A GENETIC ALGORITHM DESIGN FOR CONSTRAINED OPTIMIZATION

By

SANGAMESWAR VENKATRAMAN

Bachelor of Engineering

University of Madras

Chennai, India

May 2002

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
July, 2004

A GENETIC ALGORITHM DESIGN FOR

CONSTRAINED OPTIMIZATION

Thesis Approved:

[illegible]

Thesis Advisor

[illegible]

R. Ramakumar

[illegible]

Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to acknowledge my thesis advisor, Dr. Gary G Yen for his continued support, leadership, and encouragement during the course of this study. He always provided inspiring suggestions and helped in canalizing my efforts in the right direction. I would also like to thank the members of my advisory committee Dr. Rama Ramakumar and Dr. Rafael Fierro for their time, effort and interest shown in this thesis.

A major credit also goes to the members of the Intelligent Systems and Control Lab who helped me with most of my technical questions. A special thanks to Michel L. Goldstein for his kind help and constructive criticisms which have helped add quality to this thesis.

Thanks of course to my family physically far away who always stressed that "Efforts may fail, but one should not fail to make efforts."

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
INTRODUCTION

1.1 Optimization

What is optimization? The Merriam-Webster dictionary defines optimization as

"an act, process, or methodology of making something (as a design, system, or decision) as fully perfect, functional, or effective as possible; specifically the mathematical procedures (as finding the maximum of a function) involved in this."

Most day-to-day problems can be formulated as optimization problems. The mathematical formulation of the problem is paramount to the design of appropriate algorithms. Hence algorithms for even problems like planning one's day can be designed if it is formulated mathematically. Well-formulated optimization problems possess the following ingredients:

- An **Objective Function** which we want to maximize or minimize. For instance, we might want to maximize the profit or minimize the cost of a manufacturing process.
- A set of **Decision Variables** which affect the value of the objective function. In the manufacturing process, the decision variables may be the amounts of different materials used or the time spent on each activity.

- A set of **Constraints** allow the unknown to take certain values but excludes others. For the manufacturing process problem, it does not make sense to spend a negative amount of time on any activity, so we constrain all the time variables to be non-negative.

Now the optimization problem may be redefined from [38] as:

"Finding values of the decision variables that minimize or maximize the objective function while satisfying the constraints."

1.2 Constrained Optimization

Most real world optimization problems involve constraints. Consider a real world optimization problem such as maximizing the profits of a particular manufacturing process. Here the objective function to be maximized would be a function of various manipulating variables, including but not limited to the material consumption, the labor cost, the operating hours of the machines, and many additional factors. If the raw materials, manpower, and machines can be made available without limitation then there is no limit to the profit that can be made. However, in face of real world complications they are most likely limited in the form of constraints imposed upon the optimization function. What constitute the difficulties of the constrained optimization problem are the various limits on the decision variables, the constraints involved, the interference among constraints, and the inter-relationship between the constraints and the objective function.

1.3 Problem Formulation

The general constrained continuous parameter optimization problem as succinctly defined in [24] is to find X so as to

$$\text{Minimize } f(X), \quad X = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n \tag{1.1}$$

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

$$l(i) \le x_i \le u(i), \quad 1 \le i \le n \tag{1.2}$$

whereas the feasible region $\mathbb{F}$ is defined by a set of m additional constraints ($m \ge 0$),

$$g_j(X) \le 0, \quad (j = 1, \ldots, q) \tag{1.3}$$

$$h_j(X) = 0, \quad (j = q+1, \ldots, m). \tag{1.4}$$

where there are q inequality constraints and m-q equality constraints.

If any point X satisfies all the m constraints, then it is called a *feasible point* and the set of all such points is defined as the feasible region $\mathbb{F}$. The inequality constraints that take the value of 0, i.e. $g_j(X)=0$ at the global optimum to the problem are called the *active constraints*. We assume in our above definition of constrained optimization problem that the decision variables in X do not just take some discrete values. Hence the formulation in (1.1) applies for all kinds of continuous optimization problems. We know that given a function $f(X)$ for maximization, we can transform it to a minimization problem by using $\max f(X) = \min(-f(X))$. Hence in the following discussion and in the remaining of this thesis without loss of generality we shall consider the minimization of the objective function unless specified otherwise.

1.4 Scope Of The Research

The goal of this thesis is to come up with a unified method for global optimization algorithm for constrained problems of the following kind.

1) Linear Programming problems: It is the simplest of the constraint optimization problems and the methods for solving it are fairly well established. The basic problem of linear programming is to minimize a linear objective function of continuous real variables, subject to linear constraints. The linear programming problem is formulated as,

Minimize $f(X) = c^T X$

with q inequality constraints $g_j(X) \leq 0, \quad (j = 1, \ldots, q)$

and m-q equality constraints $h_j(X) = 0, \quad (j = q+1, \ldots, m)$.

The feasible region described by this problem is *convex* due to the linear functions. The simplex algorithm is used to solve these kinds of problems.

2) Bound Constrained Optimization problems: The only addition to an unconstrained problem is the presence of an upper and lower bound on each of the decision variables. This limits the possible solution to a particular range and these are the only constraints imposed on the problem.

Minimize $f(X)$

$l(i) \leq x_i \leq u(i), \quad 1 \leq i \leq n$

3) Quadratic Programming problems: The quadratic programming problem involves the minimization of a quadratic function subject to linear constraints. The following formulation is used in most cases,

Minimize $f(X) = \frac{1}{2} X^T Q X + c^T X$

with q inequality constraints $g_j(X) \le 0, \quad (j = 1, \ldots, q)$

and m-q equality constraints $h_j(X) = 0, \quad (j = q+1, \ldots, m)$.

where $Q \in R^{n \times n}$ is symmetric and both $g_j(X)$ and $h_j(X)$ are linear functions of X. The difficulty of solving the quadratic programming problem depends largely on the nature of the matrix Q. In *convex* quadratic problems which are relatively easier to solve, the matrix Q is positive semi definite. If Q has negative eigen values-nonconvex quadratic programming-then the objective function may have more than one local minimum.

4) Nonlinearly Constrained Optimization problems: The main techniques that have been proposed for solving constrained optimization problems are reduced-gradient methods, sequential linear and quadratic programming methods, and methods based on augmented lagrangian and exact penalty methods.

5) Non-differentiable Programming problems: In these problems the objective function and the constraint functions are not differentiable or defined over finite intervals. Though such problems are fairly well-prevalent no method has been known to effectively solve such problems.

1.5 Features Of Conventional Approaches

1) A local minimizer (X^*) is looked for rather than a global minimizer, the computation of which can be difficult.

2) Most practical methods require the strong assumption that the first and second order derivatives of the objective function and constraint functions exist. A different situation arises when the functions $f(X)$ and $h_j(X)$ and $g_j(X)$ do not have continuous derivatives which are referred to as *non-differentiable* or *non-smooth optimization.*

3) Methods used for solving the problem are usually iterative so that a sequence $X^{(1)}, X^{(2)}, X^{(3)}$ hopefully converging to X^* (which is a local minimum).

1.6 Features Of Genetic Algorithms (GAs)

While the details of implementation of GAs will be discussed in Chapter 2, we enumerate here the features of GAs [37]. This section presents the reasoning behind using GAs to solve the constrained optimization problem discussed above.

1) GAs search a population of points in parallel, not a single point.

2) GAs use probabilistic transition rules, not deterministic ones

3) GAs do not require derivative information or other auxiliary knowledge; only the objective function and the corresponding fitness levels influence the direction of the search.

These features make the GA quite different from the traditional methods we mentioned above. Thus the reasons why GAs are more suitable for solving the constrained optimization problem as opposed to traditional methods is given below,

1) Since GAs work with a population of solutions selected in a stochastic fashion, GAs are capable of escaping the local optima and find the global solution. In fact as $t \rightarrow \infty$, the GA guarantees to find the global optimum to the problem. Hence GAs are *Global Optimizers.*

2) Since GAs do not require any derivative information from the problem, this makes it suitable for a wide variety of problems. This also makes it possible for us to design one

generalized algorithm for all five types of constraint optimization problems we discussed before including the *non-smooth optimization* problems.

On the flip side the following two reasons make GA's a not-so-favorable choice,

1) Since the GAs are stochastic search techniques, there is no assurance of the quality of the final solutions reached. The difficulty in using GA's for constrained optimization problems is that the final solution may not even be feasible. Hence multiple runs of the GA's are often needed to guarantee feasible optimal solutions from the GA.

2) GAs are time consuming because they work with a population of solutions over a preset number of generations. One way to overcome it is to implement the GA using parallel processors. The challenge is to design better algorithms that can make the search more efficient and converge faster.

But the pros weighted over the cons and we decided to use GAs for the Constrained Optimization problem discussed above. Following up with decision, we have tried to overcome the disadvantages of GAs discussed above. Chapter 5 describes our proposed algorithm in detail.

1.7 Layout Of The Thesis

The goal of the thesis is to produce a reliable and fast algorithm which can be applied to a wide variety of constrained optimization problems. Towards this end, the research work was done in three parts,

DESIGN: The first part of the thesis deals with design of the appropriate algorithm. The desired features of the algorithm are that

1) it should assure feasible solutions for every run and

2) it should be composed of a generic framework that can be applied to any constrained optimization problem with the characteristics discussed in Section 1.4.

Chapter 4 discusses the design of the appropriate algorithm after a literature survey of the genetic algorithms used for constrained optimization problems is provided in Chapter 3.

TESTING: An exhaustive testing of the proposed algorithm as well as comparison with other algorithms from literature is presented in Chapter 5. In the first part of the testing, a test case generator[31] for constrained optimization problems was implemented from literature. The tests were conducted during the design stage to compare two selection schemes for the algorithm. The second part of the testing involved eleven test problems taken from reference [25] and this helped us to compare our algorithm with many others proposed in literature.

APPLICATION: We implemented our algorithm to solve the economic dispatch problem for power systems. This happens to be one of the typical applications of the algorithm. The economic dispatch problem is a non-linear constrained problem and research is still ongoing to solve it effectively and efficiently. Chapter 6 provides the complete problem formulation as well as the results obtained with our proposed algorithm.

Chapter 2

WHAT ARE GENETIC ALGORITHMS?

2.1 Introduction

Genetic Algorithms (GAs) are *stochastic search techniques* that mimic biological evolution. GAs are inspired by the mechanism of natural selection as proposed by Darwin, in which better-fitted individuals are more likely to be winners in a competing environment. The basic principles of GAs were proposed by John Holland in the 1970's. GAs operate on a population of potential solutions and this population is replaced by a hopefully fitter population at each generation. In other words, starting from a randomly initialized population the genetic algorithm proceeds along the generations to produce fitter and fitter solutions that are better suited to the problem domain. At every generation a new set of individuals are created by the process of selecting individuals according to their levels of fitness in the problem domain. New solutions are produced by breeding them together using operators borrowed from natural genetics. This process leads to the evolution of populations of individuals which are better suited to their environment than the individuals they were created from.

2.2 Major Elements of Genetic Algorithms (GAs)

A GA is defined by five elements: representation, fitness evaluation, selection scheme, genetic operations and termination. Let us first analyze a simple genetic algorithm (SGA) from [16] given in Figure 2.1,

```
Procedure GA
begin
        $t = 0$;
        initialize $P(t)$;
        evaluate $P(t)$;
        while not finished do
        begin
                $t=t+1$;
                select $P(t)$ form $P(t+1)$
                reproduce pairs in $P(t)$
                evaluate $P(t)$
        end
end
```

FIGURE 2.1 A simple genetic algorithm

The population at time *t* is represented by the time-dependent variable *P*, with the initial population of random estimates being *P(0)*. Using this outline of a GA, the rest of this section describes the major elements of a GA. Most of the material used in this section are taken from [37].

2.2.1 Population Representation and Initialization: GA's operate simultaneously on a number of potential solutions, called a population consisting of some encoding of the parameter set. There are two major schemes in representation viz. binary representation and real-valued representation.

Binary Representation: The most commonly used representation in the GA is that of single-level binary string. Here, each decision variable in the parameter set is encoded as a binary string and these strings are concatenated to form the chromosome. Gray level representations of the decision variables are also used. The advantage of the gray level coding is that the hamming distance between adjacent values is constant.

Real valued Representation: The decision variable can be used in the same decimal form in the GA without the need for any conversion to binary string. The initial GA's were based on binary coding to mimic the genetic sequence in natural evolution. However real-valued solutions offer some distinct advantages, like increased efficiency, less memory requirements, direct usage of floating point internal computer representation, no loss in precision by discretization to binary or other values and greater freedom to use different genetic operators.

Having decided on the representation, the first step in the GA is to create an initial population. This is achieved by generating the number of individuals using a random number generator which uniformly generates numbers in a given range.

2.2.2 Objective Function and Fitness Evaluation: The objective function is usually defined explicitly in the problem. In some cases we have to design objective functions based on the problem domain. In either case, the objective function provides a measure of how individuals have performed in the problem domain. In the case of a minimization problem, the individual with the lowest numerical value will be considered the fittest. This raw measure of fitness is only used as an intermediate stage in determining the relative performance of individuals in a GA.

Another function, called the fitness function, is normally used to transform the objective function value into a measure of the fitness of the solution,

$$F(X) = g(f(X))$$

Where f is the objective function, g transforms the value of the objective function to a non-negative number and F is the resulting relative fitness. The fitness of all individuals are usually normalized such that the sum of all the fitness is either unity or another integer value. This makes implementing the selection scheme straightforward since the bigger the fitness of each individual, the greater the probability of its selection.

In rank-based fitness scheme, solutions are assigned a rank based on their objective function values. Thus the fitness of the solutions are only indirectly dependent on their objective function values. This allows the solutions to be scaled linearly or non-linearly as per the user's design.

2.2.3 Selection Scheme: After each solution in the given population is allotted a fitness value, the selection scheme determines the probabilistic way in which individuals are selected.

Roulette Wheel Selection: Let Ω be the sum of the fitness values of all individuals such that $\sum_{i=1}^{popsize} F_i(X) = \Omega$. Then the range $0 - \Omega$ is split between all the individuals based on their fitness value. Thus clearly an individual with a greater fitness value will have a greater range in $0 - \Omega$. Now a random number is selected between $0 - \Omega$ and the individual's fitness in whose range it falls is selected. If j is the number of individuals to be selected, then this process is repeated j times.

Stochastic Universal Sampling: Stochastic universal sampling is a single phase sampling algorithm with minimum spread and zero-bias. The stochastic universal selection is also based on a scheme similar to roulette wheel selection. First a number is randomly generated in the range of $0-(\Omega/j)$ and then the other numbers are found by subsequently chosen by adding (Ω/j) to it $j-1$ times. Thus we can say that the range of $0-\Omega$ will be uniformly covered in this selection process.

2.2.4 Genetic Operators: After the initial population has been generated randomly and the selection scheme is designed, it is up to the genetic operators to produce the subsequent populations and eventually the final optimal solutions. Hence the genetic operators are indeed important and there are two major types.

Crossover: The basic operation for producing new chromosomes in the GA is that of crossover. Like its counterpart in nature, crossover produces new individuals which have some features of both parent's genetic material ('0' and '1' bits).

The popular types of crossover among binary-coded chromosomes are the single-point crossover, multi-point crossover and the uniform crossover as shown in Figures 2.2, 2.3 and 2.4 respectively. Let P1 and P2 be the two chromosomes selected for crossover, then a point is randomly selected in the chromosome and genetic material after this point is exchanged to produce two new chromosomes C1 and C2 respectively. P1 and P2 are called the parents and C1 and C2 are the children.

P1: 111	01011	C1: 111	00110
P2: 011	00110	C2: 011	01011

FIGURE 2.2 Single-point crossover

In multi-point crossover more than one site is chosen for swapping of genetic material. Figure 2.3 shows the double-point crossover, a particular case of the multi-point crossover. Notice that the genetic material between the two chosen points is swapped from P1 and P2 to produce C1 and C2.

P1: 111	01	011	C1: 111	00	011
P2: 011	00	110	C2: 011	01	110

FIGURE 2.3 Multi-point crossover

In uniform crossover, a randomly generated masking string is used to swap the genetic material between the two parents P1 and P2. For example, if the mask at a particular bit position is '0', then C1 inherits the bit from parent 1 and C2 from parent 2 respectively. Otherwise if the mask at the particular bit position is '1', then C1 inherits the bit from P2 and C2 inherits the bit from P1.

P1:	11101011
P2:	01100110
Mask:	00111010
C1:	11100011
C2:	01101110

FIGURE 2.4 Uniform crossover

When real-coded chromosomes are used crossover can be implemented by converting those to binary strings or directly. Intermediate crossover is such a type of crossover that is applied directly on the real-valued chromosome. The offspring is produced around and between the values of the parent's phenotypes according to,

$$O = P_1 \times \alpha(P_2 - P_1)$$

Thus one off-spring is produced for every unique value of α and pair of parents. The α is the scaling factor chosen in some interval typically $[-0.25, 1.25]$. In geometric terms, intermediate crossover is capable of producing new variables within a slightly larger hypercube than that defined by the parents but constrained by the range of α.

Mutation: In natural evolution, mutation is a random process where one allele of a gene is replaced by another to produce a new genetic structure. In GAs mutation is randomly applied with low probability typically in the range of 0.001 and 0.01. The role of mutation is to provide a guarantee that the probability of searching any string will never be zero. Mutation assures the probability of the GA finding the global optimum given infinite time to be one. Figure 2.5 shows the mutation operator on a solution called P to produce a solution C. Here the bit in the fourth position (randomly chosen) is flipped to produce C.

P: 111	0	1011
C: 111	1	1011

FIGURE 2.5 Binary mutation

For non-binary representations, mutation is achieved by either perturbing the gene values or randomly selecting new values within the allowed range. The two important factors in mutation is the probability of mutation (which defines the number of chromosomes mutated in the population) and the amount of perturbation on each mutated string (which depends on the design of the mutation operator).

2.2.5 Termination of the GA: Because the GA is a stochastic search technique, it is difficult to formally specify the convergence criteria. Also because the presence of mutation always offers a chance of finding "the solution," it is impossible to decide when to terminate the GA. In most cases, the user has no idea about where the global optimum is and hence not in a position to objectively determine the solution's merit. In [30], the authors used a technique wherein the algorithm was terminated if no appreciable improvement in the quality of solutions is found over a preset number of generations. The most commonly used technique is to run the GA many times for a particular number of generations and then evaluate the quality of the solutions obtained.

At this point, it has to be acknowledged that this section on the various elements of GAs is by no means complete. Rather, the idea has only been to introduce the GA and the interested reader is referred to [16] and [37] for a more detailed explanation. Also there are many different types of GAs which by themselves demand exhaustive descriptions. So after this subtle introduction to GAs, the next section follows up with the features of the particular GA used in this research.

2.3 Three GA Concepts Important To This Research

Elitism: The most important feature of the algorithm proposed in this research is elitism. Elitism assures that a top proportion of the population is carried through to the next generation without undergoing any changes. Elitism is defined by a percentage value of the population size. For example in a population of size 10 with elitism of 10%, the best solution in the population is copied onto the next generation as it is. When large populations are used, elitism is used by maintaining an archive of top solutions and using some of them to perform genetic operations with the others in the population.

Non-dominated Ranking: So far in this chapter, we have been discussing a case where only a single objective function that has to be optimized. But what if there are more than one objective functions to be optimized simultaneously. Conventionally a weighted sum of these objective functions would have been optimized. But how do we determine the weights? To analyze this further let us consider a bi-objective optimization problem (problem with two objective functions). Of course the discussion is exactly applicable for any number of objective functions to be optimized simultaneously.

When the objective functions are plotted one vs. the other, then it is called the "objective space" of the problem. Figure 2.6 shows the objective space of the bi-objective problem where we are simultaneously trying to minimize F1 and F2 simultaneously. Hence the optimal solutions should be at the lower left hand of the graph as near to the origin of the graph as possible. So the exact objective functions do not affect the desired region of the objective space. We have plotted 10 fictional solutions in this objective space in Figure 2.6.

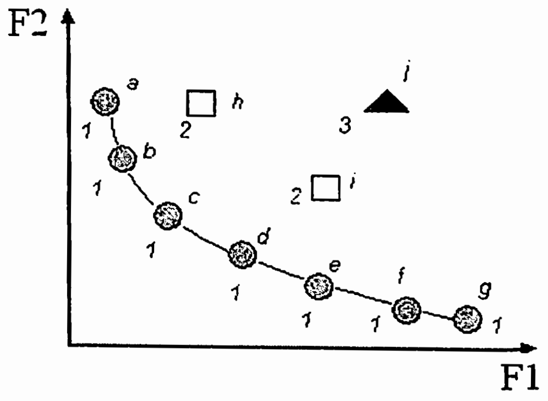

FIGURE 2.6 Objective Space of bi-objective problem

Then from [13], we have the following definitions which help us to rank solutions,

Definition 1 (inferiority): A vector **u** $=(u_1,\ldots,u_n)$ is said to be inferior to **v** $=(v_1,\ldots,v_n)$ *iff* v is partially less than **u** i.e., $\forall i=1,\ldots,n, v_i \le u_i \;\; \wedge \;\; \exists i=1,\ldots,n : v_i < u_i$

Definition 2 (superiority): A vector **u** $=(u_1,\ldots,u_n)$ is said to be superior to v $=(v_1,\ldots,v_n)$ *iff* v is inferior to **u**.

Definition 3 (non-inferiority): Vectors **u** $=(u_1,\ldots,u_n)$ and v $=(v_1,\ldots,v_n)$ are said to be non-inferior to one another if v is neither inferior nor superior to **u**.

Let us use few solutions from Figure 2.6 and apply these definitions.

Considering solutions *a* and *b*,

F1(*a*)<F1(*b*) and F2(*b*)<F2(*a*)

So Definition 3 applies here and solutions *a* and *b* are "non-inferior" to each other.

Considering solutions *a* and *h*,

F1(*a*)<F1(*h*) and F2(*a*)=F2(*h*)

So Definition 2 applies here and solutions *a* is "superior" to solution *h*. Similarly solution *b* is "superior" to solution *h*.

Ranking based on non-dominated fronts: We have adapted the ranking scheme like in [10] where each solution is associated with a particular front. First we try to find the solutions which are non-inferior to any other. Solutions which belong to this set belong to the "first front" or "non-dominated front". Solutions *a,b,c,d,e,f* and *g* belong to this front and are hence ranked one. Solutions *h* and *i* dominated by one set of solutions (*a,b,c,d,e,f*) and are hence ranked 2. Solution *j* belongs to the third front being dominated by two sets of solutions and is hence ranked 3.

Niching: Niching is applied in GAs to ensure a diversity of solutions in the population. In multi-objective optimization problems niching can be incorporated in either the "objective space" or in the "decision space." When applied in the objective space, niching helps assure a well-extended and uniformly distributed Pareto front. We have adopted the same niching scheme as in [10], where each front is assessed independently for crowdedness of solutions.

Chapter 3

CONSTRAINT-HANDLING METHODS IN GENETIC ALGORITHMS

3.1 Need For Constraint-handling Methods In GAs

Why do we need to adopt specialized methods for handling constraints using GAs? Taking a numerical example for illustration, suppose we want to maximize a function $f(X) = x_1 + x_2$ where the two variables are defined by $0 \le x_1 \le 1, 0 \le x_2 \le 1$. Under the presence of no additional constraint an optimum value of $f(X) = 2$ can be reached when $x_1 = 1$ and $x_2 = 1$. Assuming there is an additional constraint imposed on these variables described by $g(X) \equiv x_1 - x_2 = 0.5$. Considering a resolution of up to two decimal places in the discrete search space, there are only 50 feasible solutions among 10,000 possible candidates. This implies that feasible region is only 0.5% of the actual parameter space. The best objective function value that can be reached is $f(X) = 1.5$ (for $x_1 = 1$ and $x_2 = 0.5$). The problem complexity can be greatly increased by the number of constraints or the types of constraints.

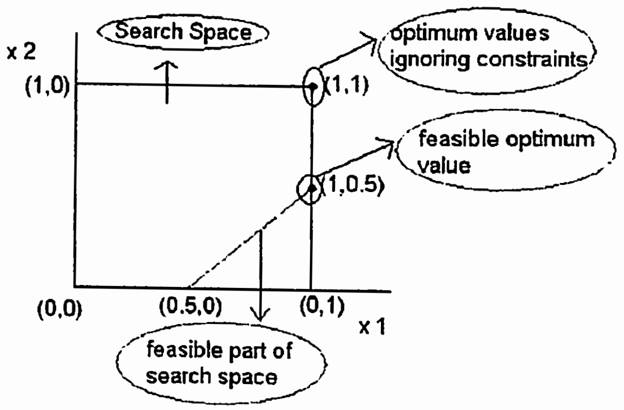

FIGURE 3.1 Search space of two-variable example

Notice that the example that we have illustrated here is almost trivial. Yet, from a search algorithms point of view, this problem presents some difficulty due to the framing of the constraints. So if we do not incorporate any constraint handling scheme, then obviously the algorithm will take a long time to converge to a feasible optimum solution.

Hence we need algorithms whose fitness schemes and selection schemes help in finding feasible optimal solutions as efficiently as possible. Here also lies the challenge, because the objective function and constraints have to be combined to form the fitness function.

3.2 Literature Survey

In this section we shall go through the various methods and test cases that have been proposed for constraint handling using genetic algorithms (GA's). We have broadly categorized our methods into i) penalty functions, ii) methods based on preference of feasible solutions over infeasible ones, iii) methods based on special operators, (iv) methods based on decoders, v) repair algorithms vi) hybrid methods, and vii) methods based on multiobjective optimization and viii) the test cases proposed for constrained optimization. We hope to present the basic idea underlying the design of each of the above constraint-handling methods and refer the reader to [4] for an exhaustive survey.

Before we can go further into explaining constraint-handling schemes using GA's let us define the constraint violation for both equality and inequality constraints,

$$c_j(X) = \begin{cases} \max(0, g_j(X)), & \text{if } 1 \le j \le q \\ \max(0, (\left|h_j(X)\right| - \delta)), & \text{if } q+1 \le j \le m \end{cases} \tag{3.1}$$

Notice that the following definition of the constraint violation would have a similar effect,

$$c_j(X) = \begin{cases} (g_j(X))^2, & \text{if } 1 \le j \le q \\ (\left|h_j(X)\right| - \delta)^2, & \text{if } q+1 \le j \le m \end{cases} \tag{3.2}$$

In either case we have to define the value of threshold (δ) for equality constraints which usually takes a small value of either 0.001 or 0.0001.

3.2.1 Penalty Function Methods: Penalty functions were used in the conventional methods of constrained optimization [12] and were amongst the first methods used to handle constraints using evolutionary algorithms. In these methods a penalty term directly dependent on the constraint violation is added to objective function before evaluating the fitness of the individual. Thus between two infeasible solutions with the same objective function, the one with a lower constraint violation will have a better chance of survival.

In death penalty method the infeasible solutions are rejected for selection into the next generation and this is the easiest way to handle constraints. The death penalty is the simplest constraint handling scheme ever proposed. Since no information regarding the feasibility of each solution is used, this technique usually takes longer to converge to the feasible optimum especially in problems with sparse feasible regions. Also when the initial population consists of no feasible solution, the whole population will have to be rejected and a new one is randomly generated. This technique may work well in problems where the feasible space is convex and covers a large part of the search space.

In static penalty method the penalty is a weighted sum of the constraint violations and the modified objective function value to allocate the fitness to an individual is,

$$obj(X) = f(X) + \sum_{j=1}^{m} r_j . c_j(X) \tag{3.3}$$

$f(X)$ – actual objective function value
r_j – penalty coefficient for each constraint
$c_j(X)$ – constraint violation corresponding to the individual X
$obj(X)$ – modified objective function value after adding penalty

The success of the static penalty method depends on the penalty coefficients chosen for each constraint. This has to be determined carefully based on the difficulty of these constraints and values of the coefficients chosen.

In [17] a dynamic penalty method was proposed where the penalty allotted to each individual would depend on the generation number and a scaling constant C in addition to its constraint violation. The authors note this as an important difference which applies more selective pressure on nearly feasible solution thus making them feasible. However the performance of the algorithm depends on the value of the constants chosen. The modified objective function is defined as,

$$obj(X) = f(X) + (C \times t)^{\alpha} \times SVC(\beta, X) \tag{3.4}$$

$$SVC(\beta, X) = \sum_{i=1}^{q} c_i^{\beta}(X) + \sum_{j=1}^{m-q} c_j(X) \tag{3.5}$$

t – *current generation number*
α, β, C – *constants*

Something to note immediately about the dynamic penalty methods is the difficulty in tuning the many parameters based on which the results are obtained.

While the penalty function methods discussed so far are easy to implement they require some degree of parameter tuning to tailor for each problem. In [29] some guidelines were provided for GA's with penalty methods and some important ones are that (i) penalties which are functions of the distance from feasibility perform better than those which are merely function of the number of violated constraints (ii) for a problem having few constraints, and few full solutions, penalties which are solely function of the number of violated constraints are not likely to find solutions and (iii) the more accurately the penalty is estimated, the better will be the solutions found.

As a method to alleviate the difficulty of deciding on the perfect penalty coefficients, a self-adaptive penalty function was proposed in [5]. The method involves the definition of NFT ("near-feasibility-threshold") which is not only problem specific but also constraint specific. Conceptually the NFT is the threshold distance from the feasible region at which the user can consider the search to be warming up. The prominence of the NFT is that the penalty function will encourage the GA to explore within the feasible region and the NFT-neighborhood of the feasible region and discourage search beyond the threshold. The method makes use of feedback about the quality of solutions in the evolutionary process and uses the information regarding the difficulty levels of each constraint. Hence the penalty method scales itself every generation by varying the NFT associated with each constraint dynamically and thus the effect of the difficulty associated with each constraint and the passage of the number of generations can be fed back to the population via the penalty function. By starting with some initial values for each constraint the algorithm adapts the penalty function based on its associated difficulty levels thus requiring no penalty coefficient definition from the user.

$$NFT = \frac{NFT_0}{1+\Lambda} \tag{3.6}$$

$NFT_0 \Rightarrow$ *upper bound for NFT*
$\Lambda \Rightarrow$ *dynamic search parameter*

From the above definition, we can see that the *NFT* can be made static or dynamic based on the definition of Λ which can be either zero (static penalty function because the *NFT* is constant through out the algorithm) or a function of the number of generations (dynamic penalty function) or a function of the number of solutions in the population that satisfy each particular constraint (a feedback about the difficulty of each constraint is hence used).

The modified objective function for evaluation is defined as,

$$obj(X) = f(X) + (F_{feasible} - F_{all})\sum_{j=1}^{m}\left(\frac{c_i(X)}{NFT_j(t)}\right)^{k_j} \tag{3.7}$$

$F_{feasible}$ – *best known feasible objective function value at generation t*
F_{all} – *best unpenalized objective function value at generation t*
$NFT_j(t)$ – *NFT corresponding to constraint j at generation t*
k_j – *adjusts the severity of the penalty for each constraint*

The advantage of this method is that it may be suitable for problems with active constraints because an exhaustive genetic search takes place at the boundary of the feasibility region due to the definition of *NFT*. But even while the penalty method has self-adaptive features, the upper limit of *NFT* has to be defined. Also there could be problems with the selective pressure if none of the solutions are feasible in which case no penalty will be applied to all solutions and the selection will be based only on the objective function.

In [11], the authors proposed a two-stage penalty function that requires no explicit definition of any parameters. The method was formulated to ensure that slightly infeasible solutions with a low objective function value remain fit. The first penalty stage ensures that the worst of the infeasible solutions has a penalized objective function that is higher or equal to that of the best solution in the population (all other solutions are penalized by a lesser amount depending on their feasibility). The second penalty increases the penalized objective value of the worst of the infeasible solutions to twice the objective value of the best solution.

In [2], the author proposed a self-adaptive penalty function of the form,

$$obj(X) = f(X) + (coef(X) \times w_1 + viol(X) \times w_2) \tag{3.8}$$

$$coef(X) = \sum_{j=1}^{m} c_j(X) \tag{3.9}$$

$$viol(X) \Rightarrow \text{number of violated constraints}$$

Here the author tries to minimize the objective function of the problem, the number of violated constraints and the magnitude of the scalar constraint violations simultaneously.

Two different populations *P1* and *P2* with corresponding sizes *M1* and *M2* was used in the approach. The second of these populations (*P2*) encoded the set of weight combinations (w_1 and w_2) that would be used to compute the fitness of individuals in *P1*. Thus one population was used to evolve the solutions while the other was used to evolve the penalty coefficients. For each individual A_j in *P2* there is an instance in *P1*. Each individual A_j $(1 \leq j \leq M2)$ in *P2* is decoded and the weight combination is used to evolve *P1* for a certain number of generations (*Gmax1*). The fitness of each individual B_k $(1 \leq k \leq M1)$ is computed using (3.8) keeping the penalty factors constant for every individual instance of *P1* corresponding to the individual A_j being processed.

The drawback with this method is the need to define the population sizes (*M1* and *M2*) and the pre specified number of generation (*Gmax1*). At the same time diversity in the solutions may be naturally maintained because of the two populations evolved simultaneously.

Because penalty functions combine the objective function value and the constraint violation value to decide the fitness of each individual, there is a domination relationship between the constraint violation and the objective function in deciding the fitness of the individual. In [30], the authors characterize the problem of choosing the appropriate penalty function coefficient (r_j) for each constraint and describe how it affects the domination between the constraint violation and the objective function optimization in affecting the rank of each individual solution.

TABLE 3.1 Domination relationship based on penalty coefficient

For a given penalty coefficient r_j, let the ranking of λ individuals be

$$\Psi(x_1) \leq \Psi(x_2) \leq \ldots \leq \Psi(x_\lambda)$$

where the individuals are ranked based on the modified objective function

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

Let us examine the adjacent pair i a nd $i+1$ in the ranked order

$$f_i + r_j c_i \leq f_{i+1} + r_j c_{i+1}$$

For a given chioce of $r_j \geq 0$, there are

three different cases which may give rise to the inequality

1) $f_i \leq f_{i+1}$ and $c_i \geq c_{i+1}$: the comparison is dominated by the

objective function as i is ranked better than $i+1$. Since the individual i dominates the individual $i+1$ with respect to both the constraint violation and the objective function, i will always be ranked better than $i+1$ "*irrespective of* r_j."

2) $f_i \geq f_{i+1}$ and $c_i < c_{i+1}$: the comparison is dominated by the penalty function only and the "*value of* r_j *determines the ranks of* i *and* $i+1$."

3) $f_i < f_{i+1}$ and $c_i < c_{i+1}$: the comparison is nondominated and yet i is ranked better than $i+1$. Here again the "*value of* r_j *determines the ranks of* i *and* $i+1$."

To overcome the problem of choosing an optimal r_j the authors propose introducing a probability factor P_f which denotes the probability of the objective function used to allocate rank to the individual. The ranking method incorporated sees to it that feasible solutions are ranked based only on their objective function while the probability factor P_f determines whether objective function or constraint violation should be used to rank infeasible individuals. A P_f value of 0.45 was found to produce very good results. This means that infeasible solutions should be ranked less often based on their objective function value (45%) and more often on their constraint violation value (55%).

While the method produced best results for all of the problems tested, there was one fundamental drawback. The method did not produce feasible solution for all the runs especially for a particular problem from the test case proposed in [20] only 7 out of 30 runs could produce feasible solutions itself. This can be attributed to the selection scheme in which constraint violation does not dominate the objective function even while ranking infeasible solutions. Hence this technique does not support the domination of all infeasible solutions by feasible solutions as advocated in Sub-section 3.2.2.

3.2.2 Preference of feasible solutions over infeasible solutions: In [28], the authors proposed a penalty function method in which feasible solutions would always have higher fitness than infeasible solution. A rank-based selection scheme was used and the rank was 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 solutions, infeasible solutions will be compared based on their constraint violation only and feasible solutions will be compared based on their objective function value only. The fitness allocation is based on the modified objective function,

$$obj(X) = \begin{cases} S(f(X)), & \text{if } \sum_{j=1}^{m} C_j(X) = 0 \\ 1 + r \times \sum_{j=1}^{m} C_j(X), & \text{if } \sum_{j=1}^{m} C_j(X) \neq 0 \end{cases} \tag{3.10}$$

S – a function which maps f into the open interval $(-\infty,1)$.

This method has some interesting properties (i) as long as feasible solutions are not found, the objective function will make no effect on the rank of the individual; (ii) 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 (iii) feasible solutions will be ranked based on their objective function value. The major drawback which we could experience in this method is a lack of diversity operators either explicitly defined or as part of the selection scheme. This could cause difficulties especially in problems with disconnected feasible components in which case the GA may be stuck within one of the feasible components and never get to explore.

The same idea as described above formed the basis of [9] where in selection was based on the following underlying principles (i) Any feasible solution wins over any infeasible solutions; (ii) Two feasible solutions are compared only based on their objective function values; (iii) two infeasible solutions are compared based on the amount of their constraint violations; and (iv) two feasible solutions i and j are compared only if they are within a critical distance d_{ij} otherwise another solution j is checked. The authors also argued that real coded representation was better suited for constrained optimization problems as it affords a greater chance of maintaining feasibility and in addition to used a niching scheme to maintain diversity among feasible solutions and binary tournament selection to make pairwise comparisons. The penalty approach was different in the sense that the coefficient r_g was unity for all constraints and all the constraints were normalized to allot equal importance to each constraint. The modified objective function before fitness allocation is,

$$Obj(X) = \begin{Bmatrix} f(X), & \text{if } X \text{ is feasible} \\ f_{max} + \sum_{j=1}^{m} c_j(X), & \text{otherwise} \end{Bmatrix} \tag{3.11}$$

This method also performed very well on a variety of test problems and niching operator was incorporated to overcome stagnation. This needed the definition of the critical distance d_{ij} and n_f, the number of different solutions tried which all happen to be within d_{ij} before i is chosen as the winner.

3.2.3 Methods based on special operators: In [26] a method for systematically handling linear constraints was proposed. In this algorithm the q linear equalities were eliminated first by solving for the values of q variables and this effectively changed the bounds on the other variables as also simultaneously reducing the search space. GENOCOP tries to locate an initial feasible solution by sampling the search space. If it is not able to find a feasible solution after some trials the user is asked to provide a feasible solution. The genetic operators are closed so that the search does not extend into the infeasible regions. An example of a closed crossover operator used is the arithmetic crossover, where in the crossover of two feasible solutions X and Y always produces feasible children,

$$\begin{aligned} &child1 = aX + (1-a)Y \\ &child2 = (1-a)X + aY \\ &0 \le a \le 1 \end{aligned} \tag{3.12}$$

While the GENOCOP is an effective search technique for problems with convex search spaces but cannot be extended to problems with nonconvex search spaces. Also if the feasibility (ρ) of the search space is low, then feasible solutions will have to be provided by the user.

3.2.4 Methods based on decoders: Decoders work on the principle of "giving instructions" to chromosomes on building feasible solutions. Each decoder imposes a relationship T between a feasible solution and a decoded one. In [20], the authors proposed a homomorphous mapping between the n-dimensional search space and a feasible search space. The authors elaborate on how the mapping is achieved for convex and nonconvex space after providing 3 important criteria that have to be satisfied to obtain a successful mapping i.e., (i) for each feasible solution s there must be a feasible decoded solution d, (ii) each decoded solution d must correspond to a feasible solution s, (iii) all feasible solutions should be represented by the same number of decodings d. Additionally it is reasonable to hope that (i) the transformation T is computationally fast and (ii) it has locality feature such that small changes in decoded solution result in small changes to the solution itself.

This method includes an additional problem-dependent parameter to partition the interval [0, 1] into subintervals of equal length such that the equation of each constraint has, at most one solution in each subinterval. The disadvantage with the homomorphous mapping is that it requires an initial feasible solution and that all infeasible solutions are rejected. Also the locality feature is violated in non-convex search spaces.

3.2.5 Methods based on repair algorithms: Repair algorithms are especially popular in combinatorial optimization techniques where it is relatively easy to repair an infeasible individual. The GENOCOP III [23] also incorporates the original GENOCOP system that handles linear constraints only and extend it by maintaining two separate populations. The first population consists of points which satisfy the linear constraints of the problem: the feasibility of these points is maintained by specialized operators. The second population consists of fully feasible reference points (which the user may have to provide) and these reference points repair any new infeasible points and make them feasible.

The size of the second population of reference points and probability of replacement of the reference solutions were two parameters which had to be set. The GENOCOP-III is a good choice in problems where it is easy to repair an infeasible solution and make it feasible but otherwise the genetic search will be heavily biased.

3.2.6 Hybrid methods: In [18] the authors proposed two hybrid Evolutionary Programming (EP) techniques to solve constrained optimization problems. The first technique is applicable only when the objective function and its gradient are known and consists of two phases: the EP provides the potential solutions for nonconvex optimization to an optimization neural network to generate a precise solution under the assumption that the evolutionary search has generated a solution near the global optimum. In the second method a two phase Evolutionary Program is used and this removes the restriction on knowledge of the gradient. The first phase uses the standard EP while an EP formulation of the augmented lagrangian method is employed in the second phase.

The major drawback with the method is the need to specify so many parameters that is needed for the successful execution of the algorithm.

3.2.7 Methods based on multiobjective optimization techniques: Using a Multiobjective Evolutionary Algorithm (MOEA) based on the Vector Evaluated Genetic Algorithm (VEGA) to solve constrained optimization problems was proposed in [32] where the solutions are first ranked based on non-domination of their constraint violations and then also ranked based on their objective function. A P_{cost} factor selects solutions based on objective function while the others are selected based on constraint violation. The P_{cost} is adjusted depending on the target proportion of feasible solutions in the population. In [3] the author proposed a subpopulation based approach like in VEGA by using $m+1$ subpopulations where m denotes the number of constraints and the first subpopulation is devoted to optimizing the objective function.

The obvious drawback of this technique is the high computational complexity especially as the number of constraints increases. Also there is not enough evidence to validate that treating each constraint independently and ranking them based on Pareto domination is a good approach.

The method differs from [32] in that non-dominated ranking is never employed but the fitness function for each problem is changed so that initially the fitness function for each subpopulation (except the first one which is based on the objective function) depends on the violation of its constraint. If the solution evaluated does not violate the constraint corresponding to the subpopulation but is infeasible, then the subpopulation will minimize the total number of violations. Finally once the solution becomes feasible it will be merged with the first subpopulation and look to minimize the objective function.

While the results produced were satisfactory, the size of each subpopulation remained an open question.

3.2.8 Test Cases: In [24] the author proposed 5 test problems and used them to compare 6 different methods of constraint handling viz. static penalty method, dynamic penalty method, the behavioral memory method, modified form of Genocop II , superiority of feasible to infeasible solutions and death penalty method (rejecting infeasible solutions). In this method an experimental way of estimating the feasibility ratio $\rho = |\mathbb{F} \cap \mathbb{S}| / |\mathbb{S}|$ was implemented by generating 1,000,000 random points from $\mathbb{S}$ and checking whether they belong in $\mathbb{F}$. The test problems were extended to 11 in [24] and this paper concludes that the floating point representation is better than binary representation for constrained optimization problems. Then in [27] a Test Case Generator (TCG) was proposed for constrained optimization problems with six tunable features (i) the number of variables of the problem (ii) number of optima in the search space (iii) number of constraints (iv) connectedness of the feasible search region (v) ratio of feasible to total search space and (vi) ruggedness of the fitness landscape. The TCG is available for free download but no experiments with it other than conducted in the same paper are reported. In [31], the authors noted some deficiencies with TCG like symmetry and equal sized basins of attraction for each subspace and proceeded to propose an advanced version that removes these concerns. Also the TCG-2 in addition incorporates the number of active constraints and different levels of decay of peaks and widths of the peaks as tunable features. The details of the Test Case Generator–2 (TCG-2) are available in [31].

3.3 Need For Further Research In Constrained Optimization Using GAs

There is certainly a need for further research constrained-handling methods for GAs. This has to do with both the way in which algorithms were designed as well as tested. From our analyses of the GAs previously proposed to solve the constrained optimization problem, we notice four common features:

1) lack of elitism

2) require choice of parameters that require a priori knowledge about the problem characteristics

3) lack of an assurance of producing feasible solutions

4) testing of algorithm limited to a handful of problems which do not give a complete insight into algorithm's performance.

We shall see in the following Chapters 4 and 5, how our proposed constraint-handling scheme is designed to overcome these features.

Chapter 4

DESIGN OF THE PROPOSED ALGORITHM

4.1 Design Requirements

1) **Assurance of feasible solutions:** From the real world perspective, it is essential that the GA used for constrained optimization produces feasible optimal solutions for every run of the algorithm. While this may be too much to ask for we would certainly hope that at least the feasibility criteria can be met for every run and that adequate optimization can be achieved.

2) **A Generic Framework:** There are various types of constrained optimization problems that we may encounter in the real world and it would be impractical if the GA used has to be tuned to fit for each problem or if it uses special operators that cannot be implemented in all problem domains. While a generic framework may not be the most efficient for each problem setting, it is the most advisable at the algorithm design stage, while necessary modifications can be made to tailor for particular problems.

4.2 Proposed Constraint-handling Scheme

GA, being a stochastic search technique can offer no guarantee of producing feasible solutions. To address this concern, we have formulated the GA in such a way that finding feasible solutions is the primary objective of the GA. Once a feasible solution is found, then the best one is maintained in the population using the elitist scheme thus assuring that the found feasible solution is not lost. However preferring feasible solutions over infeasible ones could cause the GA to be stuck in one particular feasible component where there are disconnected feasible components (especially if there is a local optimum having the feasible component as the basin of attraction) and the GA may never get to explore the other feasible components containing the global optimum. So exploring the search space guided by both the constraint satisfaction and the objective function optimization will be the secondary objective of the GA. The proposed constraint handling scheme consists of two phases and the algorithm switches smoothly from the first phase to the second based on a simple conditional statement.

Phase One (Constraint Satisfaction Algorithm): In the first phase of the algorithm the objective function is completely ignored and the entire search effort is directed towards finding a single feasible solution. Each individual of the population is ranked based on its constraint violation (minimization) only and fitness is assigned to each individual based on its rank. The elitist strategy is used and the solution with the least constraint violation is copied to the next generation. This phase takes care of the feasibility criteria and provides a usable solution (one that satisfies all constraints). We find this technique to be especially suitable for highly constrained problems wherein finding a feasible solution may be extremely difficult. In such problems it would be worthwhile and efficient to explore the search space based on the constraints alone without taking the objective function into consideration.

Phase Two (Constrained Optimization Algorithm): The algorithm switches to this phase once at least one feasible solution has been identified. This phase is treated as a bi-objective optimization problem where the constraint violations and the objective functions have to be minimized simultaneously in a modified objective space that we call the "objective function – constraint violation space," or *f-v* space for short. We have used a non-dominated sorting like in [10] to rank the individuals. We save the feasible individual with the best objective function in the population as the elitist solution. We also use a niching scheme in the *f-v* space so that sufficient diversity is maintained and the GA will continue to explore. We believe that this MOEA based approach will search to minimize both the objective function and constraint violation simultaneously and guide the algorithm in exploring the region between the constrained and unconstrained optima and the feasible and infeasible parts of the search space. The details of implementation of the algorithm are given below.

4.3 Implementation Details

4.3.1 Scalar Constraint Violation: From the problem formulation we have m constraints and the constraint violation matrix for an individual is an m-dimensional vector. Using a tolerance (δ) of 0.001 for equality constraints the constraint violation of individual i is calculated by,

$$c_{i,j}(X) = \begin{Bmatrix} \max(0, g_j(X)), & \text{if} \quad 1 \le j \le q \\ \max(0, (|h_j(X)| - \delta)), & \text{if} \quad q+1 \le j \le m \end{Bmatrix} \tag{4.1}$$

Each constraint violation is then normalized by dividing it by the largest violation of that constraint in the population. We use normalized constraint violations to treat each constraint equally. First we find the maximum violation of each constraint in the population by using (4.2),

$$cmax(j) = \max_i(|c_{i,j}(X)|), \tag{4.2}$$

$n \Rightarrow popsize$

$c_{i,j}(X) \Rightarrow$ *violation of the* i^{th} *individual on the* j^{th} *constraint*

These maximum constraint violation values are used to normalize each constraint violation. The normalized constraint violations are added together to produce a scalar constraint violation $v(X)$ for that individual which takes a value between 0 and 1.

$$v(X) = \frac{\sum_{j=1}^{m} |c_j(X)| / cmax(j)}{m} \tag{4.3}$$

$|\cdot|$ denotes the magnitude operator

$m -$ number of constraints

4.3.2 Rank Based Fitness Allocation: In both phases of the proposed algorithm we allotted a fitness to each individual based on their rank in the population. In the first phase all the individuals are ranked based on their scalar constraint violation and allotted a fitness value *r(X)*. The fitness values *r(X)* are in a range of 0 to 2 such that the sum of *r(X)* for all individuals equals *n*, the number of individuals in the population. In the second phase the individuals are allotted to different fronts based on non-domination and rank is allotted to each individual based on the front it belongs to.

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

$$fitness(X)=r(X)+d(X) \tag{4.4}$$

Note here that the elitist individual is chosen irrespective of its *fitness* but based only on the conditions for each of the two phases. The pseudo code of the algorithm is given below where g_m denotes the maximum generations used.

TABLE 4.1 Pseudo code of the proposed algorithm

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

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

4.4 Constrained Optimization – Algorithm Design

As discussed before, one of the major challenges for constrained optimization is to search for optimal solutions that are feasible with respect to the constraints. One of the approaches of effectively solving the constrained optimization problem is to treat the *m* constraints as "objectives with goals" and define preference among individuals as described in [14]. However this can lead to an extremely high dimensional objective space as the number of constraints grows. The computational complexity will become unmanageable. Hence we have used a single parameter, the *scalar constraint violation (SCV),* representing normalized net violation of constraints by an individual. To analyze the proposed algorithm further let us consider the two phases individually. Let us define the usage of the following terms,

$\mathbb{F}$ - feasible region, i.e. the domain of the search space $\mathbb{S}$ that is feasible

C_i - the i^{th} disconnected feasible component in the search space $\mathbb{S}$, $i = 1, \ldots, k$

$C_i \cap C_j = \phi,\ i \neq j$ and $i, j = 1, \ldots, k$

$C_1 \cup C_2 \cup \ldots \cup C_k = \mathbb{F}$.

Phase One - Constraint Satisfaction Problem:

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

Argument: If there are m constraints and k (>0) feasible components then a GA with selection based only on constraint violation will find a feasible solution with probability one as $t \to \infty$. This is true because in this case the scalar constraint violation is only a measure of distance from the feasible region. As selection favors minimizing this distance, a feasible solution will be ultimately reached. Since there are k feasible components, the probability of the first feasible solution being found in any one of the feasible components is $1/k$.

Next we begin our analysis of the second phase of the algorithm where the actual optimization takes place. During the design of our fitness scheme we could have chosen either one of the following two schemes: the *preference scheme* or the *non-dominated scheme*. The *preference scheme* based on [26] is defined by,

(i) Any feasible solution is better than any infeasible solution; and

(ii) Among two feasible solutions *i* and *j*, assign greater probability of selection to the solution with the better objective function.

We shall compare this with the *non-dominated scheme* in which

(i) Solutions are ranked based on the non-domination of their constraint violations and objective function values.

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

$$C_i \cap C_j = \phi,\ i \neq j \text{ and } i, j = 1, \ldots, k$$

$$C_1 \cup C_2 \cup \ldots \cup C_k = \mathbb{F}\ .$$

Phase One - Constraint Satisfaction Problem:

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

Argument: If there are m constraints and k (>0) feasible components then a GA with selection based only on constraint violation will find a feasible solution with probability one as $t \to \infty$. This is true because in this case the scalar constraint violation is only a measure of distance from the feasible region. As selection favors minimizing this distance, a feasible solution will be ultimately reached. Since there are k feasible components, the probability of the first feasible solution being found in any one of the feasible components is $1/k$.

Next we begin our analysis of the second phase of the algorithm where the actual optimization takes place. During the design of our fitness scheme we could have chosen either one of the following two schemes: the *preference scheme* or the *non-dominated scheme*. The *preference scheme* based on [26] is defined by,

(i) Any feasible solution is better than any infeasible solution; and

(ii) Among two feasible solutions i and j, assign greater probability of selection to the solution with the better objective function.

We shall compare this with the *non-dominated scheme* in which

(i) Solutions are ranked based on the non-domination of their constraint violations and objective function values.

Analyzing these two selection schemes we try to draw meaningful conclusions about the design of the selection scheme under different problem scenarios. By analyzing the performance of the preference scheme in two different scenarios, we shall be in a better position to decide on the better constraint handling scheme.

Phase Two - Constrained Optimization Problem:

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

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

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

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

In this case our initial feasible solution will belong to this feasible component and the global optimum is also present in this component. Selection based on the *preference scheme* will be more efficient in converging to the global optimum than the *non-dominated scheme* as (i) there is no need to explore looking for other feasible components because it does not exist; (ii) the infeasible solutions (which carry no useful genetic information in this case) are not encouraged in the population and this technique can lead to the global optimum in a lesser number of evaluations.

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

In this case the *preference scheme* may not be efficient in converging to the global optimum. This is because the chances of the feasible initial solution being located in the feasible component with the global optimum is $1/k$ and the magnitude of this value becomes lesser as the number of components increase so there may be a need for the GA to search for solutions in the other components. This induces a need for exploration to find feasible solutions. We analyze the two methods by taking 2 solutions i and j from the population and evaluate the selection scheme that will increase the probability of converging to the global optimum. We also assume in our discussion that the feasible solution with the best objective function is saved as the elitist solution in the population.

1) If solution i is feasible and j is infeasible, we could

(a) assign a greater probability of selection to i irrespective of the objective function values of i and j

(b) check if j has a better objective function value than i and consider both i and j non-dominated if it does. Otherwise assign a greater probability of selection to i.

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

2) Among two feasible solutions i and j, consider i and j non-dominated irrespective of the objective function values. This helps the algorithm explore more as the best feasible

solution is already stored as the elitist solution. Hence by giving both *i* and *j* an equal probability of selection we are giving the algorithm a better chance to explore.

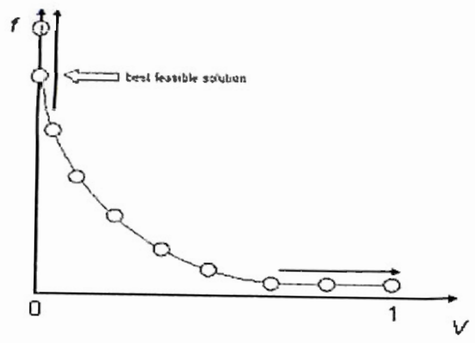

FIGURE 4.1 Schematic of the non-dominated ranking used in proposed GA

Figure 4.1 shows the non-dominated set of solutions in the *f-v* space and all these individuals are ranked one. The niching scheme assigns a different fitness to these solutions based on how crowded they are. So there is a greater selective pressure on solutions at the two corners. Hence the niching scheme tries to extend the Pareto front along the direction of the two arrows. Since the elitist solution is saved unchanged in the population, there is a greater probability of solutions around it. Hence we have indicated this solution in Figure 4.1.

In the next section, we introduce the Test Case Generator-2 (TCG-2) and perform actual experiments using the two selection schemes on the problem scenarios discussed above.

4.5 The GA Used In This Research

Real-valued representation: We have used real-valued representation in our algorithm simply because of the fact that the coding is straight-forward and simple. It is easily possible to create and maintain individuals in a range using real-valued strings and no effort is lost in decoding.

Population Size: In all of our implementations we used a population of only 10 individuals. What does such a small population size mean? Well, first it means much faster computation and second it means less randomness in the algorithm. Let us analyze 2 cases where there is no selection scheme and only mutation is applied all the individuals of the population. Let us assume that there is a fictional number of $\prod$ distinct search space solutions.

Population size=1: In this case the parallel operation of genetic algorithms is completely lost. Since mutation is used, we can assure that at most one random search space solution is generated every generation. Even in the possibility that all these solutions are distinct, it would still require $\prod$ generations to explore the search space thoroughly. As $t \rightarrow \infty$ the GA will converge to the global optimum.

Population size=100: In this case 100 solutions are evaluated in parallel. Assuming the same mutation as discussed in the above case and in the hope that every generated solution is unique, we require $\prod/100$ generations to explore the search space thoroughly. Again, as $t \rightarrow \infty$ the GA will converge to the global optimum.

What is the difference between these two cases: the difference is in "the rate of exploration." We did not want a very large population size because (i) in the first phase of the algorithm we are trying to just find one feasible solution and so we need an "exploitive algorithm," one in which the distance towards feasibility is continually minimized. In the second phase of the algorithm we introduce sufficient diversity in the population by using a niching scheme and hence a large population is not sought for.

We used the "xovmp" and "mutbga" command functions from [39] for implementing the crossover and mutation in our algorithm. We used stochastic universal selection described in Section 2.2.3 for implementing the selection scheme.

Chapter 5

TESTING THE PROPOSED ALGORITHM

5.1 Selection Scheme Comparison Using TCG-2

In this section, we have used the Test Case Generator-2 (TCG-2) to simulate different problem scenarios and perform the experiments. The TCG-2 is an enhanced version of the Test Case Generator (TCG) proposed in [31]. The nine different tunable features of the TCG-2 are,

n- dimensionality of the problem,

m- number of feasible components,

ϱ- feasibility of the search space,

c- complexity of the feasible search space,

a- number of active constraints,

p- number of peaks of the objective function,

σ- width of the peaks,

a- decay of height of the peaks,

d- distance between the different feasible components.

The search space is composed of an *n*-dimensional hypercube with each dimension ranging in the closed interval of [0,1]. The feasible regions of the search space are determined by *m*, ϱ, *c* and *d*. The general idea behind the TCG-2 is to randomly create *m* non-overlapping boxes (or rectangular areas) in the search space. The total occupancy of the *m* feasible components put together is $\rho \times |\mathbb{S}| \times (1-c)$. Considering 2 dimensions, if the complexity *c* is zero then there are *m* feasible components and each one of them is a perfect rectangle. New boxes are attached to existing ones maintaining a minimum distance *d* between the feasible components for the remaining $\rho \times |\mathbb{S}| \times c$ part of the search space. Figure 5.1 shows the feasible components in a two-dimensional search space for n =2, ρ =0.001, c =0.2, m =8, and d = 0.1.

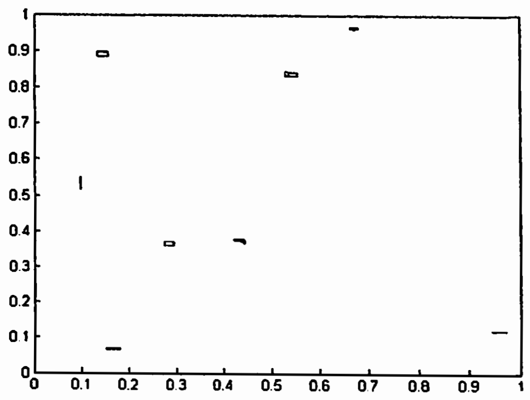

FIGURE 5.1 Feasible components for n =2, ρ =0.001, c =0.2, m =8, and d = 0.1.

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

$$cv(X) = \begin{cases} 0, & \text{if } X \text{ is inside a feasible component} \\ \left| cc_{feasible} - X \right|, & \text{otherwise} \end{cases} \tag{5.1}$$

where $cc_{feasible}$ is the closest center to any feasible component
X denotes the solution vector to be evaluated

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

All centers $\overline{c}_k$ are placed randomly in the search space with the exception of the global optimum that is placed such that there are exactly a active constraints at the global optimum. All peaks heights are evenly distributed between [a, 1] such that the global optimum has the highest peak h_k=1 while the lowest peak has h_k= a. The global optimum is placed either inside the feasible regions (if a = 0) or at the borders (if a>0). Hence the global optimum always satisfies the constraints and has a value of 1. The test scenario chosen is a function TCG-2 ($n,m,\rho,c,a,p,\sigma,a,d$). We have implemented two tests to verify the results from our previous discussions regarding selection schemes for different types of problems. In each of these tests we have defined 10 levels of difficulties and the performance of each algorithm is plotted based on how it treats problems of increasing difficulty.

To perform this test we increase our problem complexity from a problem with 1 feasible component and one optimum to a problem with 10 disconnected feasible components and 10 peaks. All of the other characteristics of the problem such as the feasibility and number of active constraints are kept the same. The test scenario is defined by TCG-2 (2,*m*,0.005,0.2,1,*p*,0.2,0.5,0.1) where *m* and *p* are varied from 1 to 10. We allow the algorithm to run a maximum of 5,000 generations. Also because we know that the global optimum is 1, we stopped the algorithm if the best feasible objective function value reaches 0.999. When implementing both the *preference scheme* and the *non-dominated* scheme, we use niching in the second phase of the both algorithms to maintain diversity. The same elitist scheme is employed in both algorithms. The following Figures 5.2 and 5.3 compare the results from the *preference scheme* method and the *non-dominated scheme.*

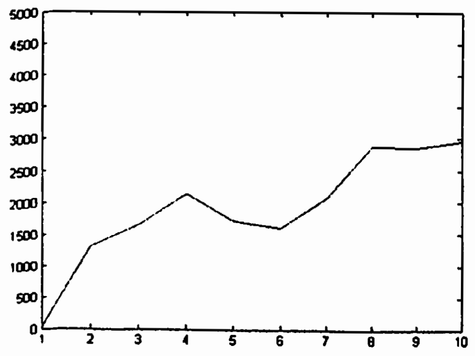

FIGURE 5.2 number of generations used in preference scheme

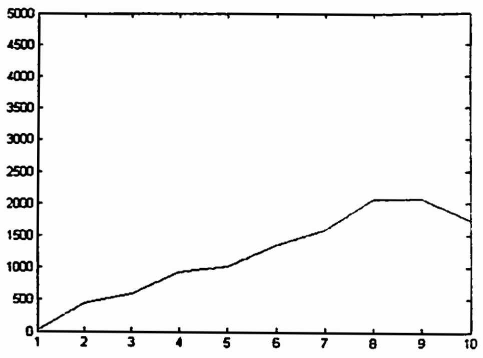

FIGURE 5.3 number of generations used in non-dominated scheme

Starting with the same number of generations required for *m* and *p* value of 1, the non-dominated scheme shows a much better performance when the number of disconnected components and the number of peaks increase. Even though the increase in the number of generations required to solve the problem does not increase linearly with *m* and *p*, we can clearly see that at all stages the non-dominated scheme performs better. Figures 5.4 and 5.5 show the mean objective function values for various stages of the algorithm.

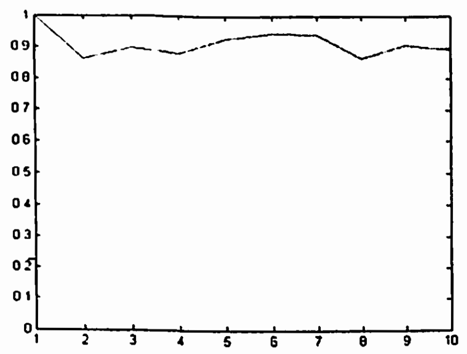

FIGURE 5.4 Mean objective function using a maximum of 5000 generations

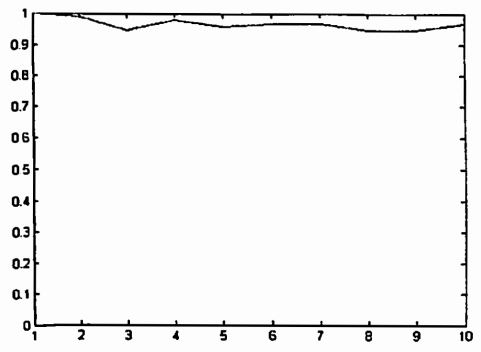

FIGURE 5.5 Mean objective function using a maximum of 5000 generations

Again we notice that better performance has been obtained by the *non-dominated scheme* for all values of m and p greater than 1. By acknowledging that the *non-dominated scheme* achieved these results with a lesser number of generations as shown in Figs. 5.4 and 5.5, we can say with certainty that the non-dominated scheme is more efficient.

In the next section we extend the testing to the eleven test problems form [24] used frequently in literature and provide a fair comparison with state-of-the-art approaches.

5.2 Comparison With Other Algorithms

We implemented the proposed constraint handling scheme for eleven test cases from [24] as shown in Table I using real-coded individuals with probability of mutation of 0.05 and that of crossover of 0.9. For all of the problems we used a population size of only 10 individuals and 10% elitism. We ran our algorithm for 5000 generations in each of 50 runs. The characteristics of these test problems are given in Table 5.1 reproduced from [24].

TABLE 5.1 SUMMARY OF TEST CASES

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

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

The feasibility ratio $\rho = |\mathbb{F} \cap \mathbb{S}| / |\mathbb{S}|$ was determined experimentally in [24] by calculating the percentage of feasible solutions among 1,000,000 randomly generated individuals. For G2 and G3, the value of *k* used was 50 which is different from the 20 used in our experiments. Also because we treat our equality constraints by relaxing them using a threshold value, the values of ϱ would be slightly different for G3, G5 and G11. From Table 5.1 we can see that we have a variety of test functions involving both maximization and minimization problems with different types of objective functions and constraints. We have used our constraint handling scheme without any modifications for solving all of these 11 problems. The results are shown in Table 5.2.

TABLE 5.2 **RESULTS USING PROPOSED CONSTRAINT HANDLING**

Function	*Optimum Value*	*worst*	*best*	*median*	*standard deviation*	*MFG*	*Infeasible Runs*
Min. G1	-15	-11.9999	-14.9999	-14.9997	0.8514	11.24	0
Max. G2	0.803553	0.672169	0.803190	0.755332	0.0327	0	0
Max. G3	1.0	0.7855820	1.00009	0.94899	0.0489	31.68	0
Min. G4	-30665.5	-30651.9595	-30665.5312	-30663.3642	3.3103	0	0
Min. G5	5126.4981	6112.2231	5126.5096	5170.5294	341.2248	1807.82	0
Min.G6	-6961.8	-6954.3186	-6961.1785	-6959.5683	1.2691	289.52	0
Min. G7	24.306	35.881930	24.410977	26.735666	2.6139	53.22	0
Max. G8	0.095825	0.095825	0.095825	0.095825	0	9.28	0
Min. G9	680.63	684.131429	680.762228	681.706290	0.7443	5.84	0
Min.G10	7049.33	12097.4078	7060.55288	7723.16672	798.68	99.86	0
Min. G11	0.75	0.8094	0.7490	0.7493	0.0093	13.32	0

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

One of the first things to be noted is that all of the 50 runs produced feasible solutions for all of the test problems and this assures that in a "real-world scenario" we produced usable solutions in every run of the algorithm. This is definitely attributable to the first phase of our algorithm which treats the constrained optimization problem as a constraint satisfaction problem. In problems G2 and G4 a feasible solution was always found in the random initial population itself and this certainly implies that the feasible space occupies a major part of the search space in these problems. Even though G1 contained 9 constraints, they were all linear and hence the feasible space is convex. Thus by minimizing the distance to the feasible regions the constraints could be satisfied effectively and a feasible solution is found early in the search process. G3 has only one constraint and it was relaxed a little by using a threshold value of 0.001 to help find feasible solutions. Finding feasible solutions was most difficult in G5 where the combination of nonlinear equalities and linear inequalities caused some challenge in locating feasible solutions. Nonlinear inequalities again caused some delay in finding feasible solutions for G8 while a combination of linear inequalities and nonlinear inequalities caused some delay in finding feasible solutions for G7 and G10.

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

1) Nonlinear constraints in general induce more difficulty in finding feasible solutions than linear constraints. This can be understood by acknowledging that GA's are stochastic search techniques that work on reinforcement learning based on the fitness values. This fitness value is a trustable indicator of how far each solution is from the feasible region when there is a linear mapping between the decision variables and the constraints. When the mapping is nonlinear then the distance between slightly infeasible solutions and completely feasible solutions in the decision space may be disproportionate to the differences in the constraint violation values. Hence the one step towards feasibility in the constraint space may involve a tedious search in the decision space.

2) The feasibility ratio and the type of constraints combine together to define the difficulty of the constraint satisfaction problem. We know from Table 5.1 that G2 and G4 have a large ϱ value and a feasible solution was found in the random initial population on all the 50 runs in spite of the fact that nonlinear constraints were present in both problems. At the same time even for problems with low ϱ like G1, feasible solutions could be easily found because the constraints were all linear. G5 involved a combination of nonlinear constraints and very low ϱ and this probably caused the difficulty in finding feasible solutions. Also G6, G9, and G10 which required relatively more generations to find feasible solutions involve a combination of low ϱ and nonlinear constraints.

In the results for the best values found by the algorithm from Table 5.3, we see that the algorithm has produced results extremely close to the optimum value known for all of the 11 test problems. For G1 even though the value of -15.0 could not be reached accurately, the algorithm consistently produced -14.9999 as the optimal value. In G2 again the optimal value of 0.8031 is fairly close to the optimum value of 0.8035. Hitting the optimum in G3 the algorithm produced a better than optimum result in G4. In G5, G6, G7 and G9 the best results produced by the algorithm differ in decimal places from the optimum value. G8 was a very easy problem and the optimum results were obtained for all 50 runs. In G10 again the algorithm having produced 7060.55 was quite close to the 7049.33 optimum which apparently no GA has reached as seen from Table 5.4. The best value for G11 is only better than the 0.75 optimum because of the tolerance in the equality constraint used. Also note that the standard deviation over 50 runs for all the problems other than G5 and G10 is extremely small and the median is very near the best values obtained. This implies that the algorithm is robust in obtaining consistent results. As the algorithm has the added attribute of producing feasible solutions for every run of the algorithm, we would need very few runs with this algorithm when working on a "real-world problem". Table 5.3 reproduced from [11] compares the best results obtained from the other algorithm in literature to those obtained with the proposed constraint handling scheme in the last column.

TABLE 5.3 COMPARISON OF BEST RESULTS

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

We can see that the algorithm has performed very well for all of the test problem reaching or obtaining values extremely near the global optimum. In fact from the table it is obvious that stochastic ranking scheme proposed in [29] has produced the best results known so far for all of the test problems. But a downside to that method as pointed out in [11] is that for G10 only 6 runs out of 30 produced feasible solutions. In [11], 17 runs out of 20 produced feasible solutions while our algorithm produced feasible solutions in all of the 50 runs. We in a way have tackled the same problem of domination between the objective function and constraint violation in assigning the fitness to the individual brought out in [11], but have solved it using non-dominated ranking as opposed to using a probability factor P_f [29]. Overall we believe that the results for these 11 problems which is a variety by itself have helped substantiate our claim that the proposed constraint handling scheme is generic and reliable.

Chapter 6

APPLICATION TO THE ECONOMIC DISPATCH PROBLEM

6.1 Introduction

The *economic dispatch problem* describes how the real power of each controlled generating unit in an area is selected to meet the load demand and to minimize the total operating costs while satisfying the operating constraints involved. The formulation of the economic dispatch problem is dependent on the various factors considered. In this chapter, we deal with the economic dispatch problem considering the ramp rate limits and the prohibited zones, which are essential in the actual operation.

Lambda-iteration [40] is the most commonly used method to tackle the economic dispatch problem. However the lambda-iteration method demands a continuous problem formulation and hence cannot be directly applied to the economic dispatch problem with discontinuous prohibited zones [40]. To overcome this, a genetic algorithm (GA) based method was proposed in [40]. This method uses the concept of equal-incremental costs and uses the GA to search for the optimal value of the incremental cost. A downside to this method is that the prohibited zone is checked for after running the algorithm and appropriate correction is made. This can lead to sub-optimal results. Also the incremental cost has to be found by differentiating the cost function of the generating units, and so the cost function has to be continuous and differentiable. While this is a worthwhile assumption, this does impair the power of the GA because the GA does not require continuity or differentiability of the cost function. The authors, in the discussion have hinted a need for further research in this direction and in handling the prohibited zones. Even while a variety of constraints were considered in the problem formulation, a real-time or dynamic economic dispatch was not implemented in any of the test systems.

The work discussed in this chapter hopes to build on [40] by using an advanced GA well crafted to handle constraints. Hence the prohibited zones of the generating units are treated as hard constraints, which have to be satisfied before a feasible solution can be reached. We have not used the equal incremental cost method because that hampers the generality of the algorithm. The equal incremental cost method is unsuitable in problems where the cost function of the generating units is not monotonically increasing. We have also solved the dynamic economic dispatch problem by building on the example given in [40]. We found it important to test the GA on the dynamic economic dispatch problem because in the real-world operation, the load on the power systems varies from one interval to another. Also dynamic economic dispatch helps us to test the algorithms ability in overcoming the ramp-rate limit constraints.

In Section 6.2 we give the problem formulation considering only one interval of operation. The test system is elaborated in Section 6.3. The implementation of the proposed algorithm is discussed in section 6.4. In Section 6.5 we compare the results between the proposed algorithm and [40]. In Section 6.6 we further extend the testing of the system from one interval to six intervals and thoroughly evaluate the proposed algorithm's performance. We finally conclude the chapter in Section 6.7.

6.2 Problem Formulation Considering One Interval Of Operation

In this section we describe the problem formulation of a power system over one interval of operation. Here the load demand (in MW) for one interval has to be met while satisfying the operating constraints. So we can assume that one interval is the time during which the load on the system remains unchanged. Thus one interval can imply different time-lengths depending upon the operation and load variation on the system. The following sub-sections describe the problem in detail as formulated in [40].

6.2.1 Objective Function: The objective is to minimize the generation costs involved while meeting the operational constraints on the system. The objective function is defined as,

$$Minimize\ F = \sum_{i=1}^{n} f_i(P_i) \tag{6.1}$$

where,
F : total generating cost of the system
P_i : power generation of unit i
$f_i(P_i)$: generation const for P_i

6.2.2 Constraints incorporating the power balance equation: The most important aspect of economic dispatch is to meet the load demand. This is structured as the power-balance equation where the transmission losses are also considered.

$$\sum_{i=1}^{n} P_i = P_D + P_{loss} \tag{6.2}$$

where,
P_D : system load demand
P_{loss} : system transmission losses involved

The network losses, which are taken into account, are calculated as,

$$P_{loss} = \sum_{i=1}^{n}\sum_{j=1}^{n} P_i B_{ij} P_j \qquad (6.3)$$

where,

B_{ij} : loss coefficients

6.2.3 Constraints incorporating upper and lower generating limits: The generating units can only be operated within a range of output MW and this forms the second set of constraints on the system.

$$\underline{P_i} \le P_i \le \overline{P_i} \qquad (6.4)$$

where,

$\underline{P_i}$: minimum generation of unit *i*

$\overline{P_i}$: maximum generation of unit *i*

6.2.4 Constraints incorporating the ramp rate limits: The third set of constraints is related to the real-time operational aspect of power systems. Practically it is not possible to adjust the unit's output generation instantaneously. The operating range of all on-line units is restricted by their ramp-rate limits. The following inequality constraints determine the acceptable range of values that can be achieved based upon the previous output MW of the generating unit.

1) if generation increases

$$P_i - P_i^o \le UR_i \qquad (6.5)$$

2) if generation decreases

$$P_i^o - P_i \le DR_i \qquad (6.6)$$

where,

P_i^o : power generation of unit i at previous hour

UR_i : ramp - rate limit of unit i as generation increases

DR_i : ramp - rate limit of unit i as generation decreases

6.2.5 Constraints incorporating the prohibited zone of operation: The final set of constraints involves the prohibited operating zones for each of the generating units. In practice, the best economy is achieved by avoiding the operation in the prohibited zone.

$$P_i \leq P_{pz}^- \text{ or } P_i \geq P_{pz}^+ \tag{6.7}$$

where,

P_{pz}^-, P_{pz}^+ : bounds of a prohibided zone

6.3 Test System

The test system used in this study was taken from [40] and is briefly described here. There are three controlled generating units in the power system, which is required to meet a load demand of 300 MW. The cost function is given by the following quadratic equation,

$$f_i(P_i) = a_i P_i^2 + b_i P_i + c_i \tag{6.8}$$

where,

a_i , b_i and c_i are constants.

The data corresponding to the generating units have been directly taken from [40] and are listed here for convenience.

TABLE 6.1 Generating units capacity and coefficients

$\underline{P}_i$	$\overline{P}_i$	$\overline{P}_i$	$a_i(\$/MW^2)$	$b_i(\$/MW)$	$c_i(\$)$
50	50	250	0.00525	8.663	328.13
2	5	150	0.00609	10.04	136.91
3	15	100	0.00592	9.76	59.16

TABLE 6.2 Generating units ramp rate limits and prohibited zones

Unit	P_i^o	$UR_i(MW/h)$	$DR_i(MW/h)$	*Prohibited Zones (MW)*
1	215	55.0	95.0	[105,117][165,177]
2	72.0	55.0	78.0	[50,60][92,102]
3	98.0	45.0	64.0	[25,32][60,67]

The matrix given below specifies the loss formula coefficients [40],

$$B_{ij} = \begin{bmatrix} 0.000136 & 0.0000157 & 0.000184 \\ 0.0000175 & 0.000154 & 0.000283 \\ 0.000184 & 0.000283 & 0.00161 \end{bmatrix}$$

To loosen the power balance constraints a little, we have allowed a maximum of 1.001 times the load. So our power balance constraint for this problem is given by,

$$300 \le \sum_{i=1}^{n} P_i \le 300.3 \tag{6.9}$$

6.4 Algorithm Implementation On The Test System

We used a real-coding of the three generator outputs in the proposed algorithm. We have used a population size of 10 and a maximum of 5000 generations. We have used the algorithm described in detail in Chapter 4. Because this is a real-world implementation we have used an early stopping criterion to terminate the GA. According to this, if there is no change in the best solution for 500 generations, then that run is terminated. The GA was run 50 times and the results are discussed below.

6.5 Results

Table 6.3 details the results of the proposed algorithm over 50 runs. We can see that the algorithm produced very consistent results by comparing the best, median and worst objective function values acheived. Also the standard deviation is extremely small. An important factor to notice is that all the runs produced feasible solutions. This combination of consistent results and feasible solutions make it unnecessary to run many trials of the algorithm in a real-world situation. The mean number of generations to produce feasible solutions is 14. This means a very fast implementation since we are using a population size of only 10 individuals. So a feasible solution even while considering the prohibited zone and ramp rate limits is found within 140 objective function evaluations. The last column gives the mean number of total generations used. The early stopping criteria saves the algorithm from running unnecessary generations after the best solution has remained unchanged for 500 generations.

TABLE 6.3 Summary of results over 50 independent trails

Best objective	Median objective	Worst objective	Standard deviation	FR (%)	FG	TG
3634.966	3652.503	3743.645	29.658	100	14	2434.387

FR(%): Percentage of the runs that produced feasible results
FG: Mean feasible generation number
TG: Mean total generations used

We compare the best results of the proposed algorithm with those from [40] in Table 6.4. Because multiple runs are not reported in [40], we will limit our comparison to the best solution found. As we can see, the proposed algorithm has obtained a major saving in the total fuel cost. This is primarily because the proposed algorithm was able to find a solution that incurs very low transmission losses. Since the algorithm was implemented on two different types of computers, the actual execution time of the algorithm is not compared.

TABLE 6.4 Comparison of best results

	P1(MW)	*P2(MW)*	*P3(MW)*	$P_{loss}(MW)$	F($)
Algorithm used in [40]	194.265	50.0	79.625	24.011	3737.166
Proposed algorithm	199.274	79.423	34.143	12.841	3634.966

P1, P2, P3 (MW): Power output from the generating units 1,2 and 3 respectively
$P_{loss}(MW)$: Total power lost due to transmission losses
F($): Total generating cost

6.6 Extended Testing Covering Six Intervals

To test the performance of the proposed algorithm in a real-world scenario we extended the testing on the system described in section 6.3 to 6 intervals with different load demands. Again an upper limit of 1.001 times the load was used in satisfying the power balance equations. The load demand on the 6 intervals is given in Table 6.5.

TABLE 6.5 Load demand on test system over 6 intervals

	Interval 1	*Interval 2*	*Interval 3*	*Interval 4*	*Interval 5*	*Interval 6*
Load Demand (MW)	300	270	322.5	345	375	337.5

The algorithm was run for 50 independent trails to evaluate its performance. In the real-world scenario, the algorithm would be run only once for each interval. The best solution in the 50 independent runs is given in Table 6.6.

TABLE 6.6 Best results obtained after 50 runs

	P1(MW)	*P2(MW)*	*P3(MW)*	$P_{loss}(MW)$	$\sum_{i=1}^{n} P_i(MW)$	*F ($)*
Interval 1	194.512	81.373	37.511	13.397	313.397	3639.65
Interval 2	186.433	76.744	15.000	8.177	278.177	3275.86
Interval 3	205.275	113.197	15	10.973	333.473	3886.006
Interval 4	219.799	122.647	15	12.447	357.447	4152.685
Interval 5	249.913	124.454	15.505	14.873	389.873	4513.704
Interval 6	225.512	109.132	15.000	12.145	349.645	4060.768
					Total Cost($\sum F$):	23528.682

The table below summarizes the performance of the algorithm over the 50 trials. We notice very consistent results with respect to the objective function values obtained. Also the standard deviation is very low considering the magnitude of the objective function values. These consistent results make multiple runs of the algorithm unnecessary. The attribute of the algorithm is very useful in the real-world implementation.

TABLE 6.7 Summary of results

Best objective	Median objective	Worst objective	Standard deviation
23528.682	23603.131	24026.944	99.865

Table 6.8 summarizes the mean number of generations to find a feasible solution and the total number of generations used in the 6 intervals. The mean feasible generations are very small in all the intervals. Also all the runs produced feasible solutions. Hence we can say that the proposed algorithm is fast and reliable in handling the dynamic economic dispatch problem.

TABLE 6.8 Mean Generations used

	Interval 1	*Interval 2*	*Interval 3*	*Interval 4*	*Interval 5*	*Interval 6*
FG	13.66	14.80	16.74	16.42	12.44	14.26
TG	1774.0	1866.5	1958.2	1721.9	1484.8	1560.2
FR(%)	100	100	100	100	100	100

6.7 Summary

We ran the proposed algorithm to implement the economic dispatch in a 3-unit system. The algorithm produced better cost values than [41] on the test problem chosen. Extended testing was carried out over 6 intervals to evaluate the performance of the algorithm in a real-time implementation. All runs of the algorithm produced feasible solutions and reached very close to the optimal value. This proves that the proposed algorithm is suitable for real-world implementation wherein only a single run is needed.

Chapter 7

CONCLUSIONS

We have implemented a two-phase Genetic Algorithm to solve the constrained optimization problem. This algorithm has the advantage of being problem independent and does not rely on any parameter tuning. The proposed constraint handling scheme was also tested on TCG-2. We assigned various levels of difficulty based on the number of disconnected components and peaks in the decision space. We provided the rationale behind using a non-dominated ranking scheme for selecting individuals in a modified objective space of the objective function plotted versus the constraint violation. This in addition to the elitist scheme helps provide the delicate balance between exploration and exploitation of the algorithm. We made a fair comparison with a preference scheme of selection wherein feasible solutions are always preferred over infeasible ones. We were able to show the more efficient performance of our algorithm as the number of disconnected feasible components and the number of peaks increase. We then extended the testing to the eleven test problems used often in literature. Apart from finding optimal solutions that are extremely close to the optimum values, our algorithm also found feasible solutions for every run of the algorithm.

We used the proposed algorithm to solve a real-world application known as the economic dispatch problem. Our results prove that the algorithm is suited for real-time implementation where only a single run is desired.

REFERENCE

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

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

[3] C.A.C. Coello, "Treating constraints as objectives for single-objective evolutionary computation," *Eng. Opt.*, vol. 32, pp. 275-308, 2000.

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

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

[6] J.C. Culberson, "On the futility of blind search: an Algorithmic view of "No Free Lunch"," *Evol. Comput.*, vol. 6, 1998.

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

[8] K. Deb and S. Agrawal, "A niched-penalty approach for constraint handling in genetic algorithms," *Proc. Int. Conf. on Artificial Neural Network and Genetic Algorithms, Portoroz,* pp. 235-243, 1999.

[9] K. Deb, "An efficient constraint handling method for genetic algorithms," *Comput. Methods Appl. Mech. and Engrg.*, vol. 186, pp. 311-338, 2000.

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

[11] R. Farmani and J.A. Wright, "Self-adaptive fitness formulation for constrained optimization," *IEEE Trans. Evol. Comput.*, vol. 7, pp. 445-455, 2003.

[12] R. Fletcher, "Practical methods of optimization", second edition, John Wiley and sons, 1990.

[13] C. M. Fonseca and P.J. Fleming, "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization", *Proc. 5th Int. Conf. Genetic Algorithms*, pp. 416-423, 1993.

[14] C. M. Fonseca and P.J. Fleming, "Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithm-Part I: A Unified Formulation", *IEEE Trans Sys, Man and Cybern – A*, vol. 28, no. 1, pp.26-37, 1998.

[15] M. Gen and R. Cheng, "A survey of penalty techniques in genetic algorithms", *Proc. of the IEEE Int. Conf. on Evol. Comput.*, pp. 804-809, 1996.

[16] D. E. Goldberg, "Genetic Algorithms in search, optimization and machine learning," Addision Wesly Publishing Company, January, 1989.

[17] J.A. Joines and C.R. Houck, "On the Use of Non-Stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems with GA's," in *Proc. IEEE Conf. Evo. Comput.* , vol. 2, pp. 579-584, 1994.

[18] J.-H. Kim and H. Myung, "Evolutionary Programming Techniques for Constrained Optimization Problems", *IEEE Trans. Evol. Comput.*, vol. 1, no. 2, pp. 129-140, 284-294, 1997.

[19] T. Kiyota, Y. Tsuji and E. Kondo, "Unsatisfying Functions and Multiobjective Fuzzy Satisfying Design using Genetic Algorithms," *IEEE Trans. Syst., Man, Cybern. B*, vol. 33, pp. 889-897, 2003.

[20] S. Koziel and Z. Michalewicz, "Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization," *Evol. Comput.*, vol. 7, no. 1, pp. 19-44, 1999.

[21] A. Kuri and J. Gutierrez, "Penalty function methods for constrained optimization with genetic algorithms : a statistical analysis," *Proc. of the Second Mexican International Conference on Artificial Intelligence.*, pp . 108-117, 2002.

[22] Z. Michalewicz and N. F. Attia, "Evolutionary optimization of constrained problems," *Proc. 3rd Annual Conference on Evolutionary Programming*, pp 98-108, 1994.

[23] Z. Michalewicz and G. Nazhiyath, "GENOCOP III: a co-evolutionary algorithm for numerical optimization problems with nonlinear constraints," *Proc. of 2nd IEEE Int'l Conf. on Evol. Comput.*, pp. 647--651, 1995.

[24] Z. Michalewicz, "Genetic algorithms, numerical optimization and constraints." *Proc. 6th Int. Con. on Genetic Algorithms*, pp. 151-158, 1995.

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

[26] Z. Michalewicz and C.Z. Janikow, "GENOCOP: a genetic algorithm for numerical optimization problems with linear constraints," *Commun. ACM*, 1996.

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

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

[29] J.T.Richardson, M. R. Palmar, G. Liepus, and M. Hillard, "Some guidelines for genetic algorithms with penalty functions," *Proc. 3rd Int. Conf. Genetic Algorithms*, vol. 3, pp. 191-197, 1989.

[30] T. P. Runarsson and X. Yao, " Stochastic Ranking for Constrained Evolutionary Optimization," *IEEE Trans. Evol. Comput.*, vol. 4, no. 3, 2000.

[31] M. Schmidt, Z. Michalewicz, "Test-Case Generator *TCG-2* for Nonlinear Parameter Optimization," *Proc. Parallel Problem Solving from Nature*, pp. 539-548, 2000.

[32] P.D. Surry, N.J. Radcliffe, and I. D. Boyd, " A multi-objective approach to constrained optimization of Gas supply networks: The COMOGA method," *Evolutionary Computing. AISB Workshop. Selected Papers*, pp. 166-180, 1995.

[33] M. Schoenauer and S. Xanthakis, " Constrained GA optimization," *Proc 5th Int. Conf. Genetic Algorithms*, pp. 573-580, 1993.

[34] J. I. van Hemert, T.Bäck, "Measuring the Searched Space to Guide Efficiency: The Principle and Evidence on Constraint Satisfaction," *7th Int. Conf. Parallel Problem Solving from Nature*, pp. 23-32, 2002.

[35] D. Whitley, "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best," *Proc. 3rd Int. Conf. Genetic Algorithms*, pp. 116-123, 1989.

[36] D.H. Wolpert and W.G. Macready, " No Free Lunch Theorems for Optimization," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 67-82, 1997.

[37] "Genetic algorithms in engineering systems," eds. A.M.S. Zalzala and P.J.Fleming, *The Institution of Electrical Engineers*, 1997.

[38] http://www-fp.mcs.anl.gov/otc/Guide/OptWeb/opt.html, Online Source.

[39] GA toolbox available for free download from http://www.shef.ac.uk/~gaipp/ga-toolbox/

[40] P. Chen and H. Chang, "Large-scale economic dispatch by genetic algorithms," IEEE Trans. on Power Systems, Vol. 10, No. 4, pp.1919-1926, Nov. 1995.

[41] K. P. Wong and J. Yuryevich, "Evolutionary-programming-based algorithm for environmentally-constrained economic dispatch," Vol. 13, No. 2, May 1998.

[42] J. W. Lamont and E. V. Obessis, "Emission dispatch models and algorithms for the 1990's," Vol. 10, No. 2, May 1995.

VITA (1)

Sangameswar Venkatraman

Candidate for the Degree of

Master of Science

Thesis: A GENETIC ALGORITHM DESIGN FOR CONSTRAINED OPTIMIZATION

Major Field: Electrical Engineering

Biographical:

Education: Received Bachelor of Engineering degree from the University of Madras in May 2002. Completed the requirements for the Master of Science degree with a major in Electrical and Computer Engineering at Oklahoma State University in July 2004.

Professional Membership: Institute of Electrical and Electronics Engineers.