GENETIC ALGORITHM APPLIED ON MULTIOBJECTIVE OPTIMIZATION

ADDIS ABABA UNIVERSITY
COLLEGE OF COMPUTATIONAL AND NATURAL SCIENCES
DEPARTMENT OF MATHEMATICS

A project Submitted in partial fulfillment of the requirement of the degree of master of science in mathematics

By: Beletew Mekasha
Stream: Optimization
Advisor: Semu Mitiku(PhD)

June 17, 2014

ADDIS ABABA UNIVERSITY
DEPARTMENT OF MATHEMATICS

The undersigned hereby certify that they have read and recommend to the department of mathematics for acceptance of this project entitled "**GENETIC ALGORITHM APPLIED ON MULTIOBJECTIVE OPTIMIZATION**" by **Beletew Mekasha** in partial fulfillment of the requirements for the degree of Master of Science in mathematics.

Advisor: Dr. Semu Mitiku Kassa

Signature:_____

Date_____

Examiner 1: Dr. _____

Signature:_____

Date_____

Examiner 2: Dr. _____

Signature:_____

Date_____

Acknowledgment

I would like to express my deep sense of gratitude and indebtedness to my advisor, **Dr. Semu Mitiku**, for his continual encouragement and patient guidance during the early stages of chaos and confusions. I also thank him for indicating to me a future direction that I hope will continue working on. Without his support, expertise and guidance, this project's would never have come in to completion.

Finally,I would also like to express my appreciation to **my families** for their continuous encouragement.

Abstract

Multi-objective formulations are a realistic models for many complex optimization problems. In this project we presented multiobjective optimization problems using genetic algorithms developed specifically for the problems with multiple objectives. Customized genetic algorithms have been demonstrated to be particularly effective to determine excellent solutions(pareto-optimal points) to the problems. Moreover, in solving multi-objective problems, designers may be interested in a set of pareto-optimal points instead of a single point. Since genetic algorithms(GAs) work with a population of points, it seems natural to use GAs in multi-objective optimization problems to capture a number of solutions simultaneously. In this project we also describe the working principle of a binary-coded and real-parameter genetic algorithm, which is ideally suited to handle problems with a continuous search space.Moreover, a non-dominated sorting-based multi-objective evolutionary algorithm (MOEA), called non-dominated sorting genetic algorithm II (NSGA-II), is also presented.

Keywords: Generic Algorithm, Multi-objective Optimization, Elitism, Pareto optimal solutions, Ordering relation.

Contents

Introduction

The objective of this paper is to present an overview of multiple-objective optimization methods using genetic algorithms (GA). For multiple-objective problems, the objectives are generally conflicting, preventing simultaneous optimization of each objective. GA is inspired by the evolutionist theory explaining the origin of species.

In many real-life problems, objectives under consideration conflict with each other. Hence, optimizing X with respect to a single objective often results in unacceptable results with respect to the other objectives. Therefore, a perfect multi-objective solution that simultaneously optimizes each objective function is almost impossible. A reasonable solution to a multiobjective problem is to investigate a set of solutions, each of which satisfies the objectives at an acceptable level without being dominated by any other solution.

chapter 1 presents the basic terminology for use throughout the rest of the project. Furthermore, a historical overview of single-objective and multiobjective optimization is also discussed, together with a short introduction to evolutionary algorithm and genetic algorithm. Additionally, we also provide goal of multiobjective optimization problems.

Chapter 2 attempts to present the preference attitudes of the decision maker which play an essential role that specifies the meaning of optimality or desirability. In this chapter we will discuss the principles of multiobjective optimization and present optimality concepts for any solution to be optimal in the presence of multiple objectives. They are very often represented as binary relations on the objective space and are called preference orders. Furthermore, we discuss solution concepts for muIti-objective optimization problems and investigate some fundamental properties of solutions.

Additionally, in typical multi-objective optimization problem, there exist a set of solutions which are superior to the rest of solutions. In the search space when all objectives are considered but are inferior to other solutions in the space in one or more objectives. These solutions are known as pareto-optimal solution or non-dominated solutions[**?**]. As there are different algorithms for finding the minimum number from a finite set, some of approaches have also been mentioned in this project for finding the non-dominated set from a given population of solutions.

In chapter 3, The classical methods of multi-objective optimization are discussed. One way to solve multi-objective problems is to scalarize the vector of objectives in to one objective by averaging the objectives with a weight vector. This process allows a simpler optimization algorithm to be used, but the obtained solution largely depends on the choice

of the weight vector used in the scalarization process. In this chapter, we present the working principle of a binary-coded and real-parameter genetic algorithm operator.

Chapter 4 presents about Non-dominated Sorting Genetic Algorithm (NSGA-II) which carries out a non-dominated sorting of a combined parent and offspring population. Thereafter, starting from the best non-dominated solutions, each front is accepted until all population slots filled. This makes the algorithm an elitist type. For the solutions of the last allowed front, a crowded distance-based niching strategy is used to resolve which solutions are carried over to the new population.

The first multi-objective GA, called Vector Evaluated Genetic Algorithms (or VEGA), was proposed by Schaffer[**?**]. Afterward, several major multi-objective evolutionary algorithms were developed such as Multi-objective Genetic Algorithm (MOGA)[**?**], Niched Pareto Genetic Algorithm , Random Weighted Genetic Algorithm (RWGA), Non-dominated Sorting Genetic Algorithm (NSGA), Strength Pareto Evolutionary Algorithm (SPEA)[**?**], Pareto-Archived Evolution Strategy (PAES), Fast Non-dominated Sorting Genetic Algorithm (NSGA-II), Multi-objective Evolutionary Algorithm (MEA), Rank-Density Based Genetic Algorithm (RDGA).

Several survey papers have been published on evolutionary multi-objective optimization. This project takes a different course and focuses on important issues while designing a multi-objective GA and describes common techniques used in multi-objective GA to attain the goals in multi-objective optimization and we address the inclusion of an elite-preserving operator to make the algorithms better converge to the Pareto-optimal solutions by using fast and elitist Non-dominated Sorting Genetic Algorithm (NSGA-II).

Chapter 1

Preliminary

1.1 Definition and Preliminary Concepts

Optimization is the process of adjusting the inputs or characteristics of a device, mathematical process, or experiment to find the minimum or maximum output or result. The input consists of variables; the process or function is known as the cost function, objective function, or fitness function; and the output is the cost or fitness.

Optimization refers to finding one or more feasible solutions which correspond to extreme values of one or more objectives. The need for finding such optimal solutions in a problem comes mostly from the extreme purpose of either designing a solution for minimum possible cost of fabrication, or maximum possible reliability, or others. Because such extreme properties of optimal solutions, optimization methods are of great importance in practice[**?**].

Definition 1.1.1. Decision Variables: The decision variables are the numerical quantities for which values are to be chosen in an optimization problem. These quantities are denoted as x_j , $j = 1, 2, \cdots, n$. The vector X of n decision variables is represented by:

$$X = [x_1, x_2, \cdots, x_n]^T$$

Definition 1.1.2. Constraints: In most optimization problems there are always restrictions imposed by the particular characteristics of the environment or available resources (e.g., physical limitations, time restrictions, etc.). These restrictions must be satisfied in order to consider a certain solution acceptable. All these restrictions in general are called constraints, and they describe dependence among decision variables and constants (or parameters) involved in the problem. These constraints are expressed in the form of mathematical inequalities*:*

$$g_i(x) \leq 0, i = 1, \cdots, m$$

or equalities:

$$h_j(x) = 0, j = 1, \cdots, p$$

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

Given an n-dimensional decision variable vector $x = [x_1, \cdots, x_n]$ in the solution space X, find a vector x^* that minimizes a given set of K objective functions: $z(x^*) = [z_1(x^*), \cdots, z_K(x^*)]$. The solution space X is generally restricted by a series of constraints and bounds on the decision variables

In many real-life problems with more than one criteria, objectives under consideration conflict with each other. Hence, optimizing x with respect to a single objective often results in unacceptable value with respect to other objectives. Therefore, a perfect multi-objective solution that simultaneously optimizes each objective function is almost impossible. A reasonable solution to a multi-objective problem is to investigate a set of solutions, each of which satisfies the objectives at an acceptable level without being dominated by any other solution.

- **Dominance:** If all objective functions are to be minimized, a feasible solution x is said to dominate another feasible solution y ($x \succ y$), if and only if, $z_i(x) \leq z_i(y)$ for $i = 1, \cdots, k$ and $z_j(x) < z_j(y)$ for at least one index j.
- **Pareto optimal:** A solution is said to be Pareto optimal if it is not dominated by any other solution in the solution space. A Pareto optimal solution cannot be improved with respect to any objective without worsening at least one other objective. The set of all feasible non-dominated solutions in X is referred to as the Pareto optimal set, and for a given Pareto optimal set, the corresponding objective function values in the objective space is called the Pareto front.
- **Non conflicting:**If the objective functions are not conflicting to each other, the cardinality of the pareto-optimal set is one. This means that the minimum solution corresponding to any objective function is the same.

1.2 Single and multi-objective optimization

Single-objective optimization: When an optimization problem modeling a physical system involves only one objective function, the task of finding the optimal solution is called *single-objective optimization*[**?**].

Definition 1.2.1. (General Single-Objective Optimization Problem) : A general single-objective optimization problem is defined as minimizing (or maximizing) $f(x)$ subject to $g_i(x) \leq 0, i = 1, \cdots, m$, and $h_j(x) = 0, j = 1, \cdots, p$ $x \in \Omega$. A solution minimizes (or maximizes) the scalar $f(x)$ where x is a n-dimensional decision variable vector $x = (x_1, \cdots, x_n)$ from some universe Ω.

Observe that $g_i(x) \leq 0$ and $h_j(x) = 0$ represent constraints that must be fulfilled while optimizing (minimizing or maximizing) $f(x)$. Ω contains all possible x that can be used to satisfy an evaluation of $f(x)$ and its constraints. Of course, x can be a vector of continuous or discrete variables as well as f being continuous or discrete.

Multi-objective optimization: When an optimization problem involves more than one objective function, the task of finding one or more optimum solutions is known as *multi-objective optimization.*

For multiple-objective problems, the objectives are generally conflicting, preventing simultaneous optimization of each objective or finding a multi-dimensional Pareto-optimal front. As in a single-objective optimization problem, the multi-objective optimization problem may contain a number of constraints which any feasible solution (including all optimal solutions) must satisfy.

Definition 1.2.2. A general multi-objective optimization problem: A multi-objective optimization problem is defined as minimizing (or maximizing) $F(x) = (f_1(x), \cdots, f_k(x))$ subject to $g_i(x) \leq 0$, $i = 1, \cdots, m$, and $h_j(x) = 0$, $j = 1, \cdots, p$ $x \in \Omega$. A multi-objective optimization problem solution minimizes (or maximizes) the components of a vector $F(x)$ where x is a n-dimensional decision variable vector $x = (x_1, \cdots, x_n)$ from some universe Ω.

It is noted that $g_i(x) \leq 0$ and $h_j(x) = 0$ represent constraints that must be fulfilled while minimizing (or maximizing) $F(x)$ and Ω contains all possible x that can be used to satisfy an evaluation of $F(x)$. The optimal solutions in multi objective optimization can be defined from a mathematical concept of partial ordering. In the parlance of multi-objective optimization, the term domination is used for this purpose.

There are two general approaches to multiple-objective optimization. One is to combine the individual objective functions into a single composite function. Determination of a single objective is possible with methods such as utility theory, weighted sum method, etc., but the problem lies in the correct selection of the weights or utility functions to characterize the decision-makers preferences.

The second general approach is to determine an entire Pareto optimal solution set or a representative subset. A Pareto optimal set is a set of solutions that are non-dominated with respect to each other. While moving from one Pareto solution to another, there is always a certain amount of sacrifice in one objective to achieve a certain amount of gain in the other. Pareto optimal solution sets are often preferred to single solutions because they can be practical when considering real-life problems, since the final solution of the decision maker is always a trade-off between crucial parameters.

1.2.1 Difference between single and multi-objective optimization

There are a number of fundamental differences between single-objective and multi-objective optimization. These are[**?**]:

- **Two goals instead of one:** In single-objective optimization, there is one goal-the search for an optimum solution. Although the search space may have a number of local optimal solutions, the goal is always to find the global optimum solution. However, most single-objective optimization algorithms aim at finding one optimum solution even when there exist a number of optimal solutions.

However, in multi-objective optimization, there are clearly two goals. Progressing towards the pareto-optimal front is certainly an important goal. However, maintaining a diverse set of solutions in the non-dominated front is also essential. The achievement of one goal does not necessarily achieve the other goal. Explicit or implicit mechanisms to emphasize convergence near the pareto-optimal front and the maintenance of a diverse set of solutions most be introduced in an algorithm. Because of these dual tasks, multi-objective optimization is more difficult than single-objective optimization.

- **Dealing with two search spaces:** Another difficulty is that a multi-objective optimization involves two search spaces, instead of one. In single-objective optimization, there is only one search space, the decision variable space. An algorithm works in this space by accepting and rejecting solutions based on their objective function values. Here, in addition to the decision variable space there also exist the objective or criterian space. Although these two spaces are related by unique mapping between them, often the mapping is nonlinear and the properties of the two search spaces are not similar.

- **No artificial fix-ups:** Multi-objective optimization for finding multiple pareto-optimal solutions eliminate all fix-ups and can, in principle , find a set of optimal solutions corresponding to different weight and ε-vectors. The avoidance of multiple simulation runs, no artificial fix-ups, availability of efficient population based optimization algorithms.

1.3 Objective in multi-objective optimization algorithms

The ultimate goal of a multi-objective optimization algorithm is to identify solutions in the Pareto optimal set. However, identifying the entire Pareto optimal set, for many multiobjective problems, is practically impossible due to its size. In addition, for many problems, especially for combinatorial optimization problems, proof of solution optimality is computationally infeasible. Therefore, a practical approach to multi-objective optimization is to investigate a set of solutions *(the best-known Pareto set)* that represent the Pareto optimal set as much as possible. With these concerns in mind, a multi-objective optimization approach should achieve the following three conflicting goals:

- The best-known Pareto front should be as close possible as to the true Pareto front. Ideally, the best-known Pareto set should be a subset of the Pareto optimal set.

- To find a set of solutions as diverse as possible. In addition to being converged close to the parto-optimal front, they must also be sparsely spaced in the pareto-optimal region. Only with a diverse set of solutions, can we be assured of having a good set of trade-off solutions among objectives [**?**].

- In addition, the best-known Pareto front should capture the whole spectrum of the Pareto front. This requires investigating solutions at the extreme ends of the objective function space.

1.4 Evolutionary Algorithms

The potential of evolutionary algorithms for solving multiobjective optimization problems was hinted as early as the late 1960s by Rosenberg in his PhD thesis[**?**]. Evolutionary algorithm is characterized by a population of solution candidates and the reproduction process enables the combination of existing solutions to generate new solutions. This enables finding several members of the Pareto-optimal set in a single run instead of performing a series of separate runs, which is the case for some of the conventional stochastic processes.

Evolutionary algorithms are based on the principle of evolution, i.e. survival of the fittest. Unlike classical methods, they do not use a single search point but a population of points called individuals. Each individual represents a potential solution to the problem. In these algorithms, the population evolves toward increasingly better regions of the search space by undergoing statistical transformations called recombination, mutation and selection.

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 are:

a. **Representation (definition of individuals)** 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. 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.

b. **Evaluation function (or fitness function)** Typically, this function is composed of a quality measure in the phenotype space and the inverse representation. The evaluation function is commonly called the fitness function in Evolutionary Algorithms.

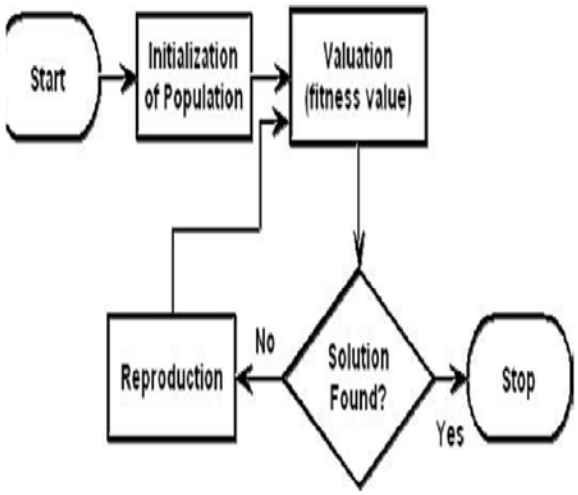
c. **Population** The role of the population is to satisfy (the representation of) possible solutions. Given a representation, defining a population can be as simple as specifying how many individuals are in it, that is. In almost all Evolutionary Algorithms applications the population size is constant 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.

d. **Parent selection mechanism** 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. 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.

e. **Recombination and mutation** The individuals chosen by selection recombine with each other and new individuals will be created. The aim is to get offspring individuals, that inherit the best possible combination of the characteristics (genes) of their parents. By means of random change of some of the genes, it is guaranteed that even if none of the individuals contain the necessary gene value for the extremum, it is still possible to reach the extremum.

f. **Survivor selection mechanism (replacement)** Between all individuals in the current population are chose those, who will continue and by means of crossover and mutation will produce offspring population.The best n individuals are directly transferred to the next generation.



Figuer 1.1: Flowchart of evolutionary algorithm iteration. Adopted from: - [**?**]

1.4.1 Genetic Algorithms

The genetic algorithm(GA), as the name implies, mimic the biological evolution process to find the fittest set of parameters. This application of biological process, named as genetic algorithm was developed by Holland and his colleagues in the 1960s and 1970s[**?**]. GA searches the solution space in multiple and random directions. GA is observed to be an effective algorithm for highly nonlinear solution spaces since it is not typically trapped in local optima. The GAs differ from most optimization techniques because of their global searching from a population of solutions rather than from one single solution.

GA Operators: The simplest form of genetic algorithm involves three types of operators: selection, crossover and mutation.

i **Selection:** Select a new population on the basis of the assigned fitness.

ii **Crossover:** Make pairs randomly and a crossover for each pair according to a given crossover rate (probability) to create two offspring.

iii **Mutation:** Mutate each individual according to a given mutation rate (probability).

The evolutionary algorithms use the three main principles of the natural evolution which is reproduction, natural selection and diversity of the species, maintained by the differences of each generation with the previous.

Genetic Algorithms works with a set of individuals, representing possible solutions of the task. The selection principle is applied by using a criterion, giving an evaluation for the individual with respect to the desired solution. The best-suited individuals create the next generation.

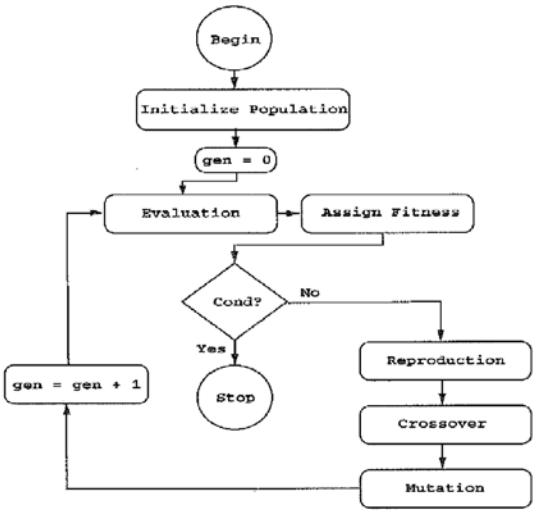


Figure 1.2: flowchart of the working principle of a GA. Adopted from:- [**?**]

In its general form, genetic algorithm(GA) work through the following steps:

(1) Creation of a random initial population of N_p potential solutions to the problem and evaluation of these individuals in terms of their fitness, i.e. of their corresponding objective function values;

(2) Check for termination of the algorithm :- As in the most optimization algorithms, it is possible to stop the genetic optimization by:

- Value of the function:- The value of the function of the best individual is within defined range around a set value. However, it is not recommended to use this criterion alone, because of the stochastic element in the search the procedure, the optimization might not finish within sensible time;
- Maximal number of iterations:- this is the most widely used stopping criteria. It guarantees that the algorithms will give some results within some time, whenever it has reached the extremum or not;

- Stall generation:- if within initially set number of iterations (generations) there is no improvement of the value of the fitness function of the best individual the algorithms stops.

(3) selection of a pair of individuals as parents;

(4) crossover of the parents, with generation of two children;

(5) genetic mutation;

(6) replacement in the population, so as to maintain the population number N_p constant.

Chapter 2

Multi-objective optimization

2.1 Multi-objective optimization Formulation

A multi-objective optimization problem has a number of objective function which are to be minimized or maximized. As in the single-objective optimization problem, here too the problem usually has a number of constraints which only feasible solution (including the optimal solution) satisfy them. The general form of multi-objective optimization(MOOP) is[**?**]:

$$Minimize/Maximize \quad f_m(x), \quad m = 1, 2, \cdots, M; \tag{2.1}$$

$$\begin{array}{lll} \text{Subject to} & g_j(x) \geq 0, & j = 1, 2, \cdots, J; \\ & h_k(x) = 0, & k = 1, 2, \cdots, K; \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)} & i = 1, 2, \cdots, n. \end{array}$$

A solution x is a vector of n decision variables; $x = (x_1, \cdots, x_n)$. The last set of constraints are called variable bounds, restricting each decision variable x_i to take a value with in a lower x_i^L and upper x_i^U bound. These bounds constitute a decision space.

Associated with the problem are J inequality and K equality constrains. The terms $g_j(x)$ and $h_k(x)$ are called constraint functions. A solution x that does not satisfy all of the $(J + K)$ constraints and all of the $2N$ variable bounds stated above is called an infeasible solution. On the other hand, if any solution x satisfies all constraints and variable bounds, it is known as a feasible solution.

There are M objective functions $f(x) = (f_1(x), \cdots, f_m(x))^T$ considered in the above formulation. Each objective function can be either minimized or maximized. The duality principle, in the context of optimization suggests that we can convert a maximization problem in to a minimization one by multiplying the objective function by -1[**?**].

For each solution x in the decision variable space(i.e., the space, of which the feasible set X is a subset, is called the decision space), there exists a point in the *objective space*, denoted by $f(x) = z = (z_1, z_2, \cdots, z_m)^T$. The mapping takes place between an $n-$ dimensional solution vector and an $m-$ dimensional objective vector. Multi-objective optimization is sometimes

referred to as vector optimization, because a vector of objectives, instead of a single objective, is optimized[**?**].

Linear and Nonlinear Multi-objective Optimization Problem: If all objective functions and constraint functions are linear, the resulting multi-objective optimization problem is called a multi-objective linear program(MOLP). Like the linear programming problems, MOLPs also have many theoretical properties. However, if any of the objective or constraint functions are nonlinear, the resulting problem is called a nonlinear multi-objective problem[**?**].

2.1.1 Convex and Non convex Multi-Objective Optimization Problem

Before we discuss a convex multi-objective optimization problem, let us first define a convex function:

Definition 2.1.1. A subset S of $\mathbb{R}^n$ is said to be convex if for any two pair of element $x^1, x^2 \in \mathbb{R}^n$, the following condition is true:

$$\lambda x^1 + (1-\lambda)x^2 \in S$$

for all $0 \leq \lambda \leq 1$.

Definition 2.1.2. A function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is a convex function if for any two pair of solution $x^1, x^2 \in \mathbb{R}^n$, the following condition is true:

$$f(\lambda x^1 + (1-\lambda)x^2) \leq \lambda f(x^1) + (1-\lambda)f(x^2) \tag{2.2}$$

for all $0 \leq \lambda \leq 1$.

The above definition gives rise to the following properties of a convex function:

- The linear approximation of $f(x)$ at any point in the interval $[x^1, x^2]$ always underestimates the actual function value.
- The Hessian matrix of $f(x)$ is positive definite for all x.
- For a convex function, a local minimum is always a global minimum.

A function satisfying the inequality shown in (**??**) with a $'>'$ sign instead of a $'\leq'$ sign is called a non-convex function. To test it a function is convex with in an interval, the Hessian matrix $\bigtriangledown^2 f$ is calculated and checked for its positive-definiteness at all points in the interval. One of the ways to check the positive-definiteness of a matrix is to compute the eigenvalues of the matrix and check to see if all eigenvalues are positive. To test if a function is non-convex in an interval,the Hessian matrix $-\bigtriangledown^2 f$ is checked for its positive-definiteness[**?**].

Definition 2.1.3. A multi-objective optimization problem is convex if all its objective functions are convex and the feasible region is convex[**?**].

Proposition 2.1. Let X be a convex set in $\mathbb{R}^n$, $f(x) = (f_1(x), \cdots, f_k(x))$ and $f : \mathbb{R}^n \longrightarrow \mathbb{R}^k$ is $\mathbb{R}^k_+$ convex if and only if the functions f_i are convex,for all $i = 1, 2, \cdots, k$.

Proof. ($\Rightarrow$) : let f is $\mathbb{R}^k_+$ is convex and $x, y \in X$ for all $\lambda \in [0,1]$
$\Rightarrow \lambda f(x) + (1-\lambda) f(y) - f(\lambda x + (1-\lambda)y) \in \mathbb{R}^k_+$

$\Rightarrow \lambda(f_1(x), \cdots, f_k(x)) + (1-\lambda)(f_1(y), \cdots, f_k(y)) - [f_1(\lambda x + (1-\lambda)y), \cdots, f_k(\lambda x + (1-\lambda)y)]$

$\Rightarrow (\lambda f_1(x), \cdots, \lambda f_k(x)) + ((1-\lambda) f_1(y), \cdots, (1-\lambda) f_k(y)) - [f_1(\lambda x + (1-\lambda)y), \cdots, f_k(\lambda x + (1-\lambda)y)]$

$\Rightarrow [\lambda f_1(x) + (1-\lambda) f_1(y), \cdots, \lambda f_k(x) + (1-\lambda) f_k(y)] - [f_1(\lambda x + (1-\lambda)y), \cdots, f_k(\lambda x + (1-\lambda)y)]$

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

$\Rightarrow f_i$ is convex.
($\Leftarrow$) Let f_i is convex for all $i = 1, \cdots, k$
$\Rightarrow \lambda f_i(x) + (1-\lambda) f_i(y) - [f_i(\lambda x + (1-\lambda)y)] \in \mathbb{R}_+, \forall (i = 1, \cdots, k)$ and $\forall \lambda \in [0,1]$

$\Rightarrow [\lambda f_1(x) + (1-\lambda) f_1(y), \cdots, \lambda f_k(x) + (1-\lambda) f_k(y)] - [f_1(\lambda x + (1-\lambda)y), \cdots, f_k(\lambda x + (1-\lambda)y)]$

$\Rightarrow (\lambda f_1(x), \cdots, \lambda f_k(x)) + ((1-\lambda) f_1(y), \cdots, (1-\lambda) f_k(y))) - [f_1(\lambda x + (1-\lambda)y), \cdots, f_k(\lambda x + (1-\lambda)y)]$

$\Rightarrow \lambda(f_1(x), \cdots, f_k(x)) + (1-\lambda)(f_1(y), \cdots, f_k(y)) - [f_1(\lambda x + (1-\lambda)y), \cdots, f_k(\lambda x + (1-\lambda)y)]$

$\Rightarrow \lambda f(x) + ()1 - \lambda) f(y) - f(\lambda x + (1-\lambda)y) \in \mathbb{R}^k_+$
$\Rightarrow f$ is $\mathbb{R}^k_+$ - convex function. □

2.2 Ordering Relations and Dominance Structure

First we introduce binary relations and some of their properties to define several classes of orders. That is, let S be any set. A binary relation on S is a subset R of $S \times S$. Some properties of binary relations are[**?**]:

Definition 2.2.1. A binary relation R on S is called

- *reflexive if* $(x, x) \in R$ *for all* $x \in S$,
- *irreflexive if* $(x, x) \notin R$ *for all* $x \in S$,
- *symmetric if* $(x, y) \in R \Rightarrow (y, x) \in R$ *for all* $x, y \in S$,
- *asymmetric if* $(x, y) \in R \Rightarrow (y, x) \notin R$ *for all* $x, y \in S$,
- *antisymmetric if* $(x, y) \in R$ *and* $(y, x) \in R \Rightarrow x = y$ *for all* $x, y \in S$,
- *transitive if* $(x, y) \in R$ *and* $(y, z) \in R \Rightarrow (x, z) \in R$ *for all* $x, y, z \in S$,
- *negatively transitive if* $(x, y) \notin R$ *and* $(y, z) \notin R \Rightarrow (x, z) \notin R$ *for all* $x, y, z \in S$,
- *connected if* $(x, y) \in R$ *or* $(y, x) \in R$ *for all* $x, y \in S$ *with* $x \neq y$,
- *strongly connected (or total) if* $(x, y) \in R$ *or* $(y, x) \in R$ *for all* $x, y \in S$.

Definition 2.2.2. A binary relation R on a set S is

- *an equivalence relation if it is reflexive, symmetric, and transitive,*
- *a preorder (quasi-order) if it is reflexive and transitive.*

Instead of $(x, y) \in R$ we shall also write xRy. In the case of R being a preorder the pair (S, R) is called a preordered set. In the context of (pre)orders yet another notation for the relation R is convenient. We shall write $x \preceq y$ as shorthand for $(x, y) \in R$ and $x \npreceq y$ for $(x, y) \notin R$ and indiscriminately refer to the relation R or the relation $\npreceq$. This notation can be read as "preferred to. "Actually, $\prec$ and $\sim$ can be seen as the strict preference and equivalence (or indifference) relation, respectively.

Definition 2.2.3. A binary relation $\preceq$ is called

- *partial order if it is reflexive, transitive and antisymmetric,*
- *strict partial order if it is asymmetric and transitive (or, equivalently, if it is irreflexive and transitive).*

Let x and y be a vector on $\mathbb{R}^n$, then some of the frequently used orders on $\mathbb{R}^n$ are:

Notation	Definition	Name
$x \leqq y$	$x_k \leqslant y_k,\ k = 1, \cdots, n$	weak componentwise order
$x \leqslant y$	$x_k \leqslant y_k, k = 1, \cdots, n; x \neq y$	componentwise order
$x < y$	$x_k < y_k, k = 1, \cdots, n$	strict componentwise order
$x <_{lex} y$	$x_{k^*} < y_{k^*}$ or $x = y$ for some k^*	lexicographic order

Table 2.1: Some of orders on $\mathbb{R}^n$

Definition 2.2.4. A subset $C \subseteq \mathbb{R}^p$ is called a cone, if $\alpha d \in C$ for all $d \in C$ and for all $\alpha \in \mathbb{R}$, $\alpha > 0$.

Definition 2.2.5. A cone C in $\mathbb{R}^p$ is called

- *nontrivial or proper if $C \neq \emptyset$ and $C \neq \mathbb{R}^p$,*
- *convex if $\alpha d_1 + (1 - \alpha)d_2 \in C$ for all $d_1, d_2 \in C$ and for all $0 < \alpha < 1$,*
- *pointed if for $d \in C, d \neq 0, -d \notin C$, i.e., $C \bigcap (-C) \subseteq \{0\}$.*

Due to the definition of a cone, C is convex if for all $d_1, d_2 \in C$ we have $d_1 + d_2 \in C$, too: $\alpha d_1 \in C$ and $(1 - \alpha)d_2 \in C$ because C is a cone.

A preference order: A preference order represents a preference attitude of the decision maker in the objective space. It is a binary relation on a set $Y = f(X) \subset \mathbb{R}^P$, where f is the vector-valued objective function and X the feasible decision set[**?**]. The basic binary relation $\succ$ means strict preference; that is, $y \succ z$ for $y, z \in Y$ implies that the result (objective value) y is preferred to z. From this, we may define two other binary relations $\sim$ and $\succsim$ as

$$y \sim z \text{ if and only if not } y \succ z \text{ and not } z \succ y,$$

$$y \precsim z \text{ if and only if } y \succ z \text{ or } y \sim z.$$

The relation $\sim$ is called indifference (read $y \sim z$ as y is indifferent to z), and $\precsim$ is called preference-indifference (read $y \precsim z$ as z is not preferred to y).

Definition 2.2.6. (Preference Function)
Given the preference order $\succ$ on a set Y, a real-valued function u on Y such that

$$u(y) > u(z) \text{ if and only if } y \succ z \text{ for all } y, z \in Y$$

is called a preference function.

Remark 2.1. Note the following relationships:

$$y \sim z \Longleftrightarrow \textit{not } y \succ z \textit{ and not } z \succ y \Longleftrightarrow u(y) = u(z),$$

$$y \precsim z \Longleftrightarrow y \succ z \textit{ or } y \sim z \Longleftrightarrow u(y) \geqq u(z).$$

Definition 2.2.7. (Efficiency)
Let Y be a feasible set in the objective space $\mathbb{R}^P$, and let $\succ$ be a preference order on Y. Then an element $y^{'} \in Y$ is said to be an efficient (noninferior) element of Y with respect to the order $\succ$ if there does not exist an element $y \in Y$ such that $y \succ y^{'}$. The set of all efficient elements is denoted by $\xi(Y, \succ)$. That is,

$$\xi(Y, \succ) = \{y^{'} \in Y | \nexists y \in Y : y \succ y^{'}\}.$$

Domination Structures: Preference orders (and more generally, binary relationships) on a set Y can be represented by a point-to-set map from Y to Y. In fact, a binary relationship may be considered to be a subset of the product set $Y \times Y$, and so it can be regarded as a graph of a point-to-set map from $Y$to Y. Namely, we identify the preference order $\succ$ with the graph of the point-to-set map $P[\mathbf{?}]$:

$$P(y) = \{y^{'} \in Y : y \succ y^{'}\}$$

$[P(y)$ is a set of the elements in Y less preferred to $y]$,

Another way of representing preference orders by point-to-set maps is the concept of domination structures. For the preference $\succ$ and for $y \in Y \subset \mathbb{R}^P$, the set

$$D^{+}(y) := \{d \in \mathbb{R}^P : y + d \succ y\}$$

is called the domination set for y, and

$$D^{-}(y) := \{d \in \mathbb{R}^P : y \succ y + d\}$$

is called the dominated set for y. In addition,

$$I(y) := \{d \in \mathbb{R}^P : y \sim y + d\}$$

is called the indifference set for y.
In our multiobjective optimization problem(MOOP) in which each objective function is to be minimized, the domination set for Pareto order $\leqslant$ is given by

$$D^+(y) = \mathbb{R}^P_- \setminus \{0\}$$

for any $y \in \mathbb{R}^P$.
For each $y \in Y \subset \mathbb{R}^P$, we define the set of domination factors

$$D(y) := \{d \in \mathbb{R}^P : y \succ y + d\} \cup \{0\}.$$

This means that deviation $d \in D(y)$ from y is less preferred to the original y. Then the point-to-set map D from Y to $\mathbb{R}^P$ clearly represents the given preference order. We call D the domination structure.

Definition 2.2.8. Given a set Y in $\mathbb{R}^P$ and a domination structure $D(.)$, the set of efficient elements is defined by

$$\xi(Y, D) = \{y^{'} \in Y | \nexists y \in Y : y \in y + D(y)\}.$$

This set $\xi(Y, D)$ is called the efficient set with respect to the domination structure D.

The most important and interesting special case of the domination structures is when $D(.)$ is a constant point-to-set map, particularly when $D(y)$ is a constant cone for all y. When $D(y) = D(acone)$, the domination structure $D(.)$ is said to be

- asymmetry $\Longleftrightarrow [d \in D, d \neq O \Longrightarrow -d \notin D]$
- transitivity $\Longleftrightarrow [d, d^{'} \in D \Longrightarrow d + d^{'} \in D]$.

Pointed convex cones are often used for defining domination structures. We usually write $y \leqq_D y^{'}$ for $y,\ y^{'} \in \mathbb{R}^p$ if and only if $y^{'} - y \in D$ for a convex cone D in $\mathbb{R}^p$. Also $y \leq_D y^{'}$ means that $y^{'} - y \in D$ but $y - y^{'} \notin D$. When D is pointed, $y \leq_D y^{'}$ if and only if $y^{'} - y \in D \setminus \{0\}$. When $D = \mathbb{R}^P_+$, it is omitted as $\leqq$ or $\leq$. In other words,

$$y \leqq y^{'} \text{ if and only if } y_i \leqq y_i^{'} \text{ for all } i = 1, \cdots, p,$$

$$y \leq y^{'} \text{ if and only if } y \leqq y^{'} \text{ and } y \neq y^{'},$$

i.e.,

$$\begin{aligned} &y_i \leqq y_i^{'} \text{ for all } i = 1, \cdots, p, \\ &y_i < y_i^{'} \text{ for some } i \in \{1, \cdots, p\}. \end{aligned}$$

2.3 Solution Concept

The concept of optimal solutions to multi-objective optimization problems is not trivial and in itself debatable. It is closely related to the preference attitudes of the decision makers. The most fundamental solution concept is that of efficient solutions (also called non-dominated solutions or non-inferior solutions) with respect to the domination structure of the decision maker[**?**].

We consider the multi-objective optimization problem (P) minimize

$$f(x) = (f_l(x), f_2(x), \cdots, f_p(x)$$

subject to $x \in X \subset \mathbb{R}^n$. Let, $Y = f(X) = \{y : y = f(x), x \in X\}$.
A domination structure representing a preference attitude of the decision maker is supposed to be given as a point-to-set map D from Y into $\mathbb{R}^p$.

Definition 2.3.1. (efficient solution) A point $x^* \in X$ is said to be an efficient solution to the multi-objective optimization problem (P) with respect to the domination structure D if $f(x^*) \in \xi(Y, D)$; that is, if there is no $x \in X$ such that $f(x) \in f(x) + D(f(x))$ and $f(x) \neq f(x^*)$ (i.e., such that $f(x^*) \in f(x) + D(f(x)) \setminus \{O\}$).

Proposition 2.2. Given two domination structures D_1 and D_2, D_1 is said to be included by D_2, if

$$D_1(y) \subset D_2(y)$$

for all $y \in Y$. In this case, $\xi(Y, D_1) \supset \xi(Y, D_2)$

Proof. Let $y \in \xi(Y, D_2)$. There is no $y \in Y$ such that $f(y) \in f(y) + D_2(f(y) \setminus \{0\})$.
Since D_1 is included in D_2, there is no $y \in Y$, $f(y) \in f(y) + D_1(f(y) \setminus \{0\})$. This implies that $y \in \xi(Y, D_1)$. Therefore, $\xi(Y, D_1) \supset \xi(Y, D_2)$

Proposition 2.3. Let D be a nonempty cone containing 0, then

$$\xi(Y, D) \supset \xi(Y + D, D)$$

with equality holding if D is pointed and convex.

Proof. The result is trivial if Y is empty, so we assume otherwise. First suppose $y \in \xi(Y + D, D)$ but $y \notin \xi(Y, D)$. If $y \notin Y$, there exist $y^{'} \in Y$ and nonzero $d \in D$ such that $y = y^{'} + d$. Since $0 \in D, Y \subset Y + D$. Hence, $y \notin \xi(Y + D, D)$, which is a contradiction. If $y \in Y$, we directly have a similar contradiction.
Next suppose that D is pointed and convex, $y \in \xi(Y, D)$ but $y \notin \xi(Y + D, D)$. Then there exists a $y^{'} \in Y + D$ with $y - y^{'} = d^{'} \in D \setminus \{0\}$. Then $y^{'} = y^{''} + d^{''}$ with $y^{''} \in Y, d^{''} \in D$. Hence, $y = y^{''} + (d^{'} + d^{''})$ and $d^{'} + d^{''} \in D$, since D is a convex cone. Since D is pointed, $d^{'} + d^{''} \neq 0$ and so $y \notin \xi(Y, D)$, which leads to a contradiction. This completes the proof of the proposition.

Proposition 2.4. Let Y_1 and Y_2 be two sets in $\mathbb{R}^p$, and let D be a constant domination structure on $\mathbb{R}^p$ (a constant cone). Then

$$\xi(Y_1 + Y_2, D) \subset \xi(Y_1, D) + \xi(Y_2, D).$$

Proof. Let $y^* \in \xi(Y_1 + Y_2, D)$. Then $y^* = y^l + y^2$ for some $y^l \in Y_1$ and $y^2 \in Y_2$. We show that $y^l \in \xi(Y_1, D)$. If we suppose the contrary, then there exist $y \in Y_1$ and nonzero $d \in D$ such that $y^l = y + d$. Then $y^* = y^l + y^2 = y + y^2 + d$ and $y + y^2 \in Y_1 + Y_2$, which contradicts the assumption $y^* \in \xi(Y_1 + Y_2, D)$. Similarly we can prove that $y^2 \in \xi(Y2, D)$. Therefore, $y^* \in \xi(Y_1, D) + \xi(Y_2, D)$.

2.4 Dominance and Pareto Optimality

Most multi-objective optimization algorithms use the concept of dominance in their search. We first define some special solution which are often used in multi-objective optimization algorithms[**?**].

Ideal objective vector: For each of the M confilicting objective, there exists one different optimal solution. An objective vector constructed with these individual optimal objective values consitutes the ideal objective vector[**?**].

Definition 2.4.1. The m^{th} component of the ideal objective vector z^* is the constrained minimum solution of the following problem;

$$\begin{array}{l} \textit{minimize } f_m(x) \\ \textit{subject to } x \in S \end{array}$$

Thus, if the minimum solution for the m^{th} objective function is the decision vector $x^{*(m)}$ with function value f_m^*, the ideal vector is as follows.

$$z^* = f^* = (f_1^*, f_2^*, \cdots, f_m^*)$$

Nadir objective vector: The nadir objective vector, z^{nad}, represents the upper bound of each objective in the entire pareto-optimal set, and not in the entire search space.

In order to normalize each objective in the entire range of the pareto-optimal region, the knowledge of nadir and ideal objective vectors can be used as follows

$$f_i^{norm} = \frac{f_i - z_i^*}{z_i^{nad} - z_i^*}$$

Definition 2.4.2. A Solution x^1 is said to dominate the other solution x^2 , if both conditions 1 and 2 are true:

1. The solution x^1 is no worse than x^2 in all objective,
2. The solution x^1 is strictly better than x^2 in at least one objective.

If any of the above condition is violated, the solution x^1 does not dominate the solution x^2.

2.4.1 Pareto Optimality

For a given finite set of solutions, we can perform all possible pair-wise comparisons and which solutions which and which solutions are non-dominated with respect to each other. At the end, we expect to have a set of solutions, any two of which do not dominate each other.

Among a set of solution P, the non-dominated set of solutions P' are those that not dominated by any member of the set P. When the set P is the entire search space, the resulting non-dominated set P' is called the pareto-optimal set[**?**].

Definition 2.4.3.

- The non-dominated set of the entire feasible search space S is the globally pareto-optimal set.
- A solution x^1 strongly dominates a solution x^2, if the solution x^1 is strictly better than solution x^2 in all M objective.
- Among a set of solution P, the weakly non-dominated set of solutions P' are those that are not strongly dominated by any other member of the set P.
- A decision vector $x^* \in S$ is properly pareto-optimal if it is pareto-optimal and there exist some real number $M > 0$ such that for each f_i and each $x \in S$ satisfying $f_i(x) < f_i(x^*)$, there exists at least one f_j such that $f_j(x^*) < f_j(x)$ and

$$\frac{f_i(x^*) - f_i(x)}{f_j(x) - f_j(x^*)} \leq M$$

2.5 Procedures for Finding Non-dominated Set

Finding the non-dominated set of solutions from a given set of solutions is similar in principle to finding the minimum of a set of real numbers. In the case of finding the non-dominated set, the dominance relation $\preceq$ (or $\prec$) can be used to identify the better of two given solutions.

For finding the non-dominated set from a given population of solutions, there are many approaches, but the three of them are mentioned as follows:

Approach 1: Naive and slow:-[**?**]In this approach, each solution x_i is compared with every other solution in the population to check if it is dominated by any solution in the population. If the solution x_i is found to be dominated by any solution, this means that there exists at least one solution in the population which is better than x_i in all objectives. Hence the solution x_i can not belong to the non-dominated set. However, if no solution is fond to dominate solution x_i, it is a member of the non-dominated set. This is how any other solution in the population can be checked to see if it belongs to the non-dominated set from a finite set of arbitrary selected population. The following procedure describes a step-by-step for finding the non-dominated set in the given set P of size N.

Step 1 Set solution counter $i = 1$ and create an empty non-dominated set $P^{'}$.

Step 2 For a solution $x_j \in P$ (but $j \neq i$), check if solution x_j dominates solution x_i. If yes, go to step 4.

Step 3 If more solutions are left in P, increment j by one and go to step 2; otherwise, set $P^{'} = P^{'} \cup \{x_i\}$.

Step 4 Increment i by one. If $i \leq N$, go to step 2; otherwise, stop and declare $P^{'}$ as the non-dominated set.

Approach 2:Continuously updated:-[?]In this approach, every solution from the population is checked with a partially filled population for domination. To start with, the first solution from the population is kept in an empty set $p^{'}$. Thereafter, each solution x_i (the second solution on wards) is compared with all members of the set $p^{'}$, one by one. If the solution x_i dominates any member of $p^{'}$, then that solution is removed from $p^{'}$. In this way non-members of the non-dominated solutions get deleted from $p^{'}$. Otherwise, if solution x_i is dominated by any member of $p^{'}$, the solution x_i is ignored. If solution x_i is not dominated by any member of $p^{'}$, it is entered in $p^{'}$. This is how the set $p^{'}$ grows with non-dominated solutions. When all solutions of the population are checked,the remaining members of $p^{'}$ constitute the non-dominated set. The procedure is :

Step 1 Initialize $P^{'} = \{x_1\}$. Set solution counter $i = 2$.

Step 2 Set $j = 1$.

Step 3 Compare solution x_i with x_j from $P^{'}$ for domination.

Step 4 If i dominates j, delete the j^{th} member form $P^{'}$ or update $P^{'} = P^{'} \setminus \{P^{'(j)}\}$. If $j < |P^{'}|$, increment j by one and then go to step 3. Otherwise, go to step 5. Alternatively, if the j^{th} member of $P^{'}$ dominates x_i, increment i by one and then go to step 2.

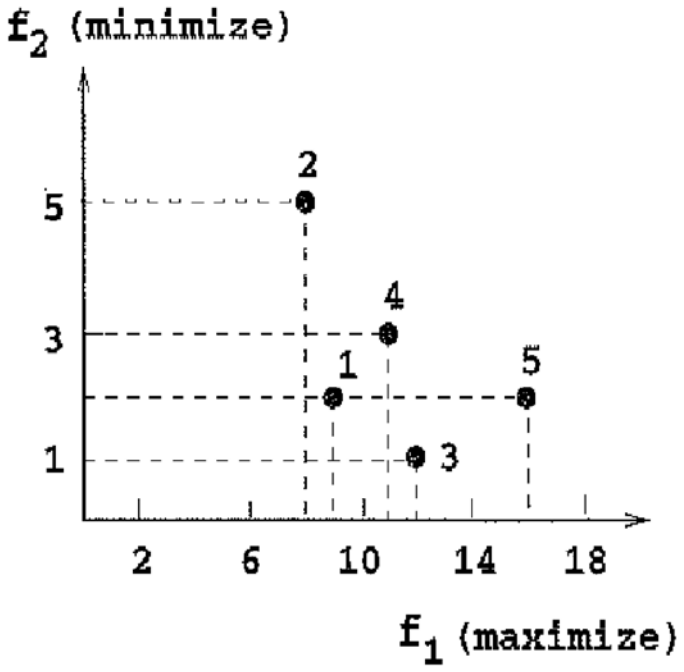
Step 5 Insert x_i in $P^{'}$ or update $P^{'} = P^{'} \cup \{x_i\}$. If $i < N$, increment i by one and go to step 2. Otherwise, stop and declare $P^{'}$ as the non-dominated set.

Approach 3: Kung et al.'s Efficient method:-[?]This approach first sorts the population according to the descending order of importance to the first objective function value. Thereafter, the population is recursively halved as top(T) and bottom(B) subpopulations. Knowing that the top-half of the population is better in terms of the first objective function, the bottom-half is then checked for domination with the top-half. The solutions of B that are not dominated by any member of T are combined with members of T to form a merged population M. The merging and the domination check starts with the innermost case (when there is only one member left in either T or B in recursive divisions of the population) and the proceeds in a bottom-up fashion. Generally the procedure of this approach is:

Step 1 Sort the population according to the descending order of importance to the first objective function and rename the population as P of size N.

Step 2 Front(P) (which is the list of the population that sorted according to the descending order of importance in the first objective function) if $|P| = 1$, return P as the output of Front (P). Otherwise, $T = Front(P^{(1)} - P^{|P|/2})$ and $B = Front(P^{((|P|/2)+1)} - P^{(|P|)})$. If the i^{th} solution of B is not dominated by any solution of T, create a merged set $M = T \cup \{x_i\}$. Return M as the output of Front(P).

Example 2.1. Let us consider a two-objective optimization problem with five different solutions shown in the objective space, as illustrated in Figure 2.1. Let us also assume that the objective function 1 needs to be maximized while the objective function 2 needs to be minimized. Five solutions with different objective function values are shown in this figure. We illustrate the working principle of the above stated approaches on the same set of five($N = 5$) solutions, as shown in Figure 2.1. Ideally, the exact objective vector for each solution will be used in executing the procedure, but here we use the figure to compare different solutions. We follow the procedure step-by-step in the following.



Figuer 2.1:A population of five solutions. Adopted from: - [?]

Identifying the non-dominated set:Approach 1.

step 1 We set $i = 1$ and $P' = \emptyset$.

step 2 We compare solution 1 with all other solutions for domination, starting from solution 2. We observe that solution 2 does not dominate solution 1. Since solution 1 is better than solution 2 in objective function 1 and solution 1 is also better than solution 2 in objective function 2. Thus, both of the conditions for domination are satisfied.

step 3 However, solution 3 dominates solution 1. Thus, we move to step 4.

step 4 Solution 1 does not belong to the non-dominated set and we increment i to 2 and move to step 2 to check the fate of solution 2.

step 2 We observe that solution 1 dominates solution 2. We therefore move to step 4.

step 4 Thus, solution 2 does not belong to the non-dominated set. Next, we check solution 3.

steps 2 and 3 Starting from solution 1, we observe that neither solution 1 nor 2 dominate solution 3. In fact, solutions 4 and 5 also do not dominates solution 3. Thus, we include solution 3 in the non-dominated set, $P^{'} = \{3\}$.

step 4 We now check solution 4.

step 2 Solution 5 dominates solution 4. Thus, it can not be a member of $P^{'}$.

step 4 Now we check the final solution (solution 5).

step 2 We observe that none of the solutions (1 to 4) dominates solution 5.

step 3 So, solution 5 also belongs to the non-dominated set. Thus, we update $P^{'} = \{3, 5\}$.

step 4 We have now considered all five solutions and the non-dominated set $P^{'} = \{3, 5\}$.

Identifying the non-dominated set:Approach 2.

step 1 $P^{'} = \{1\}$ and we set $i = 2$.

step 2 We set the solution counter of $P^{'}$ as $j = 1$ (which refers to solution 1).

step 3 We now compare solution 2($i = 2$) lone member of $P^{'}$ (solution 1) for domination. We observe that solution 1 dominates solution 2. Since the j^{th} member of $P^{'}$ dominates solution i, we we increment i to 3 and go to step 2. This means that solution 2 does not belong to the non-dominated set.

step 2 The set $P^{'}$ still has solution 1 ($j = 1$) only.

step 3 Now we compare solution 3 with solution 1. We observe that solution 3($i = 3$) dominates solution 1. Thus, we delete the j^{th} (or the first) member from $P^{'}$ and update $P^{'} = \emptyset$. Thus, $|P^{'}| = 0$. This depicts that a non-member of the non-dominated set gets deleted from $P^{'}$. We now move to step 5.

step 5 We insert $i = 3$ in $P^{'}$ or update $P^{'} = \{3\}$. Since $i < 5$ here, we increment i to 4 and move to step 2.

step 2 We set $j = 1$ which refers to the lone element (solution 3) of $P^{'}$.

step 3 By comparing solution 4 with solution 3, we observe that solution 3 dominates solution 4. Thus, we increment i to 5 and move to step 2.

step 2 We still have solution 3 in $P^{'}$.

step 3 Now we compare solution 5 with solution 3. we observe that neither of them dominates the other. Thus, we move to step 5.

step 5 We insert solution 5 in $P^{'}$ and update $P^{'} = \{3, 5\}$ as the non-dominated set.

2.5.1 Non-Dominated Sorting of a Population

Most evolutionary multi-objective optimization algorithms require us to find only the best non-dominated front in a population. These algorithms classify the population in to two sets, the non-dominated set and the remaining dominated set. In such algorithms, the population needs to be sorted according to an ascending level of non-dominated. The procedures are:

Step 1 Set all non-dominated sets P_j, $(j = 1, 2, \cdots)$ as empty sets. Set non-domination level counter $j = 1$

Step 2 Use any one of the approaches 1 to 3 to find the non-dominated set $P^{'}$ of population P.

Step 3 Update $P_j = P^{'}$ and $P = p \backslash \{P^{'}\}$.

Step 4 If $P \neq \emptyset$, increment j by one and go to step 2. Otherwise, stop and declare all non-dominated sets P_i, for $i = 1, 2, \cdots, j$.

Chapter 3

Classical Methods and Genetic Algorithm operator

3.1 Classical Methods

The task of finding multiple Pareto-optimal solutions is achieved by executing many independent single-objective optimization, each time finding a single Pareto-optimal solution. A parametric scalarizing approach (such as the weighted-sum approach, ϵ-constraint approach, and others) can be used to convert multiple objectives into a parametric single-objective function. By simply varying the parameters (weight vector or ϵ-vector) and optimizing the scalarized function, different Pareto-optimal solutions can be found[**?**].

3.1.1 ϵ-Constraints Method

Besides the scalarization approach, one of the solution technique to multi-objective optimization is the ϵ -constraints method proposed by Chankong and Haimes in 1983. The ϵ-constraint technique is based upon selecting a primary objective function and then bounding the others with a separate allowable ϵ-constraint (must be known a priori). The ϵ-constraints are then changed in order to generate another point on the Pareto front (phenotype) and so forth resulting in finding elements in the Pareto optimal set (genotype). Non-uniformity in the distribution of the Pareto front points usually occurs[**?**]. In mathematical terms, if we let $f_j(x)$ be the objective function chosen to be minimized, we have the following problem:

$$\begin{aligned} & min \quad f_j(x) \\ \text{s.t.} \quad & f_i(x) \leq \varepsilon_i, \forall i \in \{1, \cdots, n\} \backslash \{j\} \\ & x \in S, j \in \{1, \cdots, n\}, j \neq i \end{aligned} \tag{3.1}$$

Theorem 3.1. The solution of ϵ-Constraints problem(3.1) is weakly pareto-optimal[**?**].

Proof. Let $x^* \in S$ be a solution of the weighting problem. Let us suppose that it is not weakly pareto-optimal. In this case, there exists a solution $x \in S$ such that $f_i(x) < f_i(x^*)$ for all $i = 1, 2, \cdots, n$. This means that $f_j(x) < f_j(x^*) \leq \varepsilon_i$ for all $j = 1, 2, \cdots, , j \neq i$. Thus x is feasible with respect to the ϵ-Constraint problem. While on addition $f_j(x) < f_j(x^*)$,

we have a contradiction to the assumption that x^* is a solution of the ϵ-Constraint problem. Thus, x^* has to be weakly pareto-optimal.

Theorem 3.2. A decision vector $x^* \in S$ is pareto-optimal if and only if it is a solution of ϵ-Constraints problem(3.1) for every $j = 1, 2, \cdots, n$, where $\varepsilon_i = f_i(x^*)$ for $i = 1, 2, \cdots, n, i \neq j$.

Proof. Necessity: Let $x^* \in S$ be pareto-optimal. Let us assume that it does not solve the ϵ-Constraints problem For some j where $\varepsilon_i = f_i(x^*)$ for $i = 1, 2, \cdots, n$,
$j \neq i$. Then there exists a solution. $x \in S$ such that $f_j(x) < f_j(x^*)$ and $f_i(x) \leq f_i(x^*)$ when $j \neq i$. This contradicts the pareto-optimality of x^*.
Sufficiency: since $x^* \in S$ is by assumption a solution of the ϵ-Constraint problem for every $j = 1, 2, \cdots, n$ there is no $x \in S$ such that $f_j(x) < f_j(x^*)$ and $f_i(x) \leq f_i(x^*)$ when $j \neq i$. This is the definition of pareto-optimality for x^*.

Theorem 3.3. Let the multi-objective optimization problem be convex. If $x^* \in S$ is a solution of ϵ-Constraints problem(3.1) for any given f_j to be minimized and $\varepsilon_i = f_i(x^*)$ for $i = 1, 2, \cdots, n, j \neq i$. Then there exists a weighting vector $0 \leq w \in \mathbb{R}^n$,$\sum_{i=1}^n w_i = 1$, such that x^* is also a solution of weighting problem(3.2).

NOTE: The proof of this result relies on the generalized Gordon Theorem[**?**]. That is, Let f be an m-dimensional convex vector function on the convex set $X \subset \mathbb{R}^n$. Then either

I $f(x) < 0$ has a solution $x \in X$ or

II $pf(x) \geqq 0$ for all $x \in X$ for some $p \geq 0$, $p \in \mathbb{R}^m$ but never both.

Proof. Since x^* solves ϵ-Constraints problem(3.1) the system of inequalities

$$f_j(x) < f_j(x^*), f_i(x) \leq f_i(x^*), i = 1, 2, \cdots, n, j \neq i$$

has no solution in X.
Upon imposing the convexity assumption and applying the generalized Gordon Theorem, there exists $p \in \mathbb{R}^n$ with : $p \geq 0$ such that $\sum_{i=1}^n p_i[f_i(x) - f_i(x^*)] \geq 0$ for all $x \in X$. By choosing $w_i = p_i \diagup \sum_{i=1}^n p_i$ (possible since $\sum_{i=1}^n p_i > 0$), we have $w \geq 0$ and $\sum_{i=1}^n w_i = 1$ with this choice of w, we have, for all $x \in X$

$$\sum_{i=1}^n w_i[f_i(x) - f_i(x^*)] \geq 0 \Longrightarrow \sum_{i=1}^n w_i f(x) \geq \sum_{i=1}^n w_i f_i(x^*)$$

Hence x^* solves the weighting problem.

Advantages: Different pareto-optimal solutions can be found by using different ε_m values. The same method can also be used for problems having convex or non-convex objective space alike.

Disadvantages: The solution of the problem largely depends on the chosen ϵ vector. Moreover, as the number of objectives increases, there exists more elements in the ϵ vector there by requiring more information from the user.

3.1.2 Weighted Sum Method

A multi-objective problem is often solved by combining its multiple objectives into one single-objective scalar function. This approach is in general known as the weighted-sum or scalarization method. In more detail, the weighted-sum method minimizes a positively weighted convex sum of the objectives, that is,

$$min \sum_{i=1}^{n} w_i f_i(x) \tag{3.2}$$

$$s.t. \quad \sum_{i=1}^{n} w_i = 1$$

$$w_i \geqslant 0, i = 1, \cdots, n$$

$$x \in S$$

that represents a new optimization problem with a unique objective function.

Theorem 3.4. The solution of weighting problem(3.2) is weakly pareto-optimal[**?**].

Proof. Let $x^* \in S$ be a solution of the weighting problem. Let us suppose that it is not weakly pareto-optimal. In this case, there exists a solution $x \in S$ such that $f_i(x) < f_i(x^*)$ for all $i = 1, 2, \cdots, n$. According to the assumptions set to the weighting coefficients, $w_j > 0$ for at least one j. Thus, we have$\sum_{i=1}^{n} w_i f_i(x) < \sum_{i=1}^{n} w_i f_i(x^*)$. This is a contradiction to the assumption that x^* is a solution of the weighting problem. Thus x^* is weakly pareto-optimal.

Theorem 3.5. The solution of weighting problem(3.2) is pareto-optimal if the weighting coefficients are positive.

Proof. Let $x^* \in S$ be a solution of the weighting problem with positive weighting coefficients. Let us suppose that it is not pareto-optimal. This means that there exists a solution $x \in S$ such that $f_i(x) \leq f_i(x^*)$ for all $i = 1, 2, \cdots, n$ and $f_j(x) < f_j(x^*)$ for at least one j. Since $w_i > 0$ for all $i = 1, \cdots, n$, we have $\sum_{i=1}^{n} w_i f_i(x) < \sum_{i=1}^{n} w_i f_i(x^*)$. This is a contradiction to the assumption that x^* is a solution of the weighting problem. Thus x^* must be pareto-optimal.

Theorem 3.6. Let the multi-objective optimization problem be convex. If $x^* \in S$ is pareto-optimal, then there exist a weighting vector w $(w_i \geq 0, i = 1, 2, \cdots n, \sum_{i=1}^{n} w_i = 1)$ such that x^* is a solution of weighting problem(3.2).

Proof. Since x^* is pareto-optimal, it is by Theorem(3.2) a solution of the ϵ-Constraints problem for every objective function f_j to be minimized. By the aid of the convexity assumption and theorem(3.3) the proof is completed.

Advantages: This is probably the simplest way to solve a multi-objective optimization problems. The concept is intuitive and easy to use. For problems having a convex pareto-optimal front, this method guarantees finding solutions on the entire pareto-optimal set.

Disadvantages: In most non-linear multi-objective optimization problems, a uniformly distributed set of weight vectors need not find a uniformly distributed set of pareto-optimal solutions. Since this mapping is not usually known, it becomes difficult to set the weight vectors to obtain a pareto-optimal solution in a desired region in the objective space.

Moreover, different weight vectors need not necessarily lead to different pareto-optimal solutions. If the chosen single objective optimization algorithm can not find all minimum solutions for a weight vector, some pareto-optimal solutions can not be found.

3.2 Principles of Genetic Algorithms Operator

3.2.1 Binary Genetic Algorithms

A binary code represents text or computer processor instructions using the binary number system's two binary digits, 0 and 1. A binary code assigns a bit string to each symbol or instruction. For example, a binary string of eight binary digits (bits) can represent any of 256 possible values and can therefore correspond to a variety of different symbols, letters or instructions[**?**].

Representing a solution In order to use GAs to find the optimal decision variables which satisfy the constraint and objective functions, we first need to represent them in binary strings. However, in fact, GAs can be assigned to use any integer or non-integer values just by changing the string length and lower and upper bounds;

$$x_i = x_i^{min} + \frac{x_i^{max} - x_i^{min}}{2^{l_i - 1}} DV(s_i)$$

Where l_i is the string length used to code the i^{th} variable and $DV(s_i)$ is the decoded value of the string s_i (where the complete string is $s = \cup_{i=1}^{n} s_i$). It allows the decision variables are to take positive and negative values.

Assigning fitness to a solution It is important to reiterate that binary GAs work with strings representing the decision variables, instead of decision variable themselves. Once a string(or a solution) is created by genetic operators, it is necessary to evaluate the solution, particularly in the context of the underlying objective and constraint functions.

The evaluation of a solution means calculating the objective function value and constraint violations. Thereafter, a metric must be defined by using the objective function value and constraint violations to assign a relative merit to the solution (called the fitness).

- **Reproduction or Selection operator** The primary objective of the reproduction operator is to make duplicates of good solutions and eliminate bad solution in a population; while keeping the population size constant. The common method of selection operator are:

- **Tournament selection** It is played between two solutions and the better solution is chosen and placed in the matting pool. Two other solutions are picked again and another slot in the matting pool is filled with the better solution. The best solution in a population will win both times, thereby making two copies of it in the new population. Using a similar argument, the worst solution will lost in both tournament and will be eliminated from the population. In this way, any solution in a population will have zero, one or two copies in the new population.
- **Proportionate selection** Solutions are Assigned copies, the number of which is proportional to their fitness values. If the average fitness of all population member is f_{avg}, a solution with a fitness f_i gets an expected $f_i \diagup f_{avg}$ number of copies.
- **Ranking selection** First, the solutions are sorted according to their fitness, from the worst(rank 1) to the best(rank N). Each member in the sorted list is assigned a fitness equal to the applied with the ranked fitness values, and N solutions are chosen for the matting pool.

- **Crossover operator** This operator randomly chooses a locus and exchanges the subsequences before and after that locus between two chromosomes to create two offspring. Like the Selection operator, there exists a number of crossover operators i.e., single-point, two-point and uniform crossover operator.

  In *single-point* crossover operator, this is performed by randomly choosing a crossing size along the string and by exchanging all bits on the right side of the crossing sizes. For example, the strings 10000100 and 11111111 could be crossed over after the third locus in each to produce the two offspring 10011111 and 11100100.

  In a *two-point* crossover operator, two different cross sizes are chosen at random. This will divide the string into three sub-strings. The crossover operation is completed by exchanging the middle substring between the strings.

- **Mutation operator** The bit-wise mutation operator changes a 1 to a 0, and vice verse, with a mutation probability of p_m. The need for mutation is to keep diversity in the population.

3.2.2 Real-parameter Genetic Algorithms

Real parameters are used directly (without any string coding), solving real-parameter optimization problems is a step easier when compared to the binary-coded GAs. Decision variables can be directly used to compute the fitness values. Since the selection operator works with the fitness value, any **selection operator** used with binary-coded GAs can also be used in real-parameter GAs[**?**].

Simulated Binary Crossover: The procedure of computing the children solutions $x_i^{(1,t+1)}$ and $x_i^{(2,t+1)}$ from parent solutions $x_i^{(1,t)}$ and $x_i^{(2,t)}$ is described as follows. A spread factor β is defined as the ratio of the absolute difference in children values to the parent

values:

$$\beta = |\frac{x_i^{(2,t+1)} - x_i^{(1,t+1)}}{x_i^{(2,t)} - x_i^{(1,t)}}|$$

First, a random number u between 0 and 1 is created. Thereafter, from a specified probability distribution function, the ordinate $\overline{\beta}$ is found so that the area under the probability curve from 0 to $\overline{\beta}$ is equal to the chosen random number u. The probability distribution used to create a child solutionis derived from an analysis of search power and is given as follows [**?**]:

$$C(\beta) = \begin{cases} 0.5(n+1)\beta^n & \text{if, } \beta \leq 1; \\ 0.5(n+1)\dfrac{1}{\beta^{n+2}} & \text{, } otherwise. \end{cases}$$

Where n is any non-negative real number.Using this equation we can calculate $\overline{\beta}$ as follows:

$$\overline{\beta} = \begin{cases} (2u)^{\dfrac{1}{n+1}} & \text{if,} u \leq 0.5; \\ (\dfrac{1}{2(1-u)})^{\dfrac{1}{n+1}} & \text{,} otherwise. \end{cases}$$

After obtaining $\overline{\beta}$ from the above probability distribution, the children solutions are calculated as follows:

$$x_i^{(1,t+1)} = 0.5[(1+\overline{\beta})x_i^{(1,t)} + (1-\overline{\beta})x_i^{(2,t)}],$$
$$x_i^{(2,t+1)} = 0.5[(1-\overline{\beta})x_i^{(1,t)} + (1+\overline{\beta})x_i^{(2,t)}].$$

Non-Uniform Mutation: Here, the probability of creating a solution closer to the parent is more than the probability of creating one away from it. However, as the generations (t) proceed, this probability of creating solutions closer to the parents gets higher and higher.

$$y_i^{(1,t+1)} = x_i^{(1,t+1)} + \tau(x_i^{(U)} - x_i^{(L)})(1 - r_i^{1-t/t_{max}})^b$$

Where τ takes a boolean value, -1 or 1, each with a probability of 0.5 and t_{max} is the maximum number of allowed generation, while b is a user defined parameter.

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 being the upper bound on the parent component, p_k^l is the lower bound and δ_k is small variation which is calculated from a polynomial distribution by using

$$\delta_k = (2r_k)^{(\eta_m+1)} - 1, if r_k < 0$$

$$\delta_k = 1 - [2(1-r_k)]^{(\eta_m+1)}, if r_k \geq 0$$

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

Chapter 4

Non-Dominated Sorting Genetic Algorithm

4.1 The common approach to fitness function

Weighted Sum Approaches:
The classical approach to solve a multi-objective optimization problem is to assign a weight $w_i i$ to each objective function $f_i(x)$ so that the problem is converted to a single objective problem with a scalar objective function. The weighted sum approach is a straightforward to implementation, Since a single objective is used in fitness assignment and a single objective GA can be used with minimum modifications[**?**].

Population based approaches: The classical example of this sort of approach is the Vector Evaluated Genetic Algorithm (VEGA), proposed by Schaffer[**?**]. VEGA basically consists of a simple genetic algorithm with a modified selection mechanism. In the VEGA, population P_t is randomly divided into K equal sized sub-populations; $P_1, P_2, \cdots, P_K$. Then, each solution in sub-population P_i is assigned a fitness value based on objective function f_i. Solutions are selected from these sub-populations using proportional selection for crossover and mutation. Crossover and mutation are performed on the new population in the same way with the single objective GA[**?**].

Pareto-Ranking Approaches: Pareto-ranking approaches explicitly utilize the concept of Pareto dominance in evaluating fitness or assigning selection probability to solutions. The population is ranked according to a dominance rule, and then each solution is assigned a fitness value based on its rank in the population, not its actual objective function value[**?**].

Consider an individual x_i at generation t which is dominated by an individuals in the current population. Its current position in the individuals' rank can be given by

$$rank(x_i; t) = 1 + nq(x_i; t)$$

where $nq(x_i; t)$ is the number of solutions dominating solution x_i at generation t. All non-dominated individuals are assigned rank 1.

4.2 Diversity

Maintaining a diverse population is an important consideration in multi-objective GA to obtain solutions uniformly distributed over the true Pareto font. Without taking any preventive measures, the population tends to form relatively few clusters in multi-objective GA. This phenomenon is called genetic drift, and several approaches are used to prevent genetic drift, as follows.

. **Fitness Sharing** Fitness sharing aims to encourage the search in unexplored sections of a Pareto front by artificially reducing fitness of solutions in densely populated areas. To achieve this goal, densely populated areas are identified and a fair penalty method is used to penalize the solutions located in such areas. Sharing function is used to obtain an estimate of the number of solutions belonging to each optimum. The idea of fitness sharing was first proposed by Goldberg and Richardson[**?**] in the investigation of multiple local optima for multi-modal functions. They used the following function in their simulation studies:

$$sh(d_{ij}) = \begin{cases} 1 - (\frac{d}{\sigma_{share}})^{\alpha} & \text{if,} d \leq \sigma_{share} \\ 0 & \text{otherwise} \end{cases}$$

The parameter d is the distance between any two solutions in the population. Although α does not have too much effect on the performance of the sharing function method. In most application, an $\alpha = 1$ or 2 is used. The Euclidean distance between two decision variable vectors $x^{(i)}$ and $x^{(j)}$ can be calculated as d_{ij}:

$$d_{ij} = \sqrt{\sum_{k=1}^{n} (x_k^{(i)} - x_k^{(j)})^2}$$

and σ_{share} for introducing q equispaced (optima equally) niches in the search space is:

$$\sigma_{share} = \frac{\sqrt{\sum_{k=1}^{n} (x_k^{(U)} - x_k^{(L)})^2}}{2(\sqrt{q})^{1/n}}$$

If d is zero(meaning that two solutions are identical or their distance is zero), $sh(d) = 1$. This means that a solution has full sharing effect on it itself. On the other hand, if $d \geq \sigma_{share}$ meaning that two solutions are at least a distance of σ_{share} away from each other), $sh(d) = 0$. This means that two solutions which are a distance of σ_{share} away from each other do not have any sharing effect on each other.

The niche count nc_i is calculated for the i_{th} solution, as follows:

$$nc_i = \sum_{j=1}^{n} sh(d_{ij})$$

Then the niche count provides an estimate of the extent of crowding near a solution. It is important to note that nc_i is always greater than or equal to one. This is because the right side includes the term $sh(d_{ii}) = sh(0) = 1$. The final task is to calculate the shared fitness value as

$$f_i^{'} = \frac{f_i}{nc_i}$$

Two solutions might be very close in the objective function space while they have very different structural features. Therefore, fitness sharing based on the objective function space may reduce diversity in the decision variable space. However, Deb and Goldberg [**?**] reported that fitness sharing on the objective function space usually performs better than one based on the decision variable space.

Diversity Preservation

Most multi-objective evolutionary algorithms (MOEAs) try to maintain diversity within the current Pareto set approximation by incorporating density information into the selection process: an individuals chance of being selected is decreased the greater the density of individuals in its neighborhood.

The sharing function method involves a sharing parameter σ_{share} , which sets the extent of sharing desired in a problem. This parameter is related to the distance metric chosen to calculate the proximity measure between two population members. The parameter σ_{share} denotes the largest value of that distance metric within which any two solutions share each others fitness. This parameter is usually set by the user, although there exist some guidelines[**?**]. In the proposed NSGA-II, we replace the sharing function approach with a crowded-comparison approach that eliminate any user-defined parameter for maintaining diversity among population members.

Density Estimation: To get an estimate of the density of solutions surrounding a particular solution in the population, we calculate the average distance of two points on either side of this point along each of the objectives. This quantity $i_{distance}$ serves as an estimate of the perimeter of the cuboid formed by using the nearest neighbors as the vertices (call this the crowding distance).

Crowded-Comparison Operator: The crowded-comparison operator ($\prec_n$) guides the selection process at the various stages of the algorithm toward a uniformly spread-out Pareto-optimal front. Assume that every individual x_i in the population has two attributes:

1) non-domination rank (i_{rank});

2) crowding distance ($i_{distance}$).

We now define a partial order $\prec_n$ as:

$$i \prec_n j \text{ ,if } (i_{rank} < j_{rank}) \text{ or } (i_{rank} = j_{rank}) \text{ and } (i_{distance} > j_{distance})$$

That is, between two solutions with differing non-domination ranks, we prefer the solution with the lower (better) rank. Otherwise, if both solutions belong to the same front, then we prefer the solution that is located in a lesser crowded region.

Elitisim

Elitism in the context of single-objective GA means that the best solution found so far during the search has immunity against selection and always survives in the next generation. In this respect, all non-dominated solutions discovered by a multi-objective GA are considered as elite solutions.

Elitism can be introduced globally in a generational sense. Once the offspring population is created, both parent and offspring population can be combined together. Thereafter, the best N, members may be chosen to form the population of the next generation without any parameter. In this way too, parents get a chance to compute with the offspring population for their survival in the next generation. It makes sure that the fitness of the population of best solution does not deteriorate. In this way, a good solution found early on in the run will never be lost unless a better solution is discovered.

In fact, Rudolph(1996) has proved that GAs converge to the global optimal solution of some functions in the presence of elitism. Moreover, the presence of elites enhances the probability of creating better offspring[**?**]. Elitism can be implemented to different degrees. For example, one can simply keep track of the best solution in a population and update it if a better solution is discovered at any generation, but not use the elite solutions in any genetic operations. On the other hand, in another extreme implementation, all elites present in the current population can be carried over to the new population. In this way, not many new solutions get a chance to enter the new population and the search does not progress any where.

Termination of the Genetic Algorithm

Because the GA is a stochastic search method, it is difficult to formally specify convergence criteria. As the fitness of a population may remain static for a number of generations before a superior individual is found, the application of conventional termination criteria becomes problematic. A common practice is to terminate the GA after a prespecified number of generations and then test the quality of the best members of the population against the problem definition. The number of generations that evolve depends on whether an acceptable solution is reached or a set number of iterations is exceeded. After a while all the chromosomes and associated costs would become the same if it were not for mutations. At this point the algorithm should be stopped. If no acceptable solutions are found, the GA may be restarted or a fresh search initiated.

4.3 Elitist Non-Dominated Sorting Genetic Algorithm

In NSGA-II, the offspring population Q_t is first created by using the parent population P_t. However, instead of finding the non-dominated front of Q_t only, first the two populations are combined together to form R_t of size $2N$. Then, a non-dominated sorting is used to classify

the entire population R_t. Although this requires more effort compared to performing a non-dominated sorting on Q_t alone, it allows a global non-domination check among the offspring and parent solutions.

Once the non-dominated sorting is over, the new population is filled by solutions of different non-dominated fronts, one at a time. The filling starts with the best non-dominated front and continues with solutions of the second non-dominated front, followed by the third non-dominated front, and so on.

In the following, we outline the algorithm in a step-by-step format. Initially, a random population P_o is created. The population is sorted into different non-domination levels. Each solution is assigned a fitness equal to its non-domination level (1 is the best level). Thus, minimization of the fitness is assumed. Binary tournament selection (with a crowded tournament operator described later), recombination and mutation operators are used to create an offspring population Q_0 of size N. The NSGA-II procedure is outlined in the following.

step 1 Combine parent and offspring populations and create $R_t = P_t \cup Q_t$. Perform a non-dominated sorting to R_t and identify different fronts:$\mathfrak{f}_i$, $i = 1, 2, \cdots$, etc.

step 2 Set new population $P_{t+1} = \emptyset$. Set a counter $i = 1$. Until $\mid P_{t+1} \mid + \mid \mathfrak{f}_i \mid < N$, perform $P_{t+1} = P_{t+1} \cup \mathfrak{f}_i$ and $i = i + 1$.

step 3 Perform the crowding-sort ($\mathfrak{f}_i <_c$) procedure and include the most widely spread ($N - \mid P_{t+1} \mid$) solutions by using the crowding distance values in the sorted $\mathfrak{f}_i$ to P_{t+1}.

step 4 Create offspring population Q_{t+1} from P_{t+1} by using the crowded tournament selection, simulated binary crossover and polynomial mutation operators.

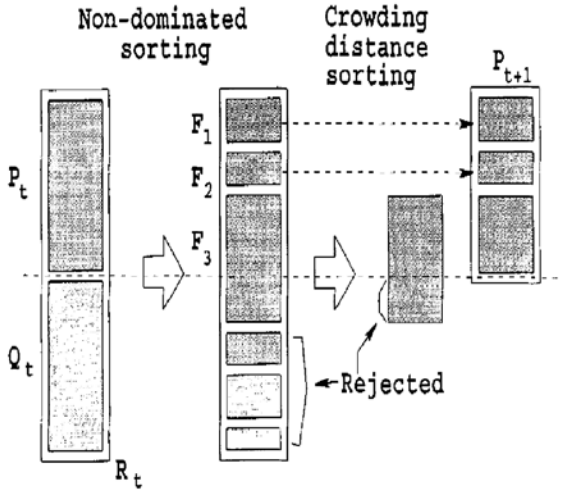


Figure 4.1: NSGA-II procedure
Adopted from:- [**?**]

Crowding Distance

To get an estimate of the density of solutions surrounding a particular solution x_i in the population, we take the average distance of two solutions on either side of solution x_i along each of the objectives. This quantity d_i serves as an estimate of the parameter of the cuboid formed by using the nearest neighbors as the vertices (we call this the crowding distance). The following algorithm is used to calculate the crowding distance of each point in the set $\mathfrak{f}$

Crowding Distance Assignment Procedure:crowding-sort ($\mathfrak{f} <_c$)

step 1 Call the number of solutions in $\mathfrak{f}$ as $l =\mid \mathfrak{f} \mid$. For each i in the set, first assign $d_i = 0$.

step 2 For each objective function $m = 1, 2, \cdots, M$, sort the set in worse order of f_m or, find the sorted indices vector: $I^m = sort(f_m, >)$.

step 3 For $m = 1, 2, \cdots, M$, assign a large distance to the boundary solutions, or $d_{I_h^m} = d_{I_l^m} = \infty$, and for all other solutions $j = 2$ to $(l-1)$, assign:

$$d_{I_j^m} = d_{I_j^m} + \frac{f_m^{I_{j+1}^m} - f_m^{I_{j-1}^m}}{f_m^{max} - f_m^{min}}$$

The index I_j denotes the solution index of the j^{th} member in the sorted list. Thus, for any objective, I_l and I_h denote the lowest and highest objective function values respectively. The second term on the right side of the last equation is the difference in objective function values between two neighboring solutions on either side of solution I_j.

Crowded Tournament Selection Operator

The crowded comparison operator ($<_c$) compares two solutions and returns the winner of the tournament. It assumes that every solution x_i has two attributes.

1, A non-domination rank r_i in the population

2, A local crowding distance (d_i) in the population.

The crowding distance d_i of a solution x_i is a measure of the search space around x_i which is not occupied by any other solution in the population. Based on these two attributes, we can define the crowded tournament selection operator as follows.

Definition 4.3.1. *Crowded tournament selection operator: A solution x_i wins a tournament with another solution $_j$ if any of the following condition are true:*

1, If solution x_i has a better rank, that is, $r_i < r_j$.

2, If they have the same rank but solution x_i has a better crowding distance than solution x_j, that is, $r_i = r_j$ and $d_i > d_j$.

The first condition makes sure that chosen solution lies on a better non-dominated front. The second condition resolves the tie of both solutions being on the same non-dominated front by deciding on their crowded distance. The one residing in a less crowded area (with a large crowding distance d_i) wins.

Example 4.1. We consider the following two-objective,two-variable minimization problem to illustrate how the algorithm, presented in this project, works.

$$\begin{aligned} &\text{minimize } f_1(x) = x_1 \\ &\text{minimize } f_2(x) = \frac{1+x_2}{x_1} \end{aligned}$$

$$\begin{aligned} \text{subject to } &0.1 \leq x_1 \leq 1, \\ &0 \leq x_2 \leq 5. \end{aligned}$$

We have also chosen six random solutions and we assume an offspring population of six solutions, in the search space for illustrating the working principle of algorithm described in this project. These solutions are also tabulated in the following table.
Table 4.1: Parent and offspring with their objective function value.

Parent population					*Offspring population*				
solution	x_1	x_2	f_1	f_2	*solution*	x_1	x_2	f_1	f_2
1	*0.31*	*0.89*	*0.31*	*6.10*	*a*	*0.21*	*0.24*	*0.21*	*5.90*
2	*0.43*	*1.92*	*0.43*	*6.79*	*b*	*0.79*	*2.14*	*0.79*	*3.97*
3	*0.22*	*0.56*	*0.22*	*7.09*	*c*	*0.51*	*2.32*	*0.51*	*6.52*
4	*0.59*	*3.63*	*0.59*	*7.85*	*d*	*0.27*	*0.87*	*0.27*	*6.93*
5	*0.66*	*1.41*	*0.66*	*3.65*	*e*	*0.58*	*1.62*	*0.58*	*4.52*
6	*0.83*	*2.51*	*0.83*	*4.23*	*f*	*0.24*	*1.05*	*0.24*	*8.54*

step 1 We first combine the population P_t and Q_t and from a set $R_t = \{1, 2, 3, 4, 5, 6, a, b, c, d, e, f\}$. Next, we perform a non-dominated sorting on R_t. We obtain the following non-dominated fronts:

$$\begin{aligned} \mathfrak{f}_1 &= \{5, a, e\}, \\ \mathfrak{f}_2 &= \{1, 3, b, d\}, \\ \mathfrak{f}_3 &= \{2, 6, c, f\}, \\ \mathfrak{f}_4 &= \{4\}. \end{aligned}$$

step 2 We set $P_{t+1} = \emptyset$ and $i = 1$. Next, we observe that $\mid P_{t+1} \mid + \mid \mathfrak{f}_1 \mid = 0 + 3 = 3$. Since this is less than the population size $N(= 6)$, we include this front in P_{t+1}. We set $P_{t+1} = \{5, a, e\}$. With these three solutions, we now need three more solutions to fill up the new parent population. Now, with the inclusion of the second front, the size of $\mid P_{t+1} \mid + \mid \mathfrak{f}_2 \mid$ is $(3+4)$, or 7. Since this is greater than 6, we stop including any more fronts into the population. Note that if fronts 3 and 4 had not been classified earlier, we could have saved these computations.

step 3 Next, we consider solutions of the second front only and observe that three (of four) solutions must be chosen to fill up three remaining slots this subpopulation (solution 1,3,b and d) by using the $<_c$ operator. We calculate the crowded distance values of these solutions in the front by using the step-by step procedure.

step C_1 We notice that $l = 4$ and set $d_1 = d_3 = d_b = d_d = 0$. We also set $f_1^{max} = 1$, $f_1^{min} = 0.1$, $f_2^{max} = 60$ and $f_2^{min} = 0$.

step C_2 For the first objective function, the sorting of these solutions is shown in table 4.2 and is as follows:

$$I^1 = \{3, d, 1, b\}.$$

step C_3 Since solutions 3 and b are boundary solution, we set $d_3 = d_b = \infty$. for the other two solutions, we obtain:

$$d_d = 0 + \frac{f_1^{(1)} - f_1^{(3)}}{f_1^{max} - f_1^{min}} = 0 + \frac{0.31 - 0.22}{1 - 0.1} = 0.10.$$

$$d_1 = 0 + \frac{f_1^{(b)} - f_1^{(d)}}{f_1^{max} - f_1^{min}} = 0 + \frac{0.79 - 0.27}{1 - 0.1} = 0.58.$$

Now, we turn to the second objective function and update the above distances. First the sorting on this objective yields $I^2 = \{b, 1, d, 3\}$. Thus, $d_3 = d_b = \infty$ and the other two distance are as follows:

$$d_d = d_d + \frac{f_2^{(3)} - f_2^{(1)}}{f_2^{max} - f_2^{min}} = 0.10 + \frac{7.09 - 6.10}{60 - 0} = 0.12.$$

$$d_1 = d_1 + \frac{f_2^{(d)} - f_2^{(b)}}{f_2^{max} - f_2^{min}} = 0.58 + \frac{6.93 - 3.97}{60 - 0} = 0.63.$$

The overall crowding distances of the four solutions are:

$$d_1 = 0.63, d_3 = \infty, d_b = \infty, d_d = 0.12.$$

Evidently solution d has the smallest parimeter of the hypercube around it than any other solution in the set $\mathfrak{f}_2$. Now, we move to the main algorithm.

Table 4.2: The fitness assignment procedure under NSGA-II .

Front 1					*Sorting in*		*Distance*
solution	x_1	x_2	f_1	f_2	f_1	f_2	
5	*0.66*	*1.41*	*0.66*	*3.65*	*third*	*first*	∞
a	*0.21*	*0.24*	*0.21*	*5.90*	*first*	*third*	∞
e	*0.58*	*1.62*	*0.58*	*4.52*	*second*	*second*	*0.54*
Front 2					*Sorting in*		*Distance*
solution	x_1	x_2	f_1	f_2	f_1	f_2	
1	*0.31*	*0.89*	*0.31*	*6.10*	*third*	*second*	*0.63*
3	*0.22*	*0.56*	*0.22*	*7.09*	*first*	*fourth*	∞
b	*0.79*	*2.14*	*0.79*	*3.97*	*fourth*	*first*	∞
d	*0.27*	*0.87*	*0.27*	*6.93*	*second*	*third*	*0.12*

step 3 Sorting,in descending order, these crowding distance values, yields the sorted set $\{3, b, 1, d\}$. We choose the first three solutions.

step 4 The new population is $P_{t+1} = \{5, a, e, 3, b, 1\}$. It is important to note that this population is formed by choosing solutions from the better non-dominated fronts.

The offspring population Q_{t+1} has to be created next by using this parent population. We realize that the exact offspring population will depend on the chosen pair of solutions participating in a tournament and the chosen crossover and mutation operators. Let us say that we pair solutions (5,e), (a,3), (1,b), (a,1), (e,b) and (3,5), so that each solution participates in exactly two tournament. In the first tournament, we observe that solutions 5 and e belong to the same front ($r_5 = r_e = 1$). Thus, we choose the one with larger crowding distance value. We find that solution 5 is the winner.

In the next comparison between solutions a and 3, solution a wins, since it belongs to a better front. Performing other tournaments, we obtain the mating pool:$\{5, 5, a, a, b, e\}$. Now, these solutions can be mated pair-wise and mutated to create Q_{t+1}. This completes one generation of the NSGA-II.

Example 4.2. *Consider the following two-objective multi-objective optimization problem:*

$$min\ f_1(x) = 3x^3 - 26x + 10$$

$$f_2(x) = 9x^2 - 26$$

subject to the constraints:

$$x \geq -2.5$$

We applied the non-dominated sorting genetic algorithm (NSGA-II) to this problem and the algorithm was coded in Matlab. We use the Simulated Binary Crossover (SBX) and the polynomial mutation operator and also we used the crossover probability of $p_c = 0.9$ and a mutation probability of $p_m = \frac{1}{n}$ (where n is the number of decision variables). The distribution indices for crossover and mutation operators as $\mu_c = 20$ and $\mu_m = 20$ respectively. It shows a typical result with a population of 50 individuals and figure 4.2 and figure 4.3 are the classical representation in multi-objective optimization: f_2 vs. f_1 after 5 and 100 generations respectively.

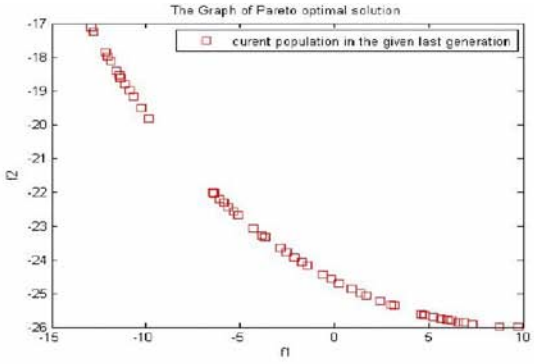


Figure 4.2:*The population after 5 generations with NSGA-II*

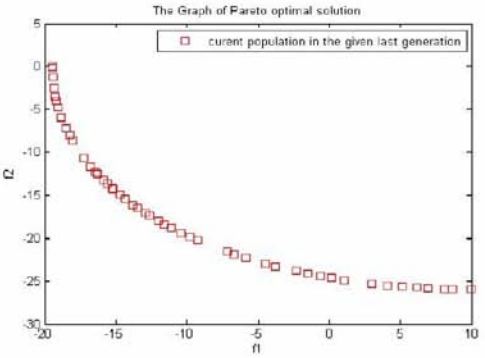


Figure 4.3:*The population after 100 generations with NSGA-II*

Example 4.3. Function $f_1(X)$ and $f_2(X)$ proposed by Zitzler [**?**] problem consists of solving the following multi-objective optimization problem:

$$\begin{aligned} &minimize\ f_1(X) = x_1 \\ &minimize\ f_2(X) = g(X).(1 - \sqrt{\frac{f_1}{g(X)}}) \end{aligned}$$

$$\begin{aligned} s.t. \qquad g(X) &= 1 + \frac{9}{n-1}\sum_{i=0}^{n} x_i \\ 0 &\leq x_i \leq 1 \end{aligned}$$

$$where\ X = (x_1, \cdots, x_n)\ and\ n = 30.$$

We applied the non-dominated sorting genetic algorithm (NSGA-II) to this problem and the algorithm was coded in Matlab. We use the Simulated Binary Crossover (SBX) and the polynomial mutation operator and also we used the crossover probability of $p_c = 0.9$ and a mutation probability of $p_m = \frac{1}{n}$ (where n is the number of decision variables). The distribution indices for crossover and mutation operators as $\mu_c = 20$ and $\mu_m = 20$ respectively. It shows a typical result with a population of 100 individuals and figure 4.4 and figure 4.5 are the classical representation in multi-objective optimization: f_2 vs. f_1 after 50 and 500 generations respectively.

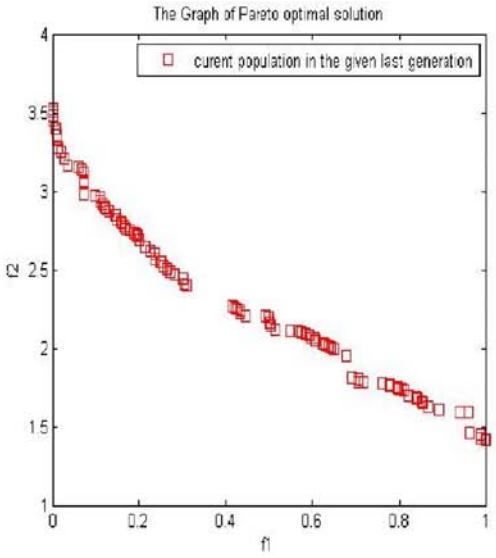


Figure 4.4: *The population after 50 generations with NSGA-II*

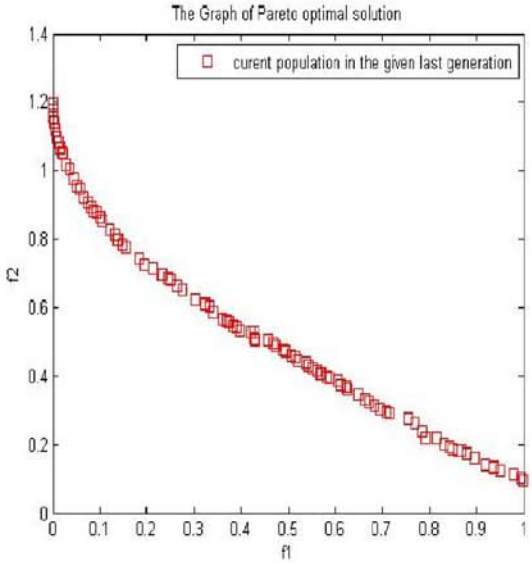


Figure 4.5: *The population after 500 generations with NSGA-II*

Bibliography

[1] Kalyanmoy-Deb, *Multiobjective Optimization using Evolutionary Algorithms*, John Wiley & Sons, Ltd, England, (2001).

[2] M.Ehrgott, *Multi-criteria Optimization*, Springer, Berlin. Heidelderg, (2nd edition), (2005).

[3] K. Miettinen, *Nonlinear Multiobjective Optimization*, Kluwer Academic Publishers, Boston(1999).

[4] A.Abraham, L. Jain and R. Goldberg(Eds). *Evolutionary Multiobjective Optimization*, Springer-Verlag London, (2005).

[5] Y. Sawaragi, H. Nakayama and T. Tanino, *Theory of Multiobjective Optimization*, Academic press INC, London, (1985).

[6] V. Chankong, Y. Y. Haimes, *Multiobjective Decision Making Theory and Methodology*, Elsevier Science Publishing Co. Inc, New York, (1983).

[7] T. Back*, Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York, (1996).

[8] C. A. C. Coello, G. B. Lamont and D. A. V. Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems* Springer, New York, (2nd edition), (2002).

[9] O. L. Mangasarian, *Nonlinear Programming*, McGraw-Hill, New York, (1969).

[10] L. Wang, A. H. C. Ng and Kalyanmoy Deb(Eds), *Multi-objective Evolutionary Optimization for Product Design and Manufacturing*, Springer, Verlag London, (2011).

[11] S.S. Rao, *Optimization theory and application*, Wiley Eastern Limited, New Delhi, (1991).

[12] C. M. Fonseca and P. J. Fleming. *Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization.* In S. Forrest, editor, Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, California, pp. 416-423, (1993).

[13] Abdullah Konak, David W. Coit, Alice E. Smith, *Multi-objective optimization using genetic algorithms:* A tutorial, Elsevier Ltd, (2005)

[14] Zitzler, E. and Thiele, L., Multiobjective evolutionary algorithms: a comparative case study and the strength of pareto approach, IEEE Transactions on Evolutionary Computation 3(4), 257-271 (1999).

[15] Schaffer, J.D. *Multiple Objective optimization with vector evaluated genetic algorithms*, In J. J. Grefenstette, editor, Proceedings of an International Conference on Genetic Algorithm and their applications, pp. 93-100, Pittsburgh,(1985).

[16] Deb, K. and Goldberg, D.E. *An investigation of of niche an species fromation in genetic function optimization*' in The Third International Conference on Genetic Algorithms. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, pp. 42-50, (1989).

[17] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In M. Schoenauer et al., editors, Parallel Problem Solving from Nature (PPSN VI), Springer, pp. 849 858, Berlin, (2000).

[18] K. Deb and R. B. Agrawal, *Simulated binary crossover for continuous search space*,Complex System 9(2), pp. 115-148, (1995).

[19] K. Deb and M. Goyal, *A combined genetic adaptive search for engineering design*, computer science and informatics 26(4), pp. 30-45,(1996).