5 x_4 x_5 = 0$$

$$h3(\vec{x}) = x_1^3 + x_2^3 + 1 = 0$$

where -2.3 ≤ x_i ≤ 2.3 (i = 1, 2) and -3.2 ≤ x_i ≤ 3.2 (i = 3, 4, 5). The optimum solution is $\vec{x}^*$ = (-1.71714224003, 1.59572124049468, 1.8272502406271, -0.763659881912867,-0.76365986736498) where $f(\vec{x}^*)$ = 0.053941514041898.

14) g14

Minimize: $$f(\vec{x}) = \sum_{i=1}^{10} x_i (c_i + \ln \frac{x_i}{\sum_{j=1}^{10} x_j})$$

subject to:

$$h1(\vec{x}) = x_1 + 2x_2 + +2x_3 + x_6 + x_{10} - 2 = 0$$

$$h2(\vec{x}) = x_4 + 2x_5 + x_6 + x_7 - 1 = 0$$

$$h3(\vec{x}) = x_3 + x_7 + x_8 + 2x_9 + x_{10} - 1 = 0$$

where the bounds are 0 ≤ x_i ≤ 10 (i = 1, . . . , 10), and c_1= -6.089, c_2 = -17.164, c_3 = -34.054, c_4 = -5.914, c_5 = -24.721, c_6 = -14.986, c_7 = -24.1, c_8 = -10.708, c_9 = -26.662, c_{10} = -22.179. The best known solution is at $\vec{x}^*$ = (0.0406684113216282, 0.147721240492452, 0.783205732104114, 0.00141433931889084, 0.485293636780388, 0.000693183051556082, 0.0274052040687766, 0.0179509660214818, 0.0373268186859717, 0.0968844604336845) where $f(\vec{x}^*)$ = -47.7648884594915.

15) g15

Minimize: $$f(\vec{x}) = 1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1 x_2 - x_1 x_3$$

subject to:

$$\text{h1}(\vec{x}) = x_1^2 + x_2^2 + x_3^2 - 25 = 0$$

$$\text{h2}(\vec{x}) = 8x_1 + 14x_2 + 7x_3 - 56 = 0$$

where the bounds are $0 \leq x_i \leq 10$ (i = 1, 2, 3). The best known solution is at $\vec{x}^*$ = (3.51212812611795133, 0.216987510429556135, 3.55217854929179921) where $f(\vec{x}^*)$ = 961.715022289961.

17) g17

Minimize: $$f(\vec{x}) = f(x_1) + f(x_2)$$

where

$$f_1(x_1) = \begin{cases} 30x_1 & 0 \leq x_1 < 300 \\ 31x_1 & 300 \leq x_1 \leq 400 \end{cases}$$

$$f_2(x_2) = \begin{cases} 28x_2 & 0 \leq x_2 < 100 \\ 29x_2 & 100 \leq x_2 < 200 \\ 30x_2 & 200 \leq x_2 < 1000 \end{cases}$$

subject to:

$$\text{h1}(\vec{x}) = -x_1 + 300 - ((x_3 x_4)/131.078) \cos(1.48477 - x_6) + ((0.90798 x_3^2)/131.078) \cos(1.47588)$$

$$\text{h2}(\vec{x}) = -x_2 - ((x_3 x_4)/131.078) \cos((1.48477 + x_6) + ((0.90798 x_4^2)/131.078) \cos(1.47588)$$

$$\text{h3}(\vec{x}) = -x_5 - ((x_3 x_4)/131.078) \sin((1.48477 + x_6) + ((0.90798 x_4^2)/131.078) \sin(1.47588)$$

$$\text{h4}(\vec{x}) = 200 - ((x_3 x_4)/131.078) \sin((1.48477 - x_6) + ((0.90798 x_3^2)/131.078) \sin(1.47588)$$

where the bounds are $0 \le x_1 \le 400$, $0 \le x_2 \le 1000$, $340 \le x_3 \le 420$, $340 \le x_4 \le 420$, $-1000 \le x_5 \le 1000$ and $0 \le x_6 \le 0.5236$. The best known solution is at $\vec{x}^* =$ (201.784467214523659, 99.9999999999999005, 383.071034852773266, 420, -10.9076584514292652, 0.0731482312084287128) where $f(\vec{x}^*) = 8853.53967480648$.

18) g18

Minimize: $$f(\vec{x}) = -0.5(x_1 x_4 - x_2 x_3 + x_3 x_9 - x_5 x_9 + x_5 x_8 - x_6 x_7)$$

subject to.

$$g1(\vec{x}) = x_3^2 + x_4^2 - 1 \le 0$$

$$g2(\vec{x}) = x_9^2 - 1 \le 0$$

$$g3(\vec{x}) = x_5^2 + x_6^2 - 1 \le 0$$

$$g4(\vec{x}) = x_1^2 + (x_2 - x_9)^2 - 1 \le 0$$

$$g5(\vec{x}) = (x_1 - x_5)^2 + (x_2 - x_6)^2 - 1 \le 0$$

$$g6(\vec{x}) = (x_1 - x_7)^2 + (x_2 - x_8)^2 - 1 \le 0$$

$$g7(\vec{x}) = (x_3 - x_5)^2 + (x_4 - x_6)^2 - 1 \le 0$$

$$g8(\vec{x}) = (x_3 - x_7)^2 + (x_4 - x_8)^2 - 1 \le 0$$

$$g9(\vec{x}) = x_7^2 + (x_8 - x_9)^2 - 1 \le 0$$

$$g10(\vec{x}) = x_2 x_3 - x_1 x_4 \le 0$$

$$g11(\vec{x}) = -x_3 x_9 \le 0$$

$$g12(\vec{x}) = x_5 x_9 \le 0$$

$$g13(\vec{x}) = x_6 x_7 - x_5 x_8 \le 0$$

where the bounds are $-10 \le x_i \le 10$ ($i = 1, \ldots, 8$) and $0 \le x_9 \le 20$. The best known solution is at $\vec{x}^* =$ (-0.657776192427943163,-0.153418773482438542, 0.323413871675240938,-0.946257611651304398, - 0.657776194376798906, -0.753213434632691414, 0.323413874123576972,-0.346462947962331735, 0.59979466285217542) where $f(\vec{x}^*) = -0.866025403784439$.

19) g19

Minimize: $$f(\vec{x}) = \sum_{j=1}^{5}\sum_{i=1}^{5} c_{ij} x_{(10+i)} x_{(10+j)} + 2\sum_{j=1}^{5} d_j x_{(10+j)}^3 - \sum_{i=1}^{10} b_i x_i$$

subject to:

$$g_j(\vec{x}) = -2\sum_{i=1}^{5} c_{ij} x_{(10+i)} - 3d_j x_{(10+j)}^2 - e_j + \sum_{i=10}^{10} a_{ij} x_i \leq 0 \quad j = 1,\ldots,5$$

where $\vec{b}$ = [-40,-2,-0.25,-4,-4,-1,-40,-60, 5, 1] and the remaining data is on Table A.1. The bounds are 0 · xi · 10 (i = 1, . . ., 15). The best known solution is at $\vec{x}^*$ = (1.66991341326291344e -17, 3.95378229282456509e-16, 3.94599045143233784, 1.06036597479721211e-16, 3.2831773458454161, 9.99999999999999822, 1.12829414671605333e-17, 1.2026194599794709e-17, 2.50706276000769697e-15, 2.24624122987970677e-15, 0.370764847417013987, 0.278456024942955571, 0.523838487672241171, 0.388620152510322781, 0.298156764974678579) where $f(\vec{x}^*)$ = 32.6555929502463.

TABLE A.1

Data set for testing problem g19

j	1	2	3	4	5
e_j	-15	-27	-36	-18	-12
c_{1j}	30	-20	-10	32	-10
c_{2j}	-20	39	-6	-31	32
c_{3j}	-10	-6	10	-6	-10
c_{4j}	32	-31	-6	39	-20
c_{5j}	-10	32	-10	-20	30
d_j	4	8	10	6	2
a_{1j}	-16	2	0	1	0
a_{2j}	0	-2	0	0.4	2
a_{3j}	-3.5	0	2	0	0
a_{4j}	0	-2	0	-4	-1
a_{5j}	0	-9	-2	1	-2.8
a_{6j}	2	0	-4	0	0
a_{7j}	-1	-1	-1	-1	-1
a_{8j}	-1	-2	-3	-2	-1
a_{9j}	1	2	3	4	5
a_{10j}	1	1	1	1	1

20) g21

Minimize: $f(\vec{x}) = x_1$

subject to:

$$g1(\vec{x}) = -x_1 + 35x_2^{0.6} + 35x_3^{0.6} \leq 0$$

$$h1(\vec{x}) = -300x_3 + 7500x_5 - 7500x_6 - 25x_4x_5 + 25x_4x_6 + x_3x_4 = 0$$

$$h2(\vec{x}) = 100x_2 + 155.365x_4 + 2500x_7 - x_2x_4 - 25x_4x_7 - 15536.5 = 0$$

$$h3(\vec{x}) = -x_5 + \ln(-x_4 + 900) = 0$$

$$h4(\vec{x}) = -x_6 + \ln(x_4 + 300) = 0$$

$$h5(\vec{x}) = -x_7 + \ln(-2x_4 + 700) = 0$$

where the bounds are $0 \leq x_1 \leq 1000$, $0 \leq x_2, x_3 \leq 40$, $100 \leq x_4 \leq 300$, $6.3 \leq x_5 \leq 6.7$, $5.9 \leq x_6 \leq 6.4$ and $4.5 \leq x_7 \leq 6.25$. The best known solution is at $\vec{x}^* =$ (193.724510070034967, 5.56944131553368433e- 27, 17.3191887294084914, 100.047897801386839, 6.68445185362377892, 5.99168428444264833, 6.21451648886070451) where $f(\vec{x}^*) = 193.724510070035$.

21) g23

Minimize: $f(\vec{x}) = -9x_5 - 15x_8 + 6x_1 + 16x_2 + 10(x_6 + x_7)$

subject to:

$$g1(\vec{x}) = x_9x_3 + 0.02x_6 - 0.025x_5 \leq 0$$

$$g2(\vec{x}) = x_9x_4 + 0.02x_7 - 0.015x_8 \leq 0$$

$$h1(\vec{x}) = x_1 + x_2 - x_3 - x_4 = 0$$

$$h2(\vec{x}) = 0.03x_1 + 0.01x_2 - x_9(x_3 + x_4) = 0$$

$$h3(\vec{x}) = x_3 + x_6 - x_5 = 0$$

$$h4(\vec{x}) = x_4 + x_7 - x_8 = 0$$

where the bounds are $0 \leq x_1, x_2, x_6 \leq 300$, $0 \leq x_3, x_5, x_7 \leq 100$, $0 \leq x_4, x_8 \leq 200$ and $0.01 \leq x_9 \leq 0.03$. The best known solution is at $\vec{x}^*$ = (0.00510000000000259465, 99.9947000000000514, 9.01920162996045897e-18, 99.9999000000000535, 0.000100000000027086086, 2.75700683389584542e-14, 99.9999999999999574, 2000.0100000100000100008) where $f(\vec{x}^*)$ = -400.055099999999584.

22) g24

Minimize: $$f(\vec{x}) = -x_1 - x_2$$

subject to:

$$g1(\vec{x}) = -2x_1^4 + 8x_1^3 - 8x_1^2 + x_2 - 2 \leq 0$$

$$g2(\vec{x}) = -4x_1^4 + 32x_1^3 - 88x_1^2 + 96x_1 + x_2 - 36 \leq 0$$

where the bounds are $0 \leq x_1 \leq 3$ and $0 \leq x_2 \leq 4$. The feasible global minimum is at $\vec{x}^*$ = (2:329520197477623:17849307411774) where $f(\vec{x}^*)$ = -5:50801327159536. This problem has a feasible region consisting on two disconnected sub-regions.

VITA

Biruk Girma Tessema

Candidate for the Degree of

Master of Science

Thesis: A SELF-ADAPTIVE GENETIC ALGORITHM FOR CONSTRAINED OPTIMIZATION

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Addis Ababa, Ethiopia, on September 10, 1981.
Current Address: 36 South Univ. Apt 4, Stillwater, OK.

Education: Graduated from St. Joseph High School, Addis Ababa, Ethiopia, in June 1999.
Received Bachelors of Science degree in Electrical Engineering from Bahirdar University, Bahirdar, Ethiopia, in July 2004.
Completed the requirements for the Masters of Science degree with major in Electrical Engineering at Oklahoma State University in December 2006.

Experience: Research and Teaching Assistant, Oklahoma State University, Stillwater, OK.
Junior Electrical Engineer, ZTE Telecommunications, Addis Ababa, Ethiopia.
Distribution Systems Engineer, Ethiopian Electric Power Corporation, Addis Ababa, Ethiopia.
Part-time Lecturer, CPU College, Addis Ababa, Ethiopia.

Professional Memberships: IEEE Computational Intelligence Society.

Name: Biruk Girma Tessema Date of Degree: December 2006

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: A SELF-ADAPTIVE GENETIC ALGORITHM FOR CONSTRAINED OPTIMIZATION

Pages in Study: 75 Candidate for the Degree of Master of Science.

Major Field: Electrical Engineering

Scope and Method of Study: This study proposes a self-adaptive penalty function algorithm for solving constrained optimization problems using genetic algorithm (GA). Constrained optimization is a practically relevant and challenging field that deals with optimization of real world problems that involve complex constraints that make them difficult to tackle. GA is a stochastic search method based on the evolutionary ideas of natural selection and genetic. In GA candidate solutions to a certain problem, called individuals, will evolve from generation to generation toward finding better solutions. In this research GA based constraint handling algorithm is proposed that combines the merits of previously designed algorithms. In the proposed method a new fitness value, called distance value, and two penalties are applied to infeasible individuals that violate the constraints. The algorithm aims to encourage infeasible individuals with better 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 the optimum solution or toward finding more feasible solutions.

Findings and Conclusions: The performance of the algorithm is tested on 22 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.

ADVISER'S APPROVAL: ______________________________

Dr Gary G. Yen