EVOLUTIONARY METHODS FOR SOLVING MULTI-OBJECTIVE OPTIMIZATION PROBLEM

BY

GIRUM WOLDEGEBRIEL

DEPARTMENT OF MATHEMATICS
ADDIS ABABA UNIVERSITY

ADVISOR: SEMU MITIKU (PhD)

GRADUATE PROJECT SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER SCIENCE IN MATHEMATICS

ADDIS ABABA, ETHIOPIA
JUNE 2012

Acknowledgement

First of all, I would like to thanks my almighty God. Next I would like to express my sincere gratitude and appreciation to my advisor Dr. Semu Mitiku for providing professional suggestion and constructive comment in preparing this paper. My special thanks go to my parents and families, whose inspiration and support have always been with me throughout my study. Last but not least to my relatives for their encouragement and support and to Areka city administration for their support.

Abstract

Evolutionary methods are characterized as a set of solution based algorithms to solve multi-objective optimization problems. Evolutionary algorithms have a potential of finding multiple Pareto optimal solution in a single simulation run. In this report we have considered non-dominated sorting genetic algorithm to solve multi objective optimization problem. We have suggested non-dominated sorting genetic algorithm–II for minimization of the objectives. Non-dominated sorting genetic algorithm–II is fast elitist search algorithm which is based on non-domination rank. Non- domination rank provides chance to the population to be chosen to become parent of the next generation. Selection is based on crowded comparison operator to pick population to variation operator.

Table of contents

Contents **pages**

List of tables and Figure

Table's **pages**

Figure **pages**

Introduction

The world have been confronted with multiple criteria decision making problems i.e. in the records of human culture, all important political, economical and cultural events have involved multiple criteria in their evolution. In contrast there are several techniques available for single objective optimization problem; relatively few techniques have been developed for multi objective optimization problem.

In single objective optimization, the search space is often well defined. As soon as there are several, possibly contradicting, objectives to be optimized simultaneously, there is no longer single optimal solution but rather a whole set of possible solution equivalent quality. When we try to optimize several objectives at the same time the search space also becomes partially ordered. To obtain the optimal solution there will be a set of tradeoffs between conflicting objectives.

From different methods for solving multi objective optimization problem on this paper we focus on evolutionary methods which are apart generating solution and evolutionary computation. The evolutionary algorithms are population based approach in which iteration are performed on a set of solution (populations). And more than one solution is generated at each iteration. Evolutionary algorithms are now developing algorithms to solve real-life multi objective problems [10]. The evolutionary algorithms do not require derivative information. Some evolutionary algorithms use elitism, which keeps best solutions from being lost. Such operation makes sure that an algorithm has guarantee for solving optimization problem.

In this project, we discuss an algorithm for solving multi objective optimization problem mainly on evolutionary, whose initial population encodes real values. This report is organized in three units. In unit one, we discuss solution concepts of multi objective optimization problem and formulation of multi objective optimization problem, in unit two methods classification based on participation of decision maker during solution finding and algorithmic concepts, in unit three evolutionary methods for solving multi objective optimization problem mainly focusing on non-dominated sorting genetic algorithms, moreover procedure and algorithmic concept and example is included.

UNIT ONE

1.1 problem setting and definitions

A problem to optimize multiple conflicting objective function simultaneously under a given constraints is called multi objective optimization problem

A multi objective optimization problem could be written in the form:-

$$\text{Min}\{f_1(x) = z_1\}$$

$$\text{Min}\{f_2(x) = z_2\}$$

. .

. .

$$\text{Min}\{f_k(x) = z_k\}$$

S. t $x \in X$ where $X \subset \mathbb{R}^n$ and X is the set of feasible Solutions in $\mathbb{R}^n$

$f_i : X \to \mathbb{R}$ f_i $i = 1, \dots, k$ be linear or non linear subject to several equality and inequality constraints. For $x = [x_1, x_2, x_3, \dots x_n]$, the vector of decision variables, our task is to determine the set f of all vectors which satisfy all the constraints, the particular set of values $[x_1^*, x_2^*, x_3^*, \dots, x_n^*]$ and also yields the optimum values for all objective functions[6].

The set of all n - tuples of real numbers denoted by $\mathbb{R}^n$ is called **Euclidean n-space**. Two Euclidean spaces are considered in multi objective optimization problems:

- The n-dimensional space of the decision variables in which each coordinate axis corresponds to a component of vector x.
- The k-dimensional space of the objective functions in which each coordinate axis corresponds to a component vector $f_k(x)$ [6].

Here we conceder the term solution as a variable vector and point as the corresponding objective vector.

1.2 concepts and definition

Definition: - A point $x \epsilon \mathbb{R}^n$ is convex combination of points $y, z \epsilon \ \mathbb{R}^n$ if $x = \lambda y + (1-\lambda)z$, $\lambda \epsilon [0,1]$

Definition :- (convex set) A subset S of $\mathbb{R}^n$ is said to be convex if

$x\lambda + (1-\lambda)y \epsilon S$ Where $\lambda \epsilon [0,1]$

In other words, S is convex if the convex combination of every pair of points in S lies in S.

Definition :- (Cone) A subset K of $\mathbb{R}^n$ is a cone if $x\lambda \epsilon K$ whenever $x \epsilon K$ and $\lambda > 0$

Definition: - A non empty cone K is called pointed if $x \epsilon K$ and $-x \epsilon K$ that is $K \cup -K = \{0\}$

Example:-

The set $S = \{ x(t) \epsilon [0,1] : x(t) \geq 0$ for all $t \epsilon [0,1]\}$ is a pointed cone.

Preposition 1.1

A cone K is convex if and only if for $x, \ y \in K, (x+y) \in K$

Proof:-

($\Longrightarrow$):- let K be convex cone,

Then for any $x, y \in K$ and $\lambda = 1/2$

$$\Longrightarrow \quad \frac{1}{2} x + (1 - \frac{1}{2}) y \in K$$

$$\Longrightarrow \quad \frac{1}{2} (x+y) \in K$$

$$\Longrightarrow \ x + y \in K$$

($\Longleftarrow$) :- let $x + y \in K$ and $\lambda \in [0,1]$

$\lambda x \in K$ And $(1-\lambda)y \in K$ since K is a cone

$\Rightarrow \quad \lambda x + (1-\lambda)y \in K$

$\Rightarrow \quad k$ Is convex

1.2.1 **Convex functions**

Definition: - Let X be a non-empty convex set in $\mathbb{R}^n$, K be a convex cone in $\mathbb{R}^k$ then a function $f : \mathbb{R}^n \longrightarrow \mathbb{R}^k$ is said to be **K**-convex function,

If $\lambda f(x) + (1-\lambda)f(y) - f(\lambda x + (1-\lambda)y) \in K$ for all x, **y** $\in$ X and for all **λ** $\in [0, 1]$

A function $f : X \to \mathbb{R}$ is said to be convex if $\lambda f(x) + (1-\lambda)f(y) - f(\lambda x + 1-\lambda y \in \mathbb{R}+$ is satisfied for all x, **y** $\in$ X and for all **λ** $\in [0, 1]$

Proposition: -1.2

Let X be a convex set in $\mathbb{R}^n$ $f(x) = (f_1(x), \dots, f_k(x))$ and : $f : \mathbb{R}^n \to \mathbb{R}^k$ is $\mathbb{R}^k_+$ convex if and only if the functions f_i are convex for all ($i = 1, \dots, k$).

Proof:-

(⇒): Let f is $\mathbb{R}^k_+$ is convex and x, **y** $\in$ X and for all **λ** $\in [0, 1]$

$\Rightarrow \lambda f(x) + (1-\lambda)f(y) - f(\lambda x + (1-\lambda)y) \in \mathbb{R}^k_+$

$\Rightarrow \lambda(f_1(x), \dots, f_k(x)) + (1-\lambda) f_1(y), \dots, f_k(y)$
$- [f_1(\lambda x + (1-\lambda)y), f_2(\lambda x + (1-\lambda)y), \dots, f_k(\lambda x + (1-\lambda)y)] \in \mathbb{R}^k_+$

$\Rightarrow (\lambda f_1(x), \dots, \lambda f_k(x)) + (1-\lambda) f_1(y), \dots, (1-\lambda) f_k(y)$
$- [f_1(\lambda x + (1-\lambda)y), f_2(\lambda x + (1-\lambda)y), \dots, f_k(\lambda x + (1-\lambda)y)] \in \mathbb{R}^k_+$

$\Rightarrow [\lambda f_1(x) + (1-\lambda) f_1(y), \dots, \lambda f_k(x) + (1-\lambda) f_k(y)]$
$- [f_1(\lambda x + (1-\lambda)y), f_2(\lambda x + (1-\lambda)y), \dots, f_k(\lambda x + (1-\lambda)y)] \in \mathbb{R}^k_+$

$\Rightarrow \lambda f_1(x) + (1-\lambda) f_1(y) - f_1(\lambda x + (1-\lambda)y) \in \mathbb{R}_+, \dots, \lambda f_k(x) (1-\lambda) f_k(y) -$
$f_k(\lambda x + (1-\lambda)y)] \in \mathbb{R}_+$

$$\Rightarrow f_i \text{ is convex}$$

($\Leftarrow$) : Let f_i is convex for all (i $=$ 1, …, k).

$\Rightarrow \lambda f_i(x) + (1 - \lambda) f_i(y) - [f_i(\lambda x + (1 - \lambda)y)] \in \mathbb{R}_+$ For all (i $=$ 1, …, k) and for all $\lambda \in [0, 1]$

$$\Rightarrow [\lambda f_1(x) + (1 - \lambda) f_1(y), \dots, \lambda f_k(x) + (1 - \lambda) f_k(y)] - [f_1(\lambda x + (1 - \lambda)y), f_2(\lambda x + (1 - \lambda)y), \dots, f_k(\lambda x + (1 - \lambda)y)] \in \mathbb{R}_+^k$$

$$\Rightarrow (\lambda f_1(x), \dots, \lambda f_k(x)) + (1 - \lambda) f_1(y), \dots, (1 - \lambda) f_k(y) - [f_1(\lambda x + (1 - \lambda)y), f_2(\lambda x + (1 - \lambda)y), \dots, f_k(\lambda x + (1 - \lambda)y)] \in \mathbb{R}_+^k$$

$$\Rightarrow \lambda(f_1(x), \dots, f_k(x)) + (1 - \lambda)(f_1(y), \dots, f_k(y)) - [f_1(\lambda x + (1 - \lambda)y), f_2(\lambda x + (1 - \lambda)y), \dots, f_k(\lambda x + (1 - \lambda)y)] \in \mathbb{R}_+^k$$

$$\Rightarrow \lambda f(x) + (1 - \lambda) f(y) - f(\lambda x + (1 - \lambda)y) \in \mathbb{R}_+^k$$

$\Rightarrow$ f Is $\mathbb{R}_+^k$- convex function.

Moreover, a function f is said to be strictly convex if for any

$$x, y \in X, x \neq y \text{ and for all } \lambda \in (0, 1),$$

$$f(\lambda x + (1 - \lambda)y) < \lambda f(x) + (1 - \lambda) f(y)$$

Definition: - A binary relation R on some set Z is defined as a subset of $Z \times Z$. We write $z^1 R z^2 \Leftrightarrow (z^1, z^2) \in R$.

The basic binary relations are:-

1. Strict preference ($\prec$) i.e. $z^1 \prec z^2$ for $z^1, z^2 \in Z$ that means z^1 is preferred to z^2.
2. Indifference ($\sim$) i.e. $z^1 \sim z^2$ means z^1 is indifference to z^2 or not $z^1 \prec z^2$ or not $z^2 \prec z^1$
3. Preference-indifference ($\preccurlyeq$)) i.e. $z^1 \preccurlyeq z^2$ as $z^1 \prec z^2$ or $z^1 \sim z^2$

Thus the binary relation R on a set Z for $z^1, z^2 \in Z$ are listed below

a) Reflexive if $z^1 R z^1$ for every $z^1 \in Z$

b) I reflexive if not $z^1 \mathrm{R} z^1$ for every $z^1 \in Z$

c) Symmetric if $z^1 R z^2$ implies $z^2 R z^1$

d) Asymmetric if $z^1 R z^2$ doesn't imply $z^2 R z^1$

e) Ant-symmetric if $z^1 R z^2, z^2 R z^1$ implies $z^1 = z^2$

f) Transitivity if $z^1 R z^2$, $z^2 R z^3$ implies $z^1 R z^3$

Definition:-

A preorder relation is a binary relation that is both transitive and reflexive. We write $x^1 \prec_{pre} x^2$ as short hand of $x^1 R x^2$.And $(X, \prec_{pre})$ is preordered set.

Strict preference:-

$$x_1 \prec_{pre} x_2 \Leftrightarrow x_1 \preccurlyeq_{pre} x_2 \wedge not\,(x_2 \preccurlyeq_{pre} x_1)$$

Indifference:-

$$x_1 \sim_{pre} x_2 : \Leftrightarrow x_1 \preccurlyeq_{pre} x_2 \wedge x_2 \preccurlyeq_{pre} x_1$$

Incomparable:-

A pair of solution $x_1, x_2 \in X$ is said to be incomparable if and only if neither $x_1 \preccurlyeq_{pre} x_2$ nor $x_2 \preccurlyeq_{pre} x_1$ .We write $x_1 \parallel x_2$.

Strict preference is I reflexive and transitive and asymmetric.

Indifference is reflexive, transitive and symmetric.

Definition:-

A partial order relation is a preorder relation that is also ant-symmetric. We call $(X, \preccurlyeq_{par})$ a partial order.

Note: - for partial orders two elements that are indifference to each other are always equal:

$$x_1 \sim x_2 \Rightarrow x_1 = x_2$$

Strict preference:-

$$x_1 \prec_{par} x_2 \Leftrightarrow x_1 \preccurlyeq_{par} x_2 \wedge not(x_2 \preccurlyeq_{par} x_1)$$

Indifference:-

$$x_1 \sim_{par} x_2 : \Leftrightarrow x_1 \preccurlyeq_{par} x_2 \wedge x_2 \preccurlyeq_{par} x_1$$

- Partial order relation is special types of preorder relation..

Definition:-

Total order relation: - is partial order relation that fulfills comparability.

i.e. $x_1 \preccurlyeq_{par} x_2 \vee x_2 \preccurlyeq_{par} x_1 \ \forall x_1, x_2 \epsilon X$

Definition: - A binary relation R on a set Z is said to be strict partial order if R is reflexive and transitive.

- Since there is no total ordering in multi objective optimizations we have to use partial ordering.

1.3 Preference Orders and ordering cone

Many evolutionary solution frame work involve iteratively generating approximate Solutions depending on different evolutionary operators and presented to the decision maker to choose the most preferred one among the approximated solution. Solutions which are picked up for evolutionary operators are chosen using the idea of natural evolution process which is the survival of the fittest.

1.3.1 Preference orders

The preference attitude of the decision maker specifies the meanings of optimality which are usually represented binary relations on the objective space are called preference orders.

The preference attitude of the decision maker is usually represented by binary relations on the set $Z = f(X) \subset \mathbb{R}^k$ where, f is the vector valued function and X is a feasible decision set.

1.3.2 Dominance relation

The dominance relation (operator) relates two solutions; therefore it is binary operator the result of this operation for two solutions in objective space has:-

1) One solution dominate the other solution
2) Solutions Do not dominate each other

Definition: - A solution x^1 dominates a solution x^2 if $f_i(x^1) \leq f_i(x^2)\ \forall_{i\in\{1,2,3..,k\}}$ and there exist $j \in \{1,2,3,\ldots,k\}$ such that $f_j(x^1) < f_j(x^2)$.

- A solution not dominated by other solution in decision space is non-dominated solution.
- Points in the objective space are non-dominated if their corresponding variable vector is non-dominated solution.
- All non-dominated points in objective space together make up a non-dominated front.

Note:-1) A non –dominated solution are called Pareto optimal solution

2) A non-dominated points in a non –domination front are called Pareto optimal points

1.4 domination structures

A preference orders on a set Z that can be represented by a set valued map from Z to Z be called ordering cones.

- A binary relation may be regarded as the graph of a set valued map from Z to Z with set valued map G identify preference order $>$ with $G(z^1) = \{ z^2 \in Z : z^1 > z^2\}$ is a set of element in Z which are less preferred to z^1.
- The concept of ordering cones (domination structure) is used for representing preference order by set valued map.

For each $z^1 \in Z \subseteq \mathbb{R}^k$ the set of domination factor is defined as: - $D(z^1) = \{d \in \mathbb{R}^k : :z^1 > z^1 + d \cup \{0\}$, where D is called the domination structure.

Note: - when D is a constant set valued map, particularly $D(z)$ is a constant cone that is domination cone is the same at every point for $z \in Z$ $(D = D(z))$ is one of the most important and interesting case of domination structure.

D As preference order is:-

- Asymmetric if and only if $d \in D, d \neq 0, -d \notin D$. Again this is true if and only if $D$is pointed.
- Transitive if and only if $d_1, d_2 \in D$ implies $d_1 + d_2 \in D$, this is true if and only if D is convex.

Definition:-

(a) A non empty subset $R \subset \mathbb{R}^m \times \mathbb{R}^m$ is called binary relation R on $\mathbb{R}^m$. We write xRy for $(x, y) \in R$

(b) A binary relation $\leq$ on $\mathbb{R}^m$ is called partial ordering on $\mathbb{R}^m$,

if for arbitrary $w, x, y, z \in \mathbb{R}^m$:

i) $x \leq x$ (reflexive)

ii) $x \leq y,\ y \leq z \Longrightarrow x \leq z$ (transitive)

iii) $x \leq y, w \leq z \Longrightarrow x + w \leq y + z$ (compatibility with addition)

iv) $x \leq y,\ \alpha \in \mathbb{R}^+ \Longrightarrow \alpha x \leq \alpha y$ (compatibility with scalar multiplication).

(c) A partial ordering $\leq$ on $\mathbb{R}^m$ is called ant-symmetric if for arbitrary $x, y \in \mathbb{R}^m$

$$x \leq y,\ y \leq x \Longrightarrow x = y$$

Proportion 1.3:-
If C is pointed and convex cone, then the ordering relation $\leq_c$ is partial order.

where $y \leq_c z \Leftrightarrow z - y \in C \quad \forall y, z \in C$

1.5 Optimality and efficiency concepts

1.5.1 Optimality concept

Due to conflicting cases objective functions, it is not easy to find a single solution that would be optimal to objective functions simultaneously.

For concept of optimality, we employ domination solution property

1.5.1.1Pareto optimality

Here we consider the term solution as a variable vector and the term point as the corresponding objective vector.

The goals of multi-objective optimizations are:

1) Find a set of solution ,which lie on the Pareto –optimal front
2) Find a set of solution, which are diverse enough to represent the entire range the Pareto –optimal front.

Let $C \subset \mathbb{R}^k$ be an ordering cone in the objective space. We assume that it is the set of all dominated directions in $\mathbb{R}^m$ and refer to it as the domination cone D. A domination cone contains all vectors $d \epsilon \mathbb{R}^k$ such that for $z, z^1 \epsilon Z$ *if* $z^1 = z + d$ for some $d \epsilon D/\{0\}$, then z^1 is dominated by z. The vectors in domination cone can be thought as bad or dominated direction to travel within $\mathbb{R}^k$. A non dominated outcome is the one that is not dominated by any other out come

Definition

An element $z \epsilon$ Z is called non dominated element of the set Y with respect to domination cone D if there does not exist an element $z^1 \epsilon Z$ and vector $d \epsilon D/\{0\}$ such that $z = z^1 + d$ or equivalently $Z \cap (z - D/\{0\}) = \emptyset$ [3] .

The cone C can also be defined as the set of all preferred direction in $\mathbb{R}^k$ and then it is referred to as the preference cone P. The cone P is then the set of all directions $d \epsilon \mathbb{R}^k$ such that $z, z^1 \epsilon Z$ *if* $z^1 = z + d$ for some $d \epsilon P/\{0\}$ then z^1 is dominated by z. In other words preference cone contains all good or preferred directions to travel in $\mathbb{R}^k$ [3].

Definition

An element $z \epsilon Z$ is called a non-dominated element of the set Y with respect to the preference order P if there do not exist an element $y^1 \epsilon Z$ and direction $d \epsilon P/\{0\}$ such that $z = z^1 - d$ or equivalently $Z \cap (z + P/\{0\}) = \phi$

The relationship between the domination and preference cones is give by $D = -P$ since the negation dominated direction in D must be preferred direction in P to maintain consistency of d $\epsilon D(P)$ are preference. Typically it is assumed that the cones are convex and pointed. In other words the sum of two dominated (preferred) directions $d^1 . d^2 \epsilon D(P)$ is again dominated (preferred) direction $d^1 + d^2 \epsilon D(P)$ if both d and –d preferred (dominated) direction then $d = 0$.

1.6 Concepts of solution

Solution of multi-objective optimization problem is mainly related to the preference attitude of the decision maker. And let a domination structure representing the preference attitude of a decision maker is supposed to be a point -to -set map D from Y in to $\mathbb{R}^k$.

$$\text{Min}\{f(x) = (f_1(x), f_2(x), f_3(x), \dots, f_k(x))\}$$

$$(\mathbb{P}) \qquad \text{S.t } x \epsilon X \quad \text{where} \quad X \subset \mathbb{R}^n$$

$$Y = f(X) = \{y : y = f(x), x \epsilon X\}$$

Definition:-

A solution $x \epsilon X$ is said to be an efficient solution to ($\mathbb{P}$) with respect to a domination structure D if there is no $x' \in X$ such that $f(x) \in f(x') + Df(x)$ and $f(x) \neq f(x')$.

Definition:-

A solution $x' \in X$ is said to be Pareto optimal solution to the problem ($\mathbb{P}$) if there is no $x \in X$ such that $f(x) \leq f(x')$.

Definition:-

A point $x' \in X$ is said to be a weak Pareto optimal solution to problem ($\mathbb{P}$) if there is no $x \in X$ such that $f(x) < f(x')$.

Proposition 1.4:-

Given two domination structures D_1 and D_2 , D_1 is said to be included in D_2 if:-

$$D_1(y) \subset D_2(y) \qquad \forall_y \in Y$$

In this case $E(Y, D_2) \subset E(Y, D_1)$

Proof: - let $y \in E(Y, D_2)$

$\Rightarrow$ There is no $y \in Y$ such that $f(y) \in f(y) + D_2(f(y)/\{0\})$

Since D_1 is included in D_2 , there is no $y \in Y$

$$f(y) \in f(y) + D_1(f(y)/\{0\}$$

$$\Rightarrow y \in E(Y, D_1)$$

$$E(Y, D_2) \subset E(Y, D_1)$$

Proposition 1.5:-

Let D be a non – empty cone containing 0. Then $E(Y + D, D) \subset E(Y, D)$.

Proof:-

Let Y is empty set

Then $E(Y + D, D) = \emptyset$

And $E(Y, D) = \emptyset$

Let $Y \neq \emptyset$ and $y \epsilon E\ (Y + D, D)$ but $y \notin E(Y, D)$

Case i) $y \notin Y$ there exists $y' \in Y$ and $0 \neq d \in D$ such that

$y = y' + d$ Since $0 \epsilon D$, $Y \subset Y + D$

$\Rightarrow y' \in Y + D$

$\Rightarrow y \notin E(Y + D, D)$

This is a contradiction to $y \in E\ (Y + D, D)$

Case ii) let $y \in Y$ *there exists* $y' \in Y$ and $0 \neq d \in D$ such that

$y = y' + d$ Since $0 \in D$, , $Y \subset Y + D$

$\Rightarrow y' \in (Y + D)$

$\Rightarrow y \notin E(Y + D, D)$

This is a contradiction to $y \in E\ (Y + D, D)$

Hence from case i and ii we get $y \in E\ (Y, D)$

$$\Rightarrow E(Y + D, D) \subset E(Y, D)$$

Proposition1 .6:-

Let D be a non – empty pointed and convex cone containing 0 then

$$E(Y + D, D) = E(Y, D)$$

Proof:-

$E(Y + D, D) \subset E(Y, D)$ From proposition

It remains to show $E(Y,D) \subset E(Y+D,D)$

Let $y \in E(Y,D)$ and $y \notin E(Y+D,D)$

$\Rightarrow y' \in Y+D$ With $y - y' \in D/\{0\}$ then $y' = y'' + d'$, $y'' \in Y$ *and* $d' \in D$ $y = y' + d$, $d \in D/\{0\}$

$y = y'' + d' + d$ And $d' + d \in D$ (since D is convex)

And $d' + d \neq 0$ (D is pointed)

$y = y'' + d''$ Where $d'' = d' + d$

$\Rightarrow y \notin E(Y,D)$ This contradicts that $y \in E(Y,D)$ then our assumption $y \notin E(Y+D,D)$ is wrong

$\Rightarrow E(Y,D) \subset E(Y+D,D)$

Hence $\Rightarrow E(Y,D) = E(Y+D,D)$

Proposition 1.7:-

Let Y_1 and Y_2 be two sets and let D be a constant domination structure on $\mathbb{R}^k$ (a constant cone) then $E(Y_1+Y_2,D) \subset E(Y_1,D) + (Y_2,\mathrm{D})$

Proof:-

Let $y \in (Y_1+Y_2,D)$

$\Rightarrow \mathrm{y} = \mathrm{y}_1 + \mathrm{y}_2$, for some$\mathrm{y}_1 \in \mathrm{Y}_1$ and $\mathrm{y}_2 \in \mathrm{Y}_2$ and no $y' \in Y_1+Y_2$ such that $y = y' + D(y'/\{0\}$ and $y' = y_3 + y_4$ no $y_3 \in Y_1 and$, $y_4 \in Y_2$

$$y' = y_3 + y_4 + D(y_3 + y_4/\{0\})$$

$\Rightarrow \mathrm{y}_1 \in \mathrm{E}(\mathrm{Y}_1,\mathrm{D})$ And $\mathrm{y}_2 \in \mathrm{E}(\mathrm{Y}_2,\mathrm{D})$

Case i) $\mathrm{y}_1 \notin \mathrm{E}(\mathrm{Y}_1,\mathrm{D})$

$\Rightarrow$ There exists $y'' \in Y_1$ and $0 \neq d \in D$ such that $\mathrm{y}_1 = y'' + \mathrm{d}$, $y = y_1 + y_2$

$y = y'' + \mathrm{d} + y_2$, $y = y'' + y_2 + d$, $y'' + y_2 \in Y_1 + Y_2$

This a contradiction to $y \in (Y_1+Y_2,D)$

Hence $\mathrm{y}_1 \in \mathrm{E}(\mathrm{Y}_1,\mathrm{D})$

Case ii) let $\mathrm{y}_2 \notin \mathrm{E}(\mathrm{Y}_2,\mathrm{D})$ similar procedure is used to show.

Thus $\mathrm{y}_2 \in \mathrm{E}(\mathrm{Y}_2,\mathrm{D})$

Therefore $E(Y_1+Y_2,D) \subset E(Y_1,D) + (Y_2,\mathrm{D})$

UNIT TWO

2 MULTI-OBJECTIVE OPTIMIZATION

2.1 METHODS FOR SOLVING MULTI-OBJECTIVE OPTIMIZATION

Multi-objective optimization is designed to generate a single optimal solution (Non-dominated solution) or a set of Pareto optimal solutions are generated; then it is useful for the decision maker to be able to obtain small set of preferred Pareto optimal solutions using unbiased techniques of filtering solution. This suggests the need for an efficient selection procedure to identify such a prepared subset that reflects the preference of the decision maker with respect to the objective function.

In most methods we a prefer the criterion space to be the decision variable space for two reasons:

1. The dimension of the decision variable space is usually considerably larger than the dimension of the criterion space,
2. The decision maker is more interested in the criterion space.

The fundamental thing in multi objective optimization is that the optimal solution of multi objective optimization problem can be characterized as solution of a certain single objective optimization problems. Thus changing of the problem into a single or a family of a single objective optimization problems with a real valued objective function depending passively on parameter is called scalarization.

There are various methods for multi-objective optimization. None of them can be said to be superior to all others in general, since the preferences of the decision maker and during selecting a solution method, the specific features of the problem to be solved must be taken into consideration.

Multi-objective optimization methods can be classified as generating methods and preference based methods.

In generating methods the set of efficient solutions is generated to the decision maker, who then selects one of the alternatives while in preference methods, the preferences of the decision maker are taken into consideration as the solution process goes on and the solution that best satisfies the decision maker`s opinions chosen.

The methods can also be classified according to the participations of the decision maker in the solution process.

These are;

1. Methods where no articulation of preference information is used (no preference methods)
2. Methods where a priori articulation of preference information is used(a priori methods)
3. Methods where a posteriori articulation of preference information is used (a posteriori methods)
4. Methods where a progressive articulation of preference information is used (interactive methods) [6].

1) In no preference methods, where the preferences of the decision maker are not taken into consideration, the multi-objective optimization problem is solved with some relatively simple method and the solution obtained is presented to the decision maker.

Thus

- The decision maker may either accept or reject the solution,
- The solution best satisfying the decision could not be found with the methods. That is why no-preference methods are suitable for such situations where the decision maker does not have any special expectation of the solution and the decision is satisfied with just optimal solution.

The working order is 1.Analyst 2. None

2) In the a priori methods, the decision maker has to specify her or his preferences, opinions and wishes before the solution process. The difficulty is that the decision maker doesn`t necessarily know beforehand what is possible to achieve in the problem and how realistic the expectations are.

Here the working order is 1.Decision maker 2. Analyst

3) The class of interactive methods is the most efficient one of the three methods we have seen above. Many week points in the above three classes are overcome by these methods.

In interactive methods:

- The decision maker can specify and correct the preferences and selections as the solutions process go on since only a part of the Pareto optimal points has to be generated and evaluated.
- The decision maker works together with an analyst or an interactive computer program.
- The decision maker can be assumed to have more confidence in the final solution since the decision maker is involved throughout the solution process.

1. Analyst. 2. Decision maker. 3. Analyst. 4. Decision maker...

4) In posterior methods, after the Pareto optimal set has been generated, it is presented to the decision maker, who chooses the most preferred among the alternatives. That is why a posteriori methods sometimes called methods for generating Pareto optimal solutions.

The difficulty here is:

a. It is hard to the decision maker to select from among a large set of alternatives,
b. One more important question to the analyst is how to present or display the alternatives to the decision maker in an effective way.

The working order in these methods is 1. Analyst 2. Decision maker

Her are different methods which are used to generate Pareto optimal solutions

- Tabu search
- Hill climbing
- Simulated annealing
- Evolutionary algorithms

  The main concern of this project is in evolutionary algorithms which is a part of evolutionary computation.

2.2. Algorithmic concepts

2.2.1. Fitness assignment

Fitness value: - is a scalar value that representing quality of an individual in the objective space.

In single –objective optimization, here objective function and fitness function are often identical. In general there are three types of fitness assignment strategies in multi-objective optimization [8].

2.2.1.1 Aggregation based

The objectives are aggregated in to single parameterized objective function. The parameter of this function are systematically varied during optimization run in order to find a set of non-dominated solution instead of single –trade off solution.

2.2.1.2 Criterion based

This methods switch between the objectives during the selection phase each time an individual is chosen for reproduction, different objectives will decide which ,member of population will be copied in to the mating pool.

2.2.1.3 Pareto dominance based

Pareto –based fitness assignment first proposed by Goldberg. Different ways of exploiting the partial order on the population have been proposed. Some of them are:-

- dominance by rank

- dominance by count
- dominance by depth

Definition: - Rank is the number of individuals by which an individual is dominated.

The use of non-dominated ranking and selection is to move a population toward the Pareto front in multi-objective optimization. First all non-dominated individuals are assigned rank one and temporarily removed from the population then the next non-dominated individuals are assigned rank two. The process continues until all the population is suitably ranked. Finally the rank of an individual determines its fitness value. Remarkable here is the fact that fitness is related to the whole population.

Dominance count: How many individuals does an individual dominate?

Dominance depth: At which "front" is an individual located? (NSGA)

2.2.2 Diversity preservation

Another goal of multi-objective evolutionary algorithm providing diversified individuals to the decision maker that has somewhat uniform distribution across the known Pareto front. Various techniques are available for maintaining diversity in multi-objective evolutionary algorithm among them crowding distance is discussed here.

2.2.2.1 Crowding distance /clustering/

Solutions of the last accepted front are ranked according to the crowded comparison distance. The crowding distance value of a solution provides an estimate of the density of solutions surrounding that solution. Crowding distance is calculated by first sorting the set of solutions in ascending objective function values. The crowding distance value of a particular solution is the average distance of its two neighboring solutions. The boundary solutions which have the lowest and highest objective function values are given an infinite crowding distance values so that they are always selected. This process is done for each objective function. The final crowding distance value of a solution is computed by adding the entire individual crowding distance values in each objective function.

2.2.3 Elitism

Elitism addresses the problem of losing good solutions during the optimization process due to random effect. One way to deal with this problem is to combine the old populations and the offspring's. i.e. the mating pool after variation, and to apply deterministic selection procedure ,instead of replacing the old population by the modified mating pool. Alternatively, secondary population, the so- called archive can be to which the promising solution in the population is copied at each generation. The archive used as an external storage separate from the optimization process. If archive is maintained, the archive posses only the current approximation of the Pareto sets i.e. dominated archives members are removed [8].

UNIT THREE

3 Evolutionary methods

3.1 History

The term evolutionary algorithm simulates the process of natural evolution. The origins of evolutionary algorithms can be traced back to the late 1950s and since the 1970s several methodologies have been proposed mainly genetic algorithms, evolutionary programming and evolutionary strategies. All of these approaches operate on a set of candidate solution. Generating the Pareto set can be computationally expensive and is often infeasible because the complexity of underlying application prevents exact methods from being applicable. Due to these reason a number of stochastic search strategies such as evolutionary algorithm, Tabu search simulated annealing and ant colony optimization have been developed. They try to find a good approximation, that is (set solution whose objective vectors are not too far away from the optimal objective vectors) [8].

An evolutionary algorithm is characterized by three features:-

1) A set of solution candidates is maintained

2) A mating selection process is performed on these set

3) Several solutions may be combined in terms of recombination to generate new solutions

Similar To natural evolution, the solution candidates are called individuals and the set of solution candidate is called population. Each individual represents a possible solution i.e. a decision vector to the problem that is currently given. However an individual is not a decision vector rather it encodes a decision vector in appropriate representation. The mating selection process usually consists of two stages:- fitness assignment and sampling. In first stage, the individuals in the current population are evaluated in the objective spaces and then assigned fitness value. Next mating pool is created by random sampling from the population according to fitness value. Usually binary tournament selection is applied. These proceeds until mating pool are filled. Then the variation operators are applied to the mating pool. The variation operator takes certain number of parents and creates number of children by combining parts of the parents. By contrast mutation operators modifies individual by changing small parts in associated vectors according to give mutation rates. Due to random effect some individuals in the mating pool may not affected by variation operators. Finally selection determines which individual of the population modifies mating pool from new population. Generally evolutionary algorithm proceeds as follows: - initial population created at random (or according to predefined scheme) then the loop consists of steps evaluation (fitness assignment), selection, recombination and mutation is done certain number of times. Each loop of the iteration is generation. And predefined number of generation serves as termination criteria of the loop and also there may be other condition that used as stopping criteria i.e. existence on an individual with sufficient quality. At the end the best individuals in the final populations represent the outcome of evolutionary algorithms.

In defining evolutionary algorithm linking the real world to the evolutionary algorithm world that is to set up a bridge between the original problem context and the problem solving space where evolution takes place is the first step. Objects forming possible solution with in the original problem context are referred to as phenotypes [4] (behavioral and physical features of an individual that directly affect its value on the objective space) while their encoding that is the individuals with in evolutionary algorithm are genotypes

and evolutionary search takes place in the genotype space. A solution good phenotype is obtained after decoding the best genotype after termination. Representation is used in two slightly different ways. Sometimes it stands for mapping from the phenotype to the genotypes referred as encoding. And the inverse mapping from genotypes to phenotypes is usually named decoding and representation has to be invertible (to each genotype there has to be at most one phenotype)[4].

3.2. Components of Evolutionary Algorithms

In this section we discuss evolutionary algorithms in detail. Evolutionary Algorithms have a number of components, procedures, or operators that must be specified in order to define a particular Evolutionary Algorithms. The most important components [4],

• Representation (definition of individuals)

• Evaluation function (or fitness function)

• Population

• Parent selection mechanism

• Variation operators, recombination and mutation

• Survivor selection mechanism (replacement)

3.2.1. Representation (Definition of Individuals)

The first step in defining an Evolutionary Algorithms is to link the "real world" to the “Evolutionary Algorithms world", that is, to set up a bridge between the original problem context and the problem-solving space where evolution takes place. Objects forming possible solutions within the original problem context are referred to as phenotypes, while their encoding, that is, the individuals within the evolutionary algorithms are called genotypes. The first design step is commonly called representation, as it amounts to specifying a mapping from the phenotypes onto a set of genotypes that are said to represent these phenotypes. It is important to understand that the phenotype space can have very different from the genotype space, and that the whole evolutionary search takes

place in the genotype space. A solution a good phenotype is obtained by decoding the best genotype after termination. To this end it should hold that the (optimal) solution to the problem at hand a phenotype is represented in the given genotype space.

3.2.2. Evaluation Function (Fitness Function)

The role of the evaluation function is to represent the requirements to be satisfied. It forms the basis for selection, and thereby it facilitates improvements. Technically, it is a function or procedure that assigns a quality measure to genotypes. Typically, this function is composed from a quality measure in the phenotype space and the inverse representation. The evaluation function is commonly called the fitness function in Evolutionary Algorithms.

.

3.2.3. Population

The role of the population is to hold (the representation of) possible solutions. Given a representation, defining a population can be as simple as specifying how many individuals are in it, that is. Setting the population size in general, they take the whole current population into account, and choices are always made relative to what we have. For instance, the best individual of the given population is chosen to seed the next generation, or the worst individual of the given population is chosen to be replaced by a new one. In almost all Evolutionary Algorithms applications the population size is constant and does not change during the evolutionary search. The diversity of a population is a measure of the number of different solutions present. No single measure for diversity exists. Typically people might refer to the number of different fitness values present, the number of different phenotypes present, or the number of different genotypes.

3.2.4 Parent Selection Mechanism

Selection is picking promising solution for variation and usually performed in randomized fashion. In other words choosing individuals for recombination and mutation to become parents to the next generation and selection effectively gives an individual with higher fitness value probably contributing one or more children in succeeding generation. The role of parent selection or mating selection is to distinguish among individuals based on

their quality, in particular, to allow the better individuals to become parents of the next generation. An individual is a parent if it has been selected to undergo variation in order to create offspring. Together with the survivor selection mechanism, parent selection is responsible for pushing quality improvements. In Evolutionary Algorithms, parent selection is typically probabilistic. Thus, high-quality individuals get a higher chance to become parents than those with low quality. Nevertheless, low-quality individuals are often given a small, but positive chance; otherwise the whole search could become too greedy and get stuck in a local optimum.

3.2.5. Variation Operators

The role of variation operators is to create new individuals from old ones. In the corresponding phenotype space this amounts to generating new candidate solutions. Variation operators in evolutionary algorithms are divided into two types based on their number of objects that it takes as inputs.

3.2.5.1. Recombination (crossover)

The process in which a new individual solution is created from the information contained within two or more parent solution is considered .Recombination operators are usually applied probabilistically according to a crossover rate (P_c the chance that a chosen pair of parents under goes this operator),which is typically in the rate [0.5 1.0] . Usually two parents are selected and a random variable is drawn from [0 1] and compared to (P_c), if the value is lower ,two off springs are created through recombination of two parents otherwise they created by copying the parents it results some copies of the parents and other individuals that present previously un seen solution.

3.2.5.2. Mutation

It is applied to one genotype and delivers a (slightly) modified mutant, the child or offspring of it. In general mutation is supposed to cause a random, unbiased change. Mutation is random change of the value of a gene (string) in the population. Mutation is performed in a way that maximizes fitness value and with strong mutation, the evolution becomes pure random search, with too weak mutation no real progress can be achieved.

There is mutation rate (i.e. how probable it is changed in one mutation operator.) thus adaptation mechanism for the mutation strength are a necessity for many optimization problem.

Example: - Before mutation let the individual is 4 1 3 8 0 7 1 3 5 1

After mutation the individual becomes 4 1 3 8 0 7 3 1 5 1

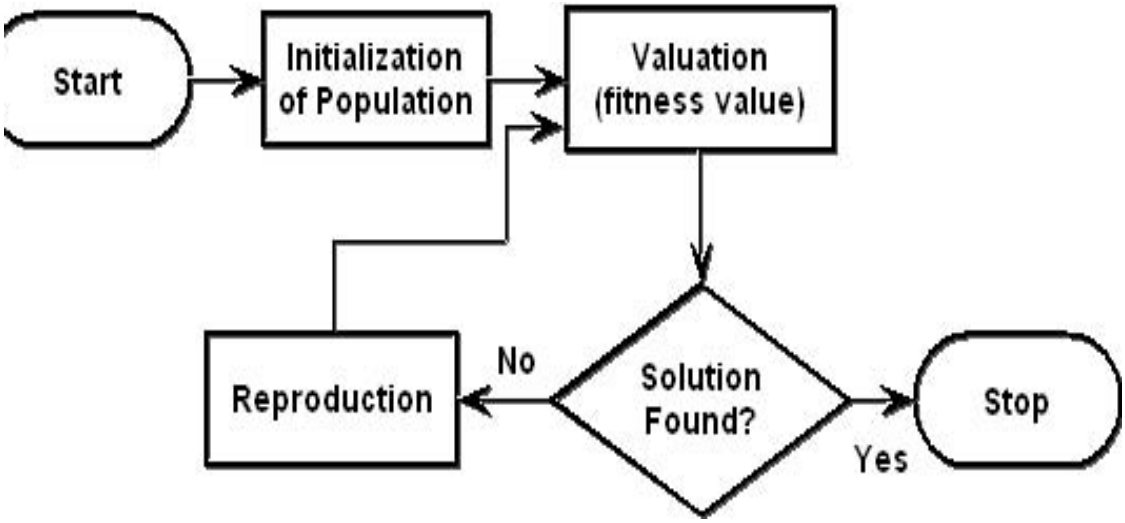

Figure 3.1 Flowchart of evolutionary algorithm iteration

Adopted from: - Carlos, A.C.C, (2).

3.3. An overview of Genetic algorithm

Genetic algorithms are adaptive methods which may be used to solve search and optimization problem they are based on genetic process of biological organisms over many generations, natural populations evolve according to the principles of natural selection and survival of the fittest. By mimicking this process genetic algorithm are able to evolve solutions to real world problems, if they have been suitably encoded. The basic principles of genetic algorithms are first founded by Holland [6].

In genetic algorithm terminology a solution vector $x \in X$is called chromosomes. Chromosomes are made of discrete units called genes. Each gene controls one or more features of the chromosomes. In the original implementation of genetic algorithm; genes are assumed to be binary numbers. In later implementation more varied gene types have been introduced such as real coded genetic algorithm. Genetic algorithm operates with a collection of chromosomes called population. The population is randomly initialized.

Genetic algorithms with populations of individuals each are representing a possible solution to give problem. Each individual is assigned a fitness score according to how good solution to the problem it is. The highly fit-individuals are given opportunities to reproduce by cross breeding with other individual in the population. This produces individuals as offspring which share some features taken from each parent. The least fit members of the populations are less likely to get selected for reproduction and die out.

A whole new population of possible solutions is thus produced by selecting the best individual from the current generation and mating them to produce a new set of individuals. This new generation contains a higher proportion of the characteristics processed by the good members of the previous generation. In this way, over many generations, good characteristics are spread throughout the populations.

The procedure for genetic algorithms is:-

Step 1:- Initialization: - randomly generate the initial population of size N and set I =0

Step 2:-Fitness assignment: - evaluate the fitness value for each population based on its objective function value.

Step 3:- If stopping criteria is satisfied, terminate the search and display the result else go to step 4

Step 4:- Crossover :-to generate the offspring using cross over, randomly select two parents solution from the initial population and then generate the two off springs using cross over operator.

Step 5 Mutation:-this operator randomly selects one parent and applies the mutation operator to generator offspring.

Step 6 Selection: - select N solution from generated population and the old population based on their fitness. Set generation i = i+1. Go to step 2 [10].

There are many formulation of multi-objective generic algorithms like multi-objective genetic algorithms (MOGA), Niched Pareto generic algorithm, Non-dominated Sorting generic Algorithm, Strength Pareto Evolutionary Algorithm (SPEA) and Non-dominated Sorting generic Algorithm (NSGA-II). The NSGA-II is advanced version of NSGA. Next NSGA-II is described below:

3.4. Non-dominated sorting genetic algorithm (NSAG-II)

Works on after initializing the population for each population it has to determine how many solutions dominate it and the set of solution to which it dominates and preserves diversity by using density of solution surrounding a particular. i.e. (Crowding distance: - how close an individual to its neighborhoods) solutions in the population i.e. computing the average distance of two points on either side of this points along each objectives of the problem. Selection is performed which takes in to consideration both non –domination rank of individual in the population and its crowding distances. Applying operators, simulated binary cross over and polynomial mutation to generate an offspring and combine with the best parents with the best offspring to keep best solutions from being lost.

The main objective of NSAG-II is to find multiple Pareto-optimal solutions in one single simulation run. NSAG-II works with a population. So, in this project we will consider NSAG-II; this is known as Elitism Non-dominated sorting algorithm. The main advantage of using these techniques is given below:

- ✓ It uses non dominated sorting techniques to provide the solution as close to the Pareto-optimal solution as positive
- ✓ It uses crowding distance techniques to provide diversity in solution.

- ✓ It also uses elitist techniques to preserve the best solution of current population of current population in next generation [10].

3.4.1 NSGA-II Algorithms

Population Initialization

The population is initialized based on the problem range and constraints and generated as follows:-

Let a vector $(P_1, P_2, P_3, \dots, P_N)$ is a chromosomes to represent a solution to the optimization problem.

$$P_i = P_i^l + \alpha_i(P_i^u - P_i^l)$$

Where, P_i^u and P_i^l are upper and lower bounds of P_i

α_i Uniformly distributed random number in the range 0 to 1

Non-Dominated sort

- The initialized population is sorted based on non-domination.

- for each individual p in main population P do the following
  - Initialize $S_p = \emptyset$. This set would contain all the individuals that is being dominated by p.
  - Initialize $n_p = 0$. This would be the number of individuals that dominate p.
  - for each individual q in P
    - if p dominated q then
      - add q to the set S_p i.e. $S_p = S_p \cup \{q\}$
    - else if q dominates p then
      - increment the domination counter for p i.e. $n_p = n_p + 1$
  - if $n_p = 0$ i.e. no individuals dominate p then p belongs to the first front: Set rank of individual p to one i.e. $p_{rank} = 1$. Update the first front set by adding p to front one i.e. $F_1 = F_1 \cup \{p\}$
- This is carried out for all the individuals in main population P.
- Initialize the front counter to one $i = 1$
- following is carried out while the i^{th} front is nonempty i.e. $F_i \neq \emptyset$

- $Q = \emptyset$; The set for storing the individuals for $(i+1)^{th}$ front.
- for each individual p in front F_i
  - for each individual q in S_p (S_p is the set of individuals dominated by p)
    - $n_q = n_q - 1$, decrement the domination count for individual q.
    - if $n_q = 0$ then none of the individuals in the subsequent fronts would dominate q. Hence set $q_{rank} = i + 1$. Update the set Q with individual q i.e. $Q = Q \cup q$.
- Increment the front counter by one.
- Now the set Q is the next front and hence $F_i = Q$

Crowding Distance

Once the non-dominated sort is complete the crowding distance is assigned. Since the individuals are selected based on rank and crowding distance all the individuals in the population are assigned a crowding distance value. Crowding distance is assigned front wise and comparing the crowding distance between two individuals in different front is meaningless.

The crowing distance is calculated as below:-

- For each front F_i, n is the number of individuals.
  - initialize the distance to be zero for all the individuals i.e. $F_i(d_j) = 0$, where j corresponds to the j^{th} individual in front F_i.
  - for each objective function m
    - Sort the individuals in front F_i based on objective m i.e. $I = sort(F_i, m)$.
    - Assign infinite distance to boundary values for each individual in F_i i.e. $I(d_1) = \infty$ and $I(d_n) = \infty$
    - for $k = 2$ to $n - 1$.
      - $I(d_k) = I(d_k) + \frac{I(k+1).m - I(k-1).m}{f_m^{max} - f_m^{min}}$

- $I(k).m$ is the value of the m^{th} objective function of the k^{th} individual in I

The basic idea behind the crowing distance is finding the Euclidian distance between each individual in a front based on their m objectives. The individuals in the boundary are always selected since they have infinite distance assignment.

Selection

Once the individuals are sorted based on non-domination and with crowding distance assigned, the selection is carried out using a *crowded-comparison-operator* ($\prec_n$).The comparison is carried out as below based on

(1) non-domination rank p_{rank} i.e. individuals in front F_i will have their rank as $p_{rank} = i$

(2) crowding distance $F_i(d_j)$

- $p \prec_n q$. If
  - $p_{rank} < q_{rank}$
  - or if p and q belong to the same front F_i then $F_i(d_p) > F_i(d_q)$ i.e. the crowing distance should be more.

The individuals are selected by using a binary tournament selection with crowded-comparison-operator.

Genetic Operators

Real-coded GA's use *Simulated Binary Crossover (SBX),* Operator for crossover and *polynomial mutation.*

Simulated Binary Crossover Simulated binary crossover simulates the bi-nary crossover observed in nature and is given as below. A spread factor β_k is obtained as the ratio of absolute difference in offspring values to that of parents.

$$\beta_k = \left|\frac{c_{1,k} - c_{2,k}}{P_{1,k} - P_{2,k}}\right|$$

β_k (≥ 0) is a sample from a random number generated having the density

$$P(\beta_k) = \frac{1}{2}(\eta_c + 1)\beta_k{}^{\eta_c}, \qquad \text{if } 0 \leq \beta_k \leq 1$$

$$P(\beta_k) = \frac{1}{2}(\eta_c + 1)\frac{1}{\beta_k{}^{\eta_c}} \quad , \qquad \text{if} \quad \beta_k > 1$$

And the offspring is calculated as follows:-

$$c_{1,k} = \frac{1}{2}[(1 - \beta_k)P_{1,k} + (1 + \beta_k)P_{2,k}] \qquad (*)$$

$$c_{2,k} = \frac{1}{2}[(1 + \beta_k)P_{1,k} + (1 - \beta_k)P_{2,k}] \qquad (**)$$

Where $c_{i,k}$- is the i^{th} child with k^{th} component and $P_{i,k}$ is the selected parent.

This distribution can be obtained from a uniformly sampled random number u between 0 and 1. η_c - is the distribution index for crossover.

$$\beta_k = (2u)^{\frac{1}{\eta+1}} \text{ , if } u > 0.5$$

$$\beta_k = \frac{1}{[2(1-u)]^{\frac{1}{\eta+1}}} \quad , \text{ if otherwise} \qquad (***)$$

Step 1: Choose a random number $u \in (0,1)$

Step 2: Calculate β_k using equation (***)

Step 3: Compute children solutions using equations (*) and (**)

Polynomial Mutation

$$c_k = p_k + (p_k^u - p_k^l)\delta_k$$

Where: _ c_k is the child and p_k is the parent with

- p_k^u *Is* the upper bound on the parent component
- p_k^l Is lower bound on the parent component
- δ_k Is small variation which is calculated from a polynomial distribution by using :-

$$\delta_k = (2r_k)^{\frac{1}{\eta_m + 1}} - 1 \text{ , if } r_k < 0.5$$

$$\delta_k = 1 - [2(1 - r_k)]^{\frac{1}{\eta_m + 1}} \text{ , if } r_k \geq 0.5$$

- r_k is a uniformly sampled random number between$(0,1)$ and η_m is mutation distribution index.

Recombination and Selection

The offspring population is combined with the current generation population and selection is performed to set the individuals of the next generation. Since all the previous and current best individuals' are added in the population, elitism is ensured. Population is now sorted based on non-domination. The new generation is filled by each front subsequently until the population size exceeds the current population size. If by adding all the individuals in front F_j the population exceeds N then individuals in front F_j are selected based on their crowding distance in the descending order until the population size is N.And hence the process repeats to generate the subsequent generations[11].

3.5. Application of NSGA-II

The real power and emission problem is to minimize various objective functions like fuel cost and emission, while satisfying several equality and in equality constraints.

The data for a system of three generators test system has been presented (appendix-A).This test system consists of three generators provide with fuel and emission coefficients. Test system losses are taken in the form of B-coefficients. The load demand is 850MW for this system. Solve power dispatch problem using NSAG-II for three generators [10].

Problem formulation:-

1) Minimization of fuel cost

The fuel cost objective is represented by the expression below

$$\text{Min} \quad F_1 = \sum_{i=1}^{N_g}(a_i + b_i P_{G_i} + c_i P_{G_i}^2) \qquad \$/\text{h} \qquad i = 1,2,3,\dots,N_g$$

$$\text{S.t.} \qquad h(P_{G_i}) = 0$$

$$g(P_{G_i}) \leq 0$$

Where:-

- a_i ,b_i , c_i are of the fuel cost coefficients i-th generators.
- P_{G_i} is the real power output of i-th generator
- N_g is the total number of generators
- F_i is the total fuel cost
- $h(P_{G_i})$ is the power balance constraint
- $g(P_{G_i})$ is the generational capacity constraints

Emission:-

2) For $(No_x, So_x \ )$

(i) Minimization of No_x emission

The No_x emission objective is represented by the expression below

$$\text{Min} \quad F_2 = \sum_{i=1}^{N_g}(a_{Ni} + b_{Ni}P_{G_i} + c_{Ni}P_{G_i}^2) \quad \$/\text{h} \quad i = 1,2,3, \dots, N_g$$

(ii) Minimization of So_x emission

The So_x emission objective is represented by the expression below

$$\text{Min} \quad F_3 = \sum_{i=1}^{N_g}(a_{Si} + b_{Si}P_{G_i} + c_{Si}P_{G_i}^2) \qquad \$/\text{h} \qquad i = 1,2,3, \dots, N_g$$

Problem formulation for the given objectives is given as

$$\text{Min} \quad F(P_{G_i}) = [F_2 \ F_3 \ ] \qquad i = 1,2,3, \dots, N_g$$

$$\text{St} \quad :- \quad h(P_{G_i}) = 0$$

$$g(P_{G_i}) \leq 0$$

Where:-

- F_2 is the total No_x emission
- F_3 is the total So_x emission

- a_{Ni}, b_{Ni}, c_{Ni} are the No_x emission coefficients of the i^{th} operator
- a_{Si}, b_{Si}, c_{Si} are the So_x emission coefficients of the i^{th} operator
- a_{Ci}, b_{Ci}, c_{Ci} are the Co_x emission coefficients of the i^{th} operation

Power balance and generation capacity constraints

1) Power balance constraints: - the total power genera must cover the total demand P_D and the real power loss in the transmission.

$$\sum_{i=1}^{N_g} P_{Gi} - P_D - P_{loss} = 0$$

2) Generation capacity constraints:- for stable operation ,the generator out puts and bus voltage magnitudes are restricted by lower and upper limits as follow.

$$P_{Gimin} \leq P_{Gi} \leq P_{Gi} \qquad i = 1,2,3,\dots,N_g$$

Step-1 Initialization:-

Initial population of 10 is taken

Table 3-1 initialization of population

S. no	P_1	P_2	P_3	Fuel cost(S/h)	Emission (ton/hr)
1	371.68	296.46	183.51	8231.29	0.11029
2	411.67	381.30	057.16	8220.82	0.10611
3	579.63	180.32	087.51	8277.56	0.09789
4	566.77	189.54	100.71	8348.87	0.09721
5	378.50	356.99	107.64	8134.05	0.10057
6	387.34	317.49	141.11	8159.69	0.10079
7	486.82	270.35	094.84	8238.13	0.09562
8	543.72	136.30	178.45	8398.83	0.11004
9	571.18	216.78	067.57	8335.91	0.09916
10	566.61	153.65	127.63	8285.77	0.09935

Step-2:-Non-dominated sorting

Entire population has been sorted according to its non-dominated level. Each solution assigned to its fitness value according to its non-dominated level. Where, level one is considered to be best level. The solution at level one did not dominated by any other solution. Whereas, solutions at the other level are dominated by at least one solution of level one, N. no is the number of the solution which dominates the respective solution.

TABLE 3.2 Non-dominated sorting of the initial population

S. no	P_1	P_2	P_3	Fuel cost(S/h)	Emission (ton/hr)	N. no	rank
1	378.50	356.99	107.64	8134.05	0.10057	0	1
2	486.82	270.35	094.84	8238.13	0.09562	0	1
3	579.63	180.32	087.51	8277.56	0.09789	1	2
4	566.77	189.54	100.71	8348.87	0.09721	1	2
5	387.34	317.49	141.11	8159.69	0.10079	1	2
6	411.67	381.30	057.16	8220.82	0.10611	2	3
7	571.18	216.78	067.57	8335.91	0.09916	2	3
8	566.61	153.65	127.63	8285.77	0.09935	2	3
9	371.68	296.46	183.51	8231.29	0.11029	3	4
10	543.72	136.30	178.45	8398.83	0.11004	8	4

Step -3:- Crowing distance

After non-dominated sorting is done, the crowding distance is assigned to each solution. Crowding distance is assigned front wise

Table 3.3 Crowding distance of sorted population

S. no	P_1	P_2	P_3	Fuel cost(S/h)	Emission (ton/hr)	N. no	rank	cd

1	378.50	356.99	107.64	8134.05	0.10057	0	1	inf
2	486.82	270.35	094.84	8238.13	0.09562	0	1	inf
3	579.63	180.32	087.51	8277.56	0.09789	1	2	2
4	566.77	189.54	100.71	8348.87	0.09721	1	2	inf
5	387.34	317.49	141.11	8159.69	0.10079	1	2	inf
6	411.67	381.30	057.16	8220.82	0.10611	2	3	inf
7	571.18	216.78	067.57	8335.91	0.09916	2	3	inf
8	566.61	153.65	127.63	8285.77	0.09935	2	3	2
9	371.68	296.46	183.51	8231.29	0.11029	3	4	inf
10	543.72	136.30	178.45	8398.83	0.11004	8	4	inf

Step 4: - Tournament selection

After the individual is sorted based on non-domination and with crowding distance assigned, the selection is carried out two solutions are randomly picked from the population and best solution is selected. Selection is carried out using crowded – comparison operator. Solution with better rank is selected as winner otherwise if both have the same rank then the solution which is lesser crowding region is proffered. Copy of the winner placed in a mating pool.

Example: - take solution 2 and solution 7

2	486.82	270.35	094.84	8238.13	0.09562	0	1	inf
7	571.18	216.78	067.57	8335.91	0.09916	2	3	inf

- Solution 2 is selected according to crowded –comparison operator.

Table 3.4 Solution selected by tournament selection

S. no	P_1	P_2	P_3	Fuel	Emission	N. no	rank	c. d

				cost(S/h)	(ton/hr)			
1	486.82	270.35	094.84	8238.13	0.09562	0	1	inf
2	378.50	356.99	107.64	8134.05	0.10057	0	1	inf
3	411.67	381.30	057.16	8220.82	0.10611	2	3	inf
4	566.77	189.54	100.71	8348.87	0.09721	1	2	inf
5	378.50	356.99	107.64	8134.05	0.10057	0	1	inf

Step 5: - Cross over and mutation

Applying crossover and mutation to each individual in the mating pool and select the parents. For two parents perform crossover generating offspring and mutation for one parent and generate the offspring.

Parent -1 378.50 356.99 107.64 8134.05 0.10057

Parent-2 411.67 381.67 057.16 8220.82 0.10611

0ffspring-1 408.35 384.16 055.99 8221.90 0.106612

Offspring -2 381.8 354.12 108.80 8133.36 0.100345

Mutation:-

Parent 385.51 352.23 108.84 8164.76 0.10023

Offspring 387.46 346.73 115.07 8188.29 0.10012

Step 6 Recombination

The offspring population Q_t is combined with the current generation P_t population and selection is performed to set the individual of the next generation of R_t . Since all pervious and current best individuals are added in the population elitist is ensured and sorted based on non-domination. The new generation is filled by each front subsequently until population size exceeds the current population size. If by adding all the individual in R_t the population exceeds N then individuals are selected based on their

crowding distance in ascending order until population size is N other individuals above N has been rejected. Hence the process repeats to generate the subsequent generations.

Table 3.5 Regenerated population after the recombination

S. no	P_1	P_2	P_3	Fuel cost(S/h)	Emission (ton/hr)	rank	c. d
1	375.98	356.31	109.30	8119.75	0.10056	1	inf
2	375.98	356.31	109.30	8119.75	0.10056	1	0
3	380.19	354.12	108.8	8133.36	0.10034	1	0.001895
4	380.50	357.78	104.25	8129.00	0.10051	1	0.0001828
5	375.98	356.31	109.30	8119.75	0.10056	1	0
6	375.98	356.31	109.30	8119.75	0.10056	1	0
7	375.98	356.31	109.30	8119.75	0.10056	1	0
8	486.22	266.06	097.37	8216.87	0.95362	1	inf
9	375.98	356.31	109.30	8119.75	0.10056	1	inf
10	374.57	359.81	110.15	8146.89	0.10112	2	inf

3.6. NSGA-II procedures

Step-1:- A random initial population generated of size N. This step is repeated N times where N is the size of the population.

Step-2:- The population produced above is sorted using fast non-dominated sorting for producing fronts. This process is repeated untill the whole population is divided into fronts.

Step-3:- Initially crowded distance assignment is done for solution of a front and then the front is included parent population. For the crowded distance computation the population is sorted according each objectives value. The boundary solutions are assigned initiate

distance values all other value intermediate solutions are assigned a distance value equal to the absolute difference in the function values of two adjacent values of two solutions.

Step-4:- Cross over and mutation is applied to the parent population generated above to produce child population. To create new offspring, simulated binary crossover (SBX) operator and polynomial mutation operator used.

Step-5:- The parent and child population are combined together to produce a population of size 2N. This step performed for elitism, the elitism selected the best N value.

Step-6:- Stopping criteria is checked if achieved the Pareto optimal front is printed else go to step 2.

3.7 Matlb for NSGA-II

```
for i = 1 : N
    % Number of individuals that dominate this individual
    individual(i).n = 0;
    % Individuals which this individual dominate
    individual(i).p = [];
    for j = 1 : N
        dom_less = 0;
        dom_equal = 0;
        dom_more = 0;
        for k = 1 : M
            if (x(i,V + k) < x(j,V + k))
                dom_less = dom_less + 1;
            elseif (x(i,V + k) == x(j,V + k))
                dom_equal = dom_equal + 1;
            else
                dom_more = dom_more + 1;
            end
        end
        if dom_less == 0 && dom_equal ~= M
            individual(i).n = individual(i).n + 1;
        elseif dom_more == 0 && dom_equal ~= M
            individual(i).p = [individual(i).p j];
        end
    end
    if individual(i).n == 0
        x(i,M + V + 1) = 1;
        F(front).f = [F(front).f i];
    end
end
% Find the subsequent fronts
while ~isempty(F(front).f)
```

```
   Q = [];
   for i = 1 : length(F(front).f)
       if ~isempty(individual(F(front).f(i)).p)
              for j = 1 : length(individual(F(front).f(i)).p)
              individual(individual(F(front).f(i)).p(j)).n = ...
                     individual(individual(F(front).f(i)).p(j)).n - 1;
                     if individual(individual(F(front).f(i)).p(j)).n ==
0
                            x(individual(F(front).f(i)).p(j),M + V + 1)
= ...
                        front + 1;
                    Q = [Q individual(F(front).f(i)).p(j)];
                end
            end
       end
   end
   front =  front + 1;
   F(front).f = Q;
end

[temp,index_of_fronts] = sort(x(:,M + V + 1));
for i = 1 : length(index_of_fronts)
    sorted_based_on_front(i,:) = x(index_of_fronts(i),:);
end
current_index = 0;
% Find the crowding distance for each individual in each front
for front = 1 : (length(F) - 1)
%    objective = [];
    distance = 0;
    y = [];
    previous_index = current_index + 1;
    for i = 1 : length(F(front).f)
        y(i,:) = sorted_based_on_front(current_index + i,:);
    end
    current_index = current_index + i;
    % Sort each individual based on the objective
    sorted_based_on_objective = [];
    for i = 1 : M
        [sorted_based_on_objective, index_of_objectives] = ...
            sort(y(:,V + i));
        sorted_based_on_objective = [];
        for j = 1 : length(index_of_objectives)
            sorted_based_on_objective(j,:) =
y(index_of_objectives(j),:);
        end
        f_max = ...
            sorted_based_on_objective(length(index_of_objectives), V +
i);
        f_min = sorted_based_on_objective(1, V + i);
        y(index_of_objectives(length(index_of_objectives)),M + V + 1 +
i)...
```

```
            = Inf;
        y(index_of_objectives(1),M + V + 1 + i) = Inf;
         for j = 2 : length(index_of_objectives) - 1
            next_obj  = sorted_based_on_objective(j + 1,V + i);
            previous_obj  = sorted_based_on_objective(j - 1,V + i);
            if (f_max - f_min == 0)
                y(index_of_objectives(j),M + V + 1 + i) = Inf;
            else
                y(index_of_objectives(j),M + V + 1 + i) = ...
                     (next_obj - previous_obj)/(f_max - f_min);
            end
         end
    end
    distance = [];
    distance(:,1) = zeros(length(F(front).f),1);
    for i = 1 : M
        distance(:,1) = distance(:,1) + y(:,M + V + 1 + i);
    end
    y(:,M + V + 2) = distance;
    y = y(:,1 : M + V + 2);
    z(previous_index:current_index,:) = y;
end
f = z();


for i = 1 : pool_size
    % Select n individuals at random, where n = tour_size
    for j = 1 : tour_size
        % Select an individual at random
        candidate(j) = round(pop*rand(1));
        % Make sure that the array starts from one.
        if candidate(j) == 0
            candidate(j) = 1;
        end
        if j > 1
            % Make sure that same candidate is not choosen.
            while ~isempty(find(candidate(1 : j - 1) == candidate(j)))
                candidate(j) = round(pop*rand(1));
                if candidate(j) == 0
                    candidate(j) = 1;
                end
            end
        end
    end
    % Collect information about the selected candidates.
    for j = 1 : tour_size
        c_obj_rank(j) = chromosome(candidate(j),rank);
        c_obj_distance(j) = chromosome(candidate(j),distance);
    end
    % Find the candidate with the least rank
    min_candidate = ...
```

```
        find(c_obj_rank == min(c_obj_rank));
    % If more than one candiate have the least rank then find the
candidate
    % within that group having the maximum crowding distance.
    if length(min_candidate) ~= 1
        max_candidate = ...
        find(c_obj_distance(min_candidate) ==
max(c_obj_distance(min_candidate)));
        % If a few individuals have the least rank and have maximum
crowding
        % distance, select only one individual (not at random).
        if length(max_candidate) ~= 1
            max_candidate = max_candidate(1);
        end
        % Add the selected individual to the mating pool
        f(i,:) =
chromosome(candidate(min_candidate(max_candidate)),:);
    else
        % Add the selected individual to the mating pool
        f(i,:) = chromosome(candidate(min_candidate(1)),:);
    end
end


[N,m] = size(parent_chromosome);

clear m
p = 1;
% Flags used to set if crossover and mutation were actually performed.
was_crossover = 0;
was_mutation = 0;


for i = 1 : N
    % With 90 % probability perform crossover
    if rand(1) < 0.9
        % Initialize the children to be null vector.
        child_1 = [];
        child_2 = [];
        % Select the first parent
        parent_1 = round(N*rand(1));
        if parent_1 < 1
            parent_1 = 1;
        end
        % Select the second parent
        parent_2 = round(N*rand(1));
        if parent_2 < 1
            parent_2 = 1;
        end
        % Make sure both the parents are not the same.
```

```
        while
isequal(parent_chromosome(parent_1,:),parent_chromosome(parent_2,:))
            parent_2 = round(N*rand(1));
            if parent_2 < 1
                parent_2 = 1;
            end
        end
        % Get the chromosome information for each randomnly selected
        % parents
        parent_1 = parent_chromosome(parent_1,:);
        parent_2 = parent_chromosome(parent_2,:);
        % Perform corssover for each decision variable in the
chromosome.
        for j = 1 : V
            % SBX (Simulated Binary Crossover).
            % For more information about SBX refer the enclosed pdf
file.
            % Generate a random number
            u(j) = rand(1);
            if u(j) <= 0.5
                bq(j) = (2*u(j))^(1/(mu+1));
            else
                bq(j) = (1/(2*(1 - u(j))))^(1/(mu+1));
            end
            % Generate the jth element of first child
            child_1(j) = ...
                0.5*(((1 + bq(j))*parent_1(j)) + (1 -
bq(j))*parent_2(j));
            % Generate the jth element of second child
            child_2(j) = ...
                0.5*(((1 - bq(j))*parent_1(j)) + (1 +
bq(j))*parent_2(j));
            % Make sure that the generated element is within the
specified
            % decision space else set it to the appropriate extrema.
            if child_1(j) > u_limit(j)
                child_1(j) = u_limit(j);
            elseif child_1(j) < l_limit(j)
                child_1(j) = l_limit(j);
            end
            if child_2(j) > u_limit(j)
                child_2(j) = u_limit(j);
            elseif child_2(j) < l_limit(j)
                child_2(j) = l_limit(j);
            end
        end
        % Evaluate the objective function for the offsprings and as
before
        % concatenate the offspring chromosome with objective value.
        child_1(:,V + 1: M + V) = evaluate_objective(child_1, M, V);
        child_2(:,V + 1: M + V) = evaluate_objective(child_2, M, V);
```

```
        % Set the crossover flag. When crossover is performed two
children
        % are generate, while when mutation is performed only only
child is
        % generated.
        was_crossover = 1;
        was_mutation = 0;
    % With 10 % probability perform mutation. Mutation is based on
    % polynomial mutation.
    else
        % Select at random the parent.
        parent_3 = round(N*rand(1));
        if parent_3 < 1
            parent_3 = 1;
        end
        % Get the chromosome information for the randomnly selected
parent.
        child_3 = parent_chromosome(parent_3,:);
        % Perform mutation on eact element of the selected parent.
        for j = 1 : V
           r(j) = rand(1);
           if r(j) < 0.5
               delta(j) = (2*r(j))^(1/(mum+1)) - 1;
           else
               delta(j) = 1 - (2*(1 - r(j)))^(1/(mum+1));
           end
           % Generate the corresponding child element.
           child_3(j) = child_3(j) + delta(j);
           % Make sure that the generated element is within the
decision
           % space.
           if child_3(j) > u_limit(j)
               child_3(j) = u_limit(j);
           elseif child_3(j) < l_limit(j)
               child_3(j) = l_limit(j);
           end
        end
        % Evaluate the objective function for the offspring and as
before
        % concatenate the offspring chromosome with objective value.
        child_3(:,V + 1: M + V) = evaluate_objective(child_3, M, V);
        % Set the mutation flag
        was_mutation = 1;
        was_crossover = 0;
    end
    % Keep proper count and appropriately fill the child variable with
all
    % the generated children for the particular generation.
    if was_crossover
        child(p,:) = child_1;
        child(p+1,:) = child_2;
```

```
        was_cossover = 0;
        p = p + 2;
    elseif was_mutation
        child(p,:) = child_3(1,1 : M + V);
        was_mutation = 0;
        p = p + 1;
    end
end
f = child;

[N, m] = size(intermediate_chromosome);

% Get the index for the population sort based on the rank
[temp,index] = sort(intermediate_chromosome(:,M + V + 1));

clear temp m

% Now sort the individuals based on the index
for i = 1 : N
    sorted_chromosome(i,:) = intermediate_chromosome(index(i),:);
end

% Find the maximum rank in the current population
max_rank = max(intermediate_chromosome(:,M + V + 1));

% Start adding each front based on rank and crowing distance until the
% whole population is filled.
previous_index = 0;
for i = 1 : max_rank
    % Get the index for current rank i.e the last the last element in
the
    % sorted_chromosome with rank i.
    current_index = max(find(sorted_chromosome(:,M + V + 1) == i));
    % Check to see if the population is filled if all the individuals
with
    % rank i is added to the population.
    if current_index > pop
        % If so then find the number of individuals with in with
current
        % rank i.
        remaining = pop - previous_index;
        % Get information about the individuals in the current rank i.
        temp_pop = ...
            sorted_chromosome(previous_index + 1 : current_index, :);
        % Sort the individuals with rank i in the descending order
based on
        % the crowding distance.
        [temp_sort,temp_sort_index] = ...
            sort(temp_pop(:, M + V + 2),'descend');
        % Start filling individuals into the population in descending
order
```

```
        % until the population is filled.
        for j = 1 : remaining
            f(previous_index + j,:) = temp_pop(temp_sort_index(j),:);
        end
        return;
    elseif current_index < pop
        % Add all the individuals with rank i into the population.
        f(previous_index + 1 : current_index, :) = ...
            sorted_chromosome(previous_index + 1 : current_index, :);
    else
        % Add all the individuals with rank i into the population.
        f(previous_index + 1 : current_index, :) = ...
            sorted_chromosome(previous_index + 1 : current_index, :);
        return;
    end
    % Get the index for the last added individual.
    previous_index = current_index;
end
```

References

1) Yoshikazu Sawaragi, Hirotaka Nakayama and Tetsuzo Tanino (1976). Theory of multi-objective optimization.
2) Matthias Ehrgott. Multi-criteria optimization, (2).
3) A.E Eiben-J.Esmith. Introduction to Evolutionary computing.
4) Xin Yoo (2004). Application of multi-objective evolutionary algorithm. Vol 1.
5) Gabriele Elchfelder (2008). Adaptive scalarization Methods in multi-objective methods in optimization.
6) Carlos A.coello coello, Gary B.Lamont and David A.Van Veldhuizen. Evolutionary Algorithm for solving Multi-objective problems, (2).
7) Ajith Abrham, Lakhmi Jain and robert Goldberg. Evolutionary multi-objective optimization.
8) Eckart Zitzler, Marco Lamnanns and Stefan Bleuler. A Tutorial on evolutionary Multi-objectve optimization.
9) P. subbaraj, R. Rengaraj and S.Salivahanan (2009). Enhancement of economic dispatch problem using self-advanced real-coded genetic algorithm.
10) Javed Dhillon (2009). Multi-objective optimizationpower dispatch problem using NSAG-II.
11) Aravind Seshari. A fast elitist multi-objective genetic algorithm.

Appendix

TABLE A-1: Fuel cost coefficients for three generators system

units	c_i	b_i	a_i	P_{min}	P_{max}
1	0.001562	7.92	561	150	600
2	0.001940	7.85	310	100	400
3	0.004820	7.97	78	50	200

TABLE A-2:- No_x emission coefficients for three generators system

units	c_{Ni}	b_{Ni}	a_{Ni}
1	$1.4721848e^{-7}$	$-9.4868099e^{-5}$	0.00473254
2	$3.0207577e^{-7}$	$-9.7252878e^{-5}$	0.055821713
3	$1.9338531e^{-6}$	$-9.5373734e^{-4}$	0.027731524

TABLE A-3:- So_x emission coefficients for three generators system

units	c_{Si}	b_{Si}	a_{Si}
1	$1.6103e^{-6}$	0.00816466	0.5783298
2	$2.1999e^{-6}$	0.00891174	0.3515338
3	$5.4658e^{-6}$	0.00903782	0.0884504

TABLE A-4:- B-coefficients for three generators

0.000030 0.000010 0.000000	0.000000 0.000090 0.000000	0.000000 0.000000 0.000120

Source:-Javed Dhillon.