Chapter 2. Literature Review

2.1 Function optimization

Function optimization is the process of finding a set of possible points from feasible region of the search space so that the value of function becomes either maximum or minimum depending on the problem and requirements. Global optimization is the task of finding the absolutely best set of admissible conditions to achieve an objective under given constraints, assuming that both are formulated in mathematical terms. Global optimization problems are a class of NP-complete problems so there is not a single algorithm that solves global optimization problems in polynomial time [3].

Basically function optimization problems are made up of following three parts,

1. An objective function. We want to minimize or maximize value of the function depending upon problem. For example, in the financial application we want to maximize profit and minimize risk. In water reservoir system, we want to maximize storage of water, generation of power and irrigation at the same time by minimizing risk, evaporation losses, etc...

2. A set of variables. These variables affect the value of the objective function. In industry, the variables might include the amounts of different resources used or the time spent on each activity. In the water reservoir system, the variables are amount of rainfall, capacity of reservoir, demand for the irrigation, etc...

3. A set of constraints. The variables can take certain values and they cannot take other values depending on the constraints. In the industry, we cannot have unlimited resources or money, time spent on each activity cannot be negative, etc... In the water reservoir system, losses cannot be nil, evaporation losses must be in proportional to the area hilted by sunlight, etc...
The optimization problem is to find values of the variables that minimize or maximize the objective function value while satisfying the constraints. Mathematical formulation is

\[
\begin{align*}
\max \text{ or } \min & \quad f(x) \\
\text{subject to} & \quad x \in D, \quad \text{where} \quad D = \{x : l \leq x \leq u\} \\
\text{subject to} & \quad g_j(x) \leq 0, \quad \text{where} \quad j = 1, \ldots, J.
\end{align*}
\]

- \(x \in \mathbb{R}^n\): real \(n\) - vector of decision,
- \(f : \mathbb{R}^n \rightarrow \mathbb{R}\): continuous objective function,
- \(D \subset \mathbb{R}^n\): non-empty set of feasible decisions (a proper subset of \(\mathbb{R}^n\)),
- \(l\) and \(u\): explicit finite lower bound and upper bound on \(x\),
- \(g : \mathbb{R}^n \rightarrow \mathbb{R}^m\): finite collection of continuous constraint functions (\(J\) - vector)

The above shown model is called bounded, constraint optimization model. If the 1\textsuperscript{st} condition is relaxed then it becomes unbounded means decision variables can take any value. Relaxation of 2\textsuperscript{nd} condition is known as unconstrained optimization [3][4].

2.1.1 Types of optimization

Optimization problems can be categorized in several categories depending on the characteristics of problem. Two general categories are continuous optimization and discrete optimization.
2.1.1.1 Continuous optimization

In continuous optimization all the variables are allowed to take values from subintervals of the real line and distance between search points is equal. Global optimization algorithms try to find value of decision variables that minimizes objective function, over all possible decision vectors. Continuous optimization problems are further decomposed in to two categories, one is unconstrained optimization and other is constrained optimization. In the unconstrained optimization problem, 2nd condition of mathematical model is not present so there can be any relation between decision variables. Nonlinear least squares, nonlinear equations, global optimization, non differentiable optimization are unconstrained optimization. In constrained optimization, constraint condition must be satisfied, for some decision vector, if it is not satisfied then we cannot select this vector. Nonlinearly constrained optimization, bound constrained optimization, quadratic programming, linear programming, network programming, global optimization, nondifferentiable optimization are constrained optimization [4].

2.1.1.2 Discrete optimization

In discrete optimization, variables take some fixed points from the real line and they may not be regularly spaced and there can be any distance between search points. Discrete problems are hard to solve then continuous problems. Generally two techniques are used to solve discrete optimization problems. First one is the integer programming which is used in many applications where the solution of an optimization problem makes sense only if certain of unknowns are integer only. Second is stochastic programming which can be applied to continuous optimization problems also. In many actual problems, the problem data cannot be known accurately for a variety of reasons. The first reason is due to simple measurement error, the second and more fundamental reason is that some data represent information about the future and simply cannot be known with certainty. In this type of problems, stochastic programming plays important role [4].
2.1.2 Global optimization methods

There is not a single fixed method, which solves any global optimization problem efficiently and effectively. Because global optimization problems are complex problems and they can have many different characteristics like convex or nonconvex, continuous or discrete, deterministic or stochastic, etc... So a very general optimization technique works for the broad category of problems but its efficiency might be lower for the more special problem instances. On the other hand, highly specialized method can do very good job for one particular class of problems but it may even fail to find a solution for other category of problems [3][4]. Generally global optimization techniques are applied depending on the characteristics of problem and based on different constraints. Mainly global optimization methods can be classified in two categories first is exact methods and second is heuristic methods.

2.1.2.1 Exact methods

Exact methods have a rigorous guarantee for finding global optima. However, the associated computational time may be too high, in high dimensional models it can be in thousands of years so they are not practical. Most-continuous global optimization and combinatorial models are NP-hard; therefore any amount of resources will not resolve their tractability. In this type of situation, exact stochastic algorithms are more useful. Some of the exact methods are discussed below [3][4].

2.1.2.2 Adaptive stochastic search methods

These procedures are based on the random sampling in D. D is non-empty set of feasible decisions. Adaptive search strategy adjustments, sample clustering, deterministic solution refinement options, statistical, stopping rules can be added as enhancements to the basic scheme of pure random sampling. This method can be applied to both discrete and continuous optimization.
2.1.2.3 Branch and bound algorithm

Adaptive partition, sampling, and bounding procedures can be applied to continuous global optimization models, analogously to the well-known pure integer programming, or mixed integer linear programming methods. This general approach subsumes many specific cases and allow for significant generalizations. Branch-and-bound methods are applicable to diverse broadly structured global optimization problems.

2.1.2.4 Enumerative strategies

These methods are based upon a complete enumeration of all possible solutions so brute-force approach.

2.1.2.5 Relaxation strategies

In this general approach, the global optimization problem is replaced by a sequence of relaxed sub-problems that are easier to solve. Successive refinement of sub-problems to approximate the initial problem is applied: cutting plans, more general cuts, and other customizations are possible. Relaxation algorithms are applicable to diverse structured global optimization such as concave minimization or DC programming models.

2.1.2.6 Heuristic methods

Heuristic methods are an intelligence search that tries to find out best possible solution but they do not have strict convergence guarantees; however, in many cases, they have a proven track record and they may offer practical tools to handle models that are out of theoretically correct, rigorous methodology. They exploit the tolerance for imprecision, uncertainty, partial truth, and approximation to achieve tractability, robustness and low solution cost [3][4].
• **Continuation methods**

These approaches first transform the objective function into some smoother, simpler function with fewer local optima and then use a local minimization procedure to find out all optima back to the original function.

• **Evolutionary computation**

Evolutionary optimization approaches heuristically mimic biological evolution models. Various deterministic and stochastic algorithms can be constructed based on diverse evolutionary game rules. These strategies are applicable to both discrete and continuous global optimization problems under mild structural requirements [4].

• **Simulation annealing**

These techniques are based upon the physical analogy of cooling crystal structures that spontaneously arrive at a stable configuration, characterized by globally or locally minimal potential energy. Simulated annealing is applicable to both discrete and continuous global optimization problems [5].

• **Tabu search**

The essential idea of this popular meta-heuristic is to forbid search moves to points already visited in the search space, at least within the next few steps. Tabu search methodology has been primarily used to solve combinatorial optimization problems, but it can also be extended to handle continuous global optimization problems [4].

### 2.2 Single-Objective Optimization and Optimization Algorithms

Many real world problems can be devised as optimization problems having an overall objective under limited resources forming the constraints for the given problem. In a multi-modal situation, the problem of finding the global optimal solution is often difficult by using the traditional methods. Optimization as one of the key topics in
engineering, several metaheuristic approaches has been developed for tackling such problems.

Section 2.2.1 defines the single-objective optimization problem and Section 2.2.2 presents several metaheuristic approaches for solving single-objective optimization problems.

### 2.2.1 Optimization Problems

This section summarizes the common concepts related to optimization in which the basic idea about optimization is first discussed in Section 2.2.1.1 and the principle of single-objective is then outlined in Section 2.2.1.2.

#### 2.2.1.1 Introduction and Overview

Optimization being one of the key topics in engineering is often referred to as the process of finding the best solution to a given problem. Indeed, optimization problems are very common where many problems in the real world can be formulated as an optimization problem having global optima that may be buried in local optima. Some examples of these problems include:

- In financial investment analysis, the finding of the best portfolio in terms of the return.
- In distribution planning, the finding of the best route to travel in between two locations with an aim to minimize the transit time.

- In production scheduling, the minimization of idle time in planning the operations in a factory.

For these problems, all candidate solutions can be completely ranked and ordered with respect to their corresponding objectives. The goal of such an optimization process is
therefore well-defined that aims to find the highest ranked solution within the whole ordered set of candidate solutions [4][23].

2.2.1.2 Single-Objective Optimization

A single-objective optimization problem is defined as the search for a unique optimal solution of a function. Optimization thus requires the finding of the vector of decision variables that optimizes the corresponding objective functions \( f(x) \):

\[
\text{for a maximization case: } f(\alpha) \geq f(x) \tag{2.2}
\]

\[
\text{for a minimization case: } f(\alpha) \leq f(x)
\]

where \( \bar{x} = [x_1, x_2, x_3, ..., x_n] \in \Omega \) is the n-dimension vector of variables representing the decision variables of the function \( f \) and \( \alpha = [\alpha_1, \alpha_2, ..., \alpha_n] \in \Omega \) is the vector of decision variables such that the objective function \( f \) is optimized. \( \Omega \in [l, u] \) is the feasible region of the decision variables with \( l \) being the lower bound and \( u \) being the upper bound.

The primary aim of global optimization is to find out the vector of decision variables (\( \alpha \)) such that the function is optimized [6].

2.2.2 Single-Objective Optimization Algorithms

Based on the definition of an optimization problem given in Section 2.2.1, this section presents a review and qualitative analysis of algorithms proposed in the literature that aim to tackle this kind of problems. Classical techniques and General heuristics approaches are introduced and discussed in section 2.2.2.1 and Population-based stochastic search methods are then presented in Section 2.2.2.2.
2.2.2.1 Classical Techniques and General Heuristics Approaches

There are many classical optimization techniques including the linear programming, the nonlinear programming, and the dynamic programming for solving general optimization problems. Heuristics technique is another favored approach by the research community for solving optimization problems efficiently and effectively.

A heuristics is a rule of thumb that used to produces a good solution for a complex problem. The rules of the heuristics usually ignores whether the solution is truly the best possible solution. They are typically applied when there is no exact procedure for finding an optimal solution, or when the exact procedure is not desirable in terms of available computation resources.

A heuristics approach usually involves using some information about the problem being tackled to formulate specific rules for solving the problem. Such requirement restrains the usage of general heuristics approach as it requires certain preliminary knowledge about the target problem. Metaheuristics are thus become famous as they are generalization of heuristics approaches which can simply be applied for general-purpose problem solving in finding good solutions. A good metaheuristics is one that can direct the search in the right direction. This can dramatically reduce the amount of redundancy search in the search process for a good solution.

Local search, or namely the neighborhood search, is one of the most common and simplest strategies for metaheuristics. A Local search strategy start with a randomly generated initial solution and the process is then continued to generate neighborhood solutions. Once a better solution is found, the solution replaces the original solution and the process repeats. Such a strategy is very similar to a hill climbing process and it will generally converge to a local optimum solution. Several variants of local search technique have been proposed including the Tabu search and the gradient approach [8][23].
2.2.2.2 Population-based stochastic search methods

Classical heuristic approaches are generally simplistic in idea. The rule-based concept adopted by most of the heuristic approaches makes them fail to tackle some of the problems. Heuristic approaches are in general not targeted for all type of problem, but focused on several problems with certain characteristics. In contrast, population-based stochastic search methods have been developed for all type of problems that require minor adjustment in the algorithm domain.

Over the past decade, a number of population-based stochastic search methods have been proposed in the literature for optimization problems. In particular, genetic algorithm (GA), particle swarm optimization algorithm (PSO) and ant colony optimization algorithm (ACO) are some of the well known algorithms for solving general optimization problems. The considerable volumes of related research provide clear evidence of the interest in this subject matter [8][23].

Genetics is one of the biological processes that have been widely researched and analyzed in the research community. Genetic algorithm which mimics the process of gene evolution is always a hot topic in the development of computation algorithms.

2.3 Multi-Objective Optimization and Optimization Algorithms

Section 2.2 focuses on the discussion of single-objective optimization and corresponding algorithms. This section discusses multi-objective optimization. A multi-objective optimization problem is quite different from a single-objective problem in that the definition of optimum has to be re-defined. Algorithms for solving the multi-objective optimization problems produce a set of trade-off solutions for decision makers to select.

This section provides an in-depth analysis of the multi-objective optimization problems. Section 2.3.1 defines the multi-objective optimization problem and discusses the differences between single and multiple objective optimizations. Representative
optimization algorithms have been proposed and a detailed review of these approaches will be presented in Section 2.3.2

2.3.1 Optimization Problems

This section presents the general concepts related to optimization in which the basic idea about optimization is first discussed in Section 2.3.1.1. The principle of multi-objective optimization is then outlined in Section 2.3.1.2. This is then followed by the discussion about the differences between single-objective and multi-objective optimization. Fundamental concepts regarding multi-objective optimization are given in Section 2.3.1.3.

2.3.1.1 Introduction and Overview

Real world problems usually involve simultaneously optimizing more than one single consideration (objective). In science, engineering, finance, operation research, and many other domains, the norm of the problems is to have multiple aims. Some examples of these problems include:

- In financial investment analysis, the finding of the best portfolio that balances risk and return.

- In operation research, the finding of the best route to travel in between two locations with the aim to shorten the transit time and minimize the related cost.

- In scheduling, the minimization of idle time and maximization of output in planning the operations in a factory.

These optimization problems required the simultaneous optimization of more than one objective. Furthermore, these objectives are often inter-conflicting implying that it is impossible to find one single optimum solution that satisfies all objectives at the same
time. A solution may be optimal to one objective, but at the same time be sub-optimal to another objective.

As a matter of fact, there is not even a universally accepted definition of optimum as in the case of single-objective optimization. Reasonable solutions to a multi-objective problem are indeed a set of solutions in the feasible solution space, which represents the trade-off solution set for the contradicting objectives. A multi-objective optimization problem is quite different from a single-objective one where the definition of optimum has to be re-stated. The optimal solution in a single-objective optimization has to be extended to a broader sense. Solutions which are not worst than other solutions with all objectives considered are considered at the same time are said to be acceptable optimal trade-off solutions.

In the following sections, the concepts of single-objective and multi-objective optimization are described, which is followed by the discussion of the differences between these two types of optimization problems [11].

2.3.1.2 Multi-Objective Optimization

The preceding definition is related to a problem in which there is only one objective function. For a multi-objective optimization problem, more than one objective needs to be dealt with at the same time. A multi-objective optimization problem thus requires the finding of the vector of decision variables that optimizes a number of objective functions:

Maximize / Minimize \( \bar{f}(x) \) \hspace{1cm} (2.3)

\[
\begin{align*}
\bar{f}(x) &= f_1(x) \\
&= f_2(x) \\
&= \vdots \\
&= f_n(x)
\end{align*}
\]

where \( \bar{x} = [x_1, x_2, \ldots, x_n] \in \Omega \) is the n-dimension vector of decision
variables representing the variables of objective functions $f$.

$$\Omega \in [l_i, u_i]$$ is the feasible region of the decision variables with $l$ being a lower bound and $u$ being the upper bound.

In a multi-objective optimization problem, it is almost impossible to find one single solution that is optimum for all objectives considered as these objectives are often found to be conflicting with each other. Thus for most of the time, a set of solution is found rather than one single solution to represent the trade-off between different objectives [11].

### 2.3.1.3 Multi-Objective Optimization Concept

- **Difference between Single-Objective and Multi-Objective**

In most of the studies undertaken and algorithms developed for multi-objective optimization, they do not address the specialty of multi-objective optimization and thus avoiding the investigation of complexities involved in solving true multi-objective optimization problems. They treat multi-objective optimization in the same way as single-objective optimization by incorporating preference information before carrying out the optimization process. Hence, a multiple objective problem is transformed into a single-objective problem, which results in a single optimal solution. What makes multi-objective optimization problems special and difficult to solve is when optimal solutions corresponding to individual objective functions are sufficiently different. Then, the objectives become conflicting and a unique optimal solution is not available.

In fact, there are two fundamental differences between single-objective and multi-objective optimization. A solution in single-objective optimization can be easily compared with other solutions by evaluating the value of the objective function. However, a solution of multi-objective optimization cannot be directly compared with
others. Hence Pareto dominance concept is employed to facilitate the comparison. Pairwise comparison is conducted based on the Pareto dominance concept for comparing solutions in pairs to determine which one of the two is preferred. The result from the pairwise comparison would be either a solution dominates, indifferent or being dominated by the other solution.

In single-objective optimization, the feasible solution set for the problem is obviously ordered according to the objective function. However, when several objectives are involved in the problem, the feasible solution set is no longer totally ordered. The feasible solution set is only partially ordered after the adoption of the Pareto dominance concept. This characteristic leads to the fact that some solutions in the feasible set in multi-objective optimization are still incomparable. In pairwise comparison, these solutions are indifferent amongst others. Thus, another difference between single-objective and multi-objective optimization rests on the resulting optimal solution. A single optimum solution can be found in single-objective solution; while the set of optimal trade-off solutions are considered to be the best solutions in the multi-objective optimization [14].

- **Concept of Pareto Dominance and ε-Pareto Dominance**

In case of multiple-objective optimization, it does not necessarily present a solution that is better than other solutions from the objective function, thus the value of an objective function cannot be used for comparison. The concept of Pareto dominance is used by the research community in comparing solutions within the feasible set. Fonseca and Fleming (1993) pointed out that when there is no preference information incorporated into the objectives beforehand, and Pareto dominance concept is the only way to determine if one solution is better than the other solution. Two solutions are compared on the basis of whether one solution dominates the other solution or not.

In adopting the concept of Pareto dominance, pairwise comparison between two solutions \((x \text{ and } y)\) is to consider the following two conditions.
Chapter 2 Literature Review

(1) \( x \) is no worse than \( y \) in all objectives

\[
\forall i = 1,2, \ldots, n \quad f_i(x) \geq f_i(y) \quad (2.4)
\]

(2) \( x \) is strictly better than \( y \) in some sense

\[
\exists i \in [1,n] \quad f_i(x) > f_i(y) \quad (2.5)
\]

If both the above conditions are met, solution \( x \) can said to be dominant solution of \( y \) or solution \( y \) is dominated by solution \( x \). When any of the two conditions are violated, neither solutions dominate or be dominated in this comparison. The two solutions are indifference with each other [15][30].

![Figure 2.1. Relationship of solutions in the objective space](image)

A picture regarding the relationship of solutions is given in Figure 2.1. The outcome from the pairwise comparison would only lead to three distinct results, namely, dominate, indifferent or being dominated. In this figure, a black dot represents a solution within the feasible set. The solutions in region 1 encapsulate solutions which dominate the solution concerned. The solutions in region 3 correspond to solutions which are being dominated by the solution concerned.
Figure 2.2. Example illustrating Pareto comparison in the objective Space

All solutions inside the region 2 are solution that is indifferent with the solution concerned. An illustrative example is provided in Figure 2.2. For instance if there are two objective functions \( f_1 \) and \( f_2 \) and both are to be minimized. If there are five solutions namely A, B, C, D and E which are shown in Figure 2.2. It is obvious that solution A dominates all other solutions (B, C, D, and E). It is better than all other solutions in all objective functions. Comparing solution B and solution C, the first condition for Pareto dominance is violated. Hence, they are indifferent to each other. In the comparison between solution B and solution D, they have the same value in \( f_1 \) but solution B is definitely better than solution D in objective function \( f_2 \). Thus both dominance conditions are still satisfied and solution B dominates solution D.

The adoption of Pareto dominance concept only provides a marginal order for the feasible solution set in multi-objective optimization. Following this approach will still leave a large number of solutions that are incomparable, and this may hinder the search for the optimal solution set. \( \varepsilon \)-Pareto dominance is thus established in the operations research community to deal with this case. The conditions required for one solution to dominate another are relaxed to improve the comparability in the evaluation. Various forms of \( \varepsilon \)-Pareto dominance have been proposed in the literature amongst which the multiplicative form has received the largest attention. A solution \( x \) is said to be \( \varepsilon \)-
dominating another solution \( (y) \) if and only if it fulfills the following two conditions:

\[
(1) \quad \forall i = 1, 2, ..., n \quad (1 - \varepsilon)f_i(x) \geq f_i(y) \quad (2.6) \\
(2) \quad \exists i \in [1, n] \quad (1 - \varepsilon)f_i(x) > f_i(y) \quad (2.7)
\]

The value of \( \varepsilon \) is usually called the indifference threshold and is chosen to be the maximum change in value [15].

- **Pareto Optimality**

Based on the concept of Pareto dominance, the optimality criterion for multi-objective optimization can then be introduced. Pareto optimality can be defined as the best solution that could be achieved without disadvantaging at least one objective by using the concept of Pareto dominance.

![Figure 2.3 Pareto Optimality in an objective space](image)

All solutions in the feasible solution set undergo pairwise comparison to determine the dominating solutions, and those solutions that are being dominated by which solutions. Solutions that only dominate others but are not themselves being dominated by other solutions are classified as non-dominated solutions. For these solutions, no improvement in any objective function is possible without sacrificing the optimality of any other
objective functions. Thus they are the Pareto optimal set of solutions. In general, a group of non-dominated solutions will be found after the evaluation and the whole set of solution is said to be the Pareto front of that particular multi-objective problem. The task of multi-objective optimization is the process of finding the Pareto front. A simple idea of Pareto optimality can be visualized in Figure 2.3 in which the shaded region represents the entire feasible solution set, and the black line at the left bottom part of the region represents the corresponding Pareto optimal solution set [16].

2.3.2 Multi-Objective Optimization Algorithms

In this section, an overview of classical methods for multi-objective optimization is first presented, and amongst the different algorithms, evolutionary algorithm has attracted much interest from the research community. Research literature regarding the use of evolutionary algorithm is then reviewed with particular attention paid to different design approaches of these algorithms. Other major issues regarding the development of multi-objective optimization algorithm are also discussed at the end of this section.

2.3.2.1 Classical Approaches

There are two commonly-used classical methods for tackling multi-objective optimization problems (Deb, 2005).

The first one is generally known as the aggregate approach, which aggregates the set of objectives into one single objective function. This approach converts the multi-objective optimization problem to a single-objective optimization problem. The other is the constraint method, which is proposed by (Haimes et al., 1971). This method reformulates the multi-objective optimization problem to just optimize one of the objective functions and place constraints on all remaining objective functions (Deb, 2005). This approach again converts the problem into a single-objective optimization problem.
• **Aggregate Approach**

The most popular aggregate approach combines the set of objectives usually with weighted summation of the individual objectives with user-defined weights:

\[
\text{Minimize } F(x) = \sum_{i=0}^{n} w_i f_i(x)
\]  

(2.8)

where \( w \) represent the weighing of the objectives.

These weights require the input from the decision maker on the relative importance of the objective. Thereafter, algorithms for single-objective optimization can be employed to solve the problem. When such an aggregated objective function is optimized, it is possible to obtain one particular solution for the pre-defined weighing. This is often the simplest approach and is probably the most widely used classical approach.

Although this approach is simple, it introduces a difficult problem. The value of the weights is sometimes difficult to assign. Fundamentally, the values should depend on the relative importance of each objective in the context of the problem. However, the setting of the weights requires preliminary knowledge about the situation where the decision maker may not know about any priori information. The result from such an approach is also highly subjective as the result is heavily dependent on the setting of the weights. Thus the most serious weakness of this approach is that they may not find a good trade-off solution to all objectives.

Typically, a set of trade-off solution is preferred rather than one single solution. Aggregating approach can generate a set of trade-off solutions. A change in the preference information should result in a different trade-off solution. Thus by changing the weights, a different set of trade-off solution can be obtained. However, such approach cannot assure the uniform distribution of the trade-off solutions along the Pareto front as a uniform choice of weights do not necessarily find a uniform set of solution on the Pareto-front front (Deb, 2005).
• **Constraint Method**

Constraint method is proposed in (Haimes et al., 1971). The basic concept of it is to optimize one of the objective functions and place constraints on all remaining objective functions. The approach converts the problem into a single-objective optimization problem as follows:

\[
\text{Minimize } f_\lambda(x) \quad (2.9)
\]

\[
\text{subject to } f_i(x) \leq \varepsilon_i, \quad i=1,2,...,n \text{ & } i \neq \lambda
\]

where $\varepsilon$ represent the upper bound of the objectives and converted into constraints.

This method only leads to one single solution rather than a set of compromised solutions. This method requires the selection of proper objective to be the objective function for the constraint method. Right selection is very crucial and influential to the difficult however such selection is very difficult to be accurate. Another obvious difficulty of this method rests on the choice of the value of $\varepsilon$ (Deb, 2005). Too large a value may lead to a poor solution, but too small a value may result in no solution being found by the constraint method. The setting of $\varepsilon$ is again a difficult task which assume preliminary knowledge about the problem.

Classical methods are simple, but are often difficult to deploy in real life. For this reason, a number of stochastic search methods have been developed for solving multi-objective optimization problems. These stochastic methods do not guarantee the finding of the true Pareto optimal set, but they try to approximate a good set of solutions which is close to the optimal solution set. Evolutionary algorithm, a type of stochastic methods, receives great interest for solving multi-objective optimization problems.
2.3.2.2 Evolutionary Algorithms

In previous sections, several classical multi-objective optimization methods are introduced. These methods try to turn multi-objective problems into single-objective problems. However in many real-world multi-objective problems, suitable solutions are hardly being found by such classical methods since the objectives are usually inter-conflicting. Combination is not a good idea for capturing the characteristic of all the objectives. Evolutionary algorithms in essence recognize the specificity of multi-objective optimization problems which can work simultaneous with all objectives and result in a group of trade-off solutions which is more uniformly distributed and close to the optimal Pareto optimal set.

Evolutionary algorithms are a popular tool for multi-objective optimization. Over the past decade, a number of evolutionary algorithms based multi-objective optimization algorithms have been proposed in the literature (Coello Coello, 1999; Zitzler et al., 2000; Zitzler et al., 2002). Some literature even suggests that evolutionary algorithms perform better than other search strategies in the field of multi-objective optimization.

- **Fundamental Concepts and Principles**

  Evolutionary algorithms are well suited to multi-objective optimization. They have received growing interest owing to their simplicity in nature as an algorithm and their ability to capture multiple solutions in one single run. Solutions to a multi-objective optimization problem consist of a set of trade-off solutions in the search space. The population-based nature of evolutionary algorithm is particularly matched with these problems. It allows a set of trade-off solutions to be generated in a single execution of the algorithm which comprises the trade-off solutions among all objectives and also allows pairwise comparison based on Pareto dominance to objectively evaluate the quality of the solutions within the population. The fundamental benefit of this over the multiple-start of the classical method is the cooperative search in the search space that reduces the redundancy and thus saving on the total number of evaluations used.
Another key benefit offered by the evolutionary algorithm is that it can exploit similar solutions quickly by means of mutation and recombination. Similar solutions are rapidly evolved and evaluated. Thus the algorithm requires much less computation resources in finding the whole set of trade-off solutions. The popularity of evolutionary algorithms is also due to their ability to discover good solutions efficiently for complex optimization problems involving different characteristics.

Using exact search methods for solving multi-objective optimization problems is computationally expensive and is often infeasible in finding the Pareto-optimal set because of the inherent complexity. Using classical methods requires the preliminary knowledge of the problems being optimized and this is not always the case. Due to the ease of implementation, the reduced need in computational complexity, and the little knowledge requirement, evolutionary algorithm is more suitable for general multi-objective optimization.

The term evolutionary algorithm does not have a precise definition. The term is usually used to represent a class of approaches which use the ideas from natural evolution including genetics and immunology. Evolutionary algorithm imitates the natural evolution process to guide the search and optimization processes. Those approaches are generally population-based which normally processes on a population of candidate solutions. Candidate solution of the problem plays the role as an individual in the population. Those individuals in the population are subsequently operated by two major operators of the evolutionary algorithm, namely, selection and variation.

Selection process usually involves the fitness determination for the individual which evaluate the quality of the candidate. Candidates with a better fitness value will normally have a higher chance to survive and reproduce in the evolutionary process. While variation aims at continuously modifying the selected candidates in order to create potentially better solutions. Mutation and recombination operators are employed by the
variation process.

- **Working Principle of Evolutionary Algorithm**

By analogy to the natural evolution, candidate solutions are the individuals and the set of solution candidates is the population. Each individual represents a possible solution in the search space encoded with certain appropriate structure. Figure 2.4 gives a general picture of the processes in a basic evolutionary algorithm for multi-objective optimization. For a typical evolutionary algorithm, the first step is to create a population of candidate solutions comprising the initial population. These candidate solutions may be generated randomly or created according to certain predefined schemes, requirements or principles. It forms the starting point of the evolution process. Then a loop consisting of several steps including fitness assignment, selection and variation is executed for a certain number of times and terminated with satisfying the criteria on a predefined number of generations.

In contrast to single-objective optimization, where an objective function and fitness function are often the same, in multi-objective optimization, individual candidate solutions are generally incomparable by using the objective function directly. Certain measures have to be implemented in the algorithm to compare different candidate solutions effectively. Fitness function is applied to the candidate solutions and any subsequent offspring. Fitness value is assigned to each of them thus making the comparable in some sense. This can help to decide which solution is a better solution to evolve with further resources.
Fitness assignment scheme is one of the major differences between different proposed evolutionary algorithms. In the main loop of the algorithm, candidate solutions for the
next generation are selected with regards to their fitness value. The selection is usually with a bias towards the higher fitness candidates. The selected solutions are reproduced and then subjected to variation.

Variation is applied in the forms of mutation and recombination to generate new solutions in the search space by changing the existing solution. Mutation acts on a candidate and results with a new candidate. Mutation is performed through random tweaks to the candidate solutions according to a given mutation rate. Recombination acts on the two or more selected candidates which results in one or more new candidates. Recombination is done through recombining between the involved candidates and result in creating new candidates. In the selection process, several of the new candidates together with some of the old candidates are combined and then selected by certain selection strategies as the population of the next generation. The overall goal of the selection process is to increase the overall quality of the population and retain portions of candidates to direct the search process towards a higher potential region. Therefore, low-quality individuals are generally removed from the current population, while high-quality individuals are retained for future searching process. The main loop iterates with the process of selection and variation, and this should move the population nearer to the optimal mix.

- **Development of Evolutionary Algorithm**

A number of multi-objective evolutionary algorithms have been proposed in the literature including VEGA (Schaffer, 1985), WBGA (Hajela & Lin, 1992), MOGA (Fonseca & Fleming, 1993), NPGA (Horn et al., 1994), NSGA (Srinivas & Deb, 1994), NSGA-II (Deb et al., 2000), R-NSGA-II (Deb et al., 2006), RDGA (Lu & Yen, 2003), SMS-EMOA (Beume et al., 2007), SPEA (Zitzler & Thiele, 1998), SPEA2 (Zitzler et al., 2001), IBEA (Zitzler & Künzli, 2004), PAES (Knowles, 1999), M-PAES (Knowles & Corne, 2004b), PESA (Corne et al., 2000), PESA-II (Corne et al., 2001), $\mu$ GA (Coello Coello & Pulido, 2001), $\mu$ GA2 (Pulido & Coello Coello, 2003). The first evolutionary
algorithms attempted to obtain a set of trade-off solution were proposed by Schaffer in 1985 (Schaffer, 1985). Subsequently, some new approaches designed with similar aims have been proposed. In these schemes, individuals of the population are selected according to individual objective or some fitness functions which involve weighing of objectives. These approaches did not employ the concept of Pareto dominance in the evolution process. These early attempts were subsequently named as the non-Pareto approaches.

Unlike those early attempts, the majority of the recent approaches employ the concept of Pareto dominance in some variant forms. In essence, there are two major lines of fitness assignment methods being employed in various forms in many algorithms. Research efforts have focused also on the selection process to maintain both the proximity and diversity within the population. Elitism strategies adopted in multi-objective optimization evolutionary algorithm was another breakthrough in this field. Techniques for refinement of the Pareto-based fitness assignment have also been the focus of contemporary research to increase the comparability of the non-dominated solution found in the evolution process.

Actually, most of the proposed algorithms in the literature involve similar features as the basic evolutionary algorithm discussed above. The major differences between these algorithms normally fall within five major areas, selection, fitness assignment, variation scheme, population diversity and elitism, and memory archiving. Algorithms adopt different strategies in these five different areas. In fact, these five areas are inter-related to enhance both the convergence and diversity in the resulting solution set.

- **Selection**

Selection operation within the evolutionary process is aimed to guide and steer the search process towards the true Pareto-optimal set. It addresses the problem of which solutions should preserve in the evolutionary process and aims at picking promising solutions to
exploit the search process. The main role of the selection thus is to regulate the evolution population. The search operation in the evolutionary algorithms is usually unidirectional. Selection strategies help to guide the search process towards the Pareto-optimal front and reduce the redundancy appeared in the undirected search. The selection operation is involved in two major processes within the evolutionary algorithm, the selection for reproduction and the selection for survival. Some algorithms combined these two processes into one operation. However, most of the algorithms in the literature separately recognized these two processes and focused on the different characteristics of these two processes.

Concerning the selection for reproduction, algorithms including WBGA (Hajela & Lin, 1992), MOGA (Fonseca & Fleming, 1993) and NSGA (Srinivas & Deb, 1994) adopted the ranking selection method. It is by far the simplest approach for selection. Such approach was also adopted by many researchers in the selection for survival. Individual solutions within the population are sorted according to their fitness value. The best proportion of solutions is selected to form the mating pool. The size of the selection was usually controlled by constant parameters predefined before the start of the evolutionary process. This simple approach is implemented in many evolutionary algorithms in either the selection for reproduction or the selection for survival.

Horn et al. (1994) proposed the NPGA which differs from the above algorithms in the selection operator. Binary tournament selection was implemented in NPGA in the selection for reproduction. Each time two individual solutions are chosen randomly from the population for the tournament selection. Individual solutions are competing on the certain indicator for the selection. Dominance relation was adopted by NPGA for such comparison. If one of the competing solutions dominates the others, the former is thus selected. Tied ranks are decided by diversity measure. Fitness sharing was implemented in NPGA for this comparison.
Binary selection invites higher degree of randomness into the searching. These randomness help to preserve diversity of the solutions in the population, but it will increase the redundancy search which will use up more computation resource. Binary tournament selection is subsequently used in some variant forms by many algorithms in the selection for reproduction (Deb et al., 2000; Zitzler et al., 2001; Zitzler & Thiele, 1999). The differences are mainly on the use of the measure for the comparison. The first measure is related to the convergence and most of the algorithms adopt similar strategy for this comparison (Deb et al., 2000; Zitzler et al., 2001; Zitzler & Thiele, 1999). The second measure is focused on the diversity area. In particular, NSGA-II adopted the crowding distance for the comparison in situation of tied ranks.

Local search scheme was proposed in PAES (Knowles, 1999). The scheme performs only local search with pure mutation operation. PAES represents the simplest possible non-trivial algorithm capable of generating diverse solutions in the Pareto optimal set. PAES requires much less computation resources in the evolutionary process and yet capable of generating diverse solutions in the Pareto optimal set. In the evaluation of the local search scheme proposed, simple hill climbing technique is able to achieve a good solution set.

PESA and PESA-II further the local search scheme to the selection process to bias the selection onto the less crowded region (Corne et al., 2001; Corne et al., 2000). The enhancement of diversity is the major aim in selection for reproduction operation from the family of PESA algorithms. PESA adopts the binary tournament selection originally proposed in (Horn et al., 1994) for the selection for reproduction. The comparison was based on the density of the hyperboxes in which the solution situates. The major innovation of PESA-II is on the use of the region based selection in the selection for reproduction. Instead of selecting individuals in the population, hyperboxes in the search space are selected. In this new approach, the binary tournament takes place between hyperboxes in the objective space rather than in between individual solutions. The comparison between hyperboxes focused on the density of the hyperboxes that is the
number of solutions within it. This strategy was shown to be capable of ensuring a better spread of development along the Pareto front than individual based selection. The idea behind is that for tournament selection, individuals in crowding region are more likely to be chosen for the tournament. Yet one thing to note is that region base selection requires the specification of the hyperbox dimensions. This would causes problem similar to other algorithms that have to specify the sensitive parameter.

In the selection for survival, convergence and diversity dimension are considered in different situations. When there appear only a few non-dominated solutions in the current population, convergence promotion is the main focus. In cases where there appear more non-dominated solutions than can be retained in the next generation, diversity measure tends to become the dominant consideration to reduce the size of the population to the accepted magnitude. In the early evolutionary algorithms for multi-objective optimization, all of the child solutions that result from each generation are considered as the population for the next generation. Such approaches for selection for survival is a non-elitist approach and promising solutions may lose in the evolutionary process. For the non-Pareto approach VEGA (Schaffer, 1985), selection is done separately for each objective. Populations of the next generation are selected from the current generation according to each of the objectives independently. This selection strategy fails to recognize the trade-off nature of the Pareto-optimal front. The selection is biased towards the best solution of individual objective and thus VEGA just converges to solution being only optimum to individual objective.

Some of the latest evolutionary algorithms do not separately implement the selection for reproduction and the selection for survival. They combine these two operations into one, the one selected for survival into next generation are evolved in the next generation. Some other recent evolutionary algorithms maintain the population size by means of the diversity consideration to bias the search process in favor of the less crowded regions. The selection for survival is thus rest on the diversity measure adopted by the algorithm.
for such purpose. Detail description of different diversity measure can refer to later part of this section. Zitzler et al. (2001) combined the selection for survival with the process of memory archiving. The whole group of non-dominated solutions undergoes the memory archiving to form the secondary population for each generation which contains both the convergence and diversity measure. The secondary population is then considered to be the population survived into the next generation. Such strategy employs the technique of memory archiving for the selection for survival. Detail descriptions of different memory archiving approaches can refer to later part of this section.

The advantages and disadvantages of the different schemes can broadly be considered being the trade-off between the capability and the potential to enhance convergence and diversity in the population subset and the computational complexity of the approach. The removal of non-dominated solutions at the selection for survival stage may cause deterioration in the quality for the algorithm where promising solution may loss in the current population. Solutions that are lost in certain generation may also be reproduced by the algorithm later in the process which increases the redundancy search of the algorithm.

- **Ranking and Fitness Assignment**

The major function of the selection strategy in the evolutionary process is to guide the search process towards the Pareto-optimal set. The search should therefore be able to bias towards high potential solution in the whole population. However, individual solutions for multi-objective optimization are almost incomparable. Fitness assignment function is thus applied to all individual solutions to assign each of them with a value so as to facilitate the comparison in between individuals. In general, a fitness assignment function gives individual solution a ranking or at least a partial order with reference to the whole population. Thus the fitness assignment function provides the selection operator a certain measure to select solutions which deserve to remain in the evolution process. Therefore in all studies of multi-objective evolutionary algorithms, fitness assignment function is
one of the critical parts in the algorithm design.

Different fitness assignment strategies are being proposed, namely, Pareto-based approach and non-Pareto approach. It is generally agreed that Pareto-based approaches dominates most of the recent multi-objective optimization evolutionary algorithms.

In terms of the non-Pareto approach, algorithms of this category usually employ certain preference information for aggregation as fitness assignment. Aggregation method for fitness assignment is in fact another classical approach to assign fitness to individual solution besides changing a multi-objective optimization problem into a single-objective optimization problem. In aggregation methods, fitness calculation corresponds to a linear function of the objectives of the problem with different pre-defined weighing. Preference information is already incorporated into the fitness calculation. Schaffer modified the aggregation method for VEGA (Schaffer, 1985). A linear function of the objectives where the weights changes on the distribution of the population at each generation was used for the fitness assignment. These non-Pareto approaches have the advantage of efficient where the fitness assignment can be done separately and individually for each solution. The fitness calculation does not require the consideration of the whole group of current population. However, these approaches generally assign different fitness ratings to non-dominated solutions which are contradict to the definition of non-dominance. Worst still, there may have time that a non-dominated solution is assigned with a worse fitness rating than a dominated solution. Such an approach is definitely biased towards the preference region with results in one single solution or a series of solutions around the same area of the search space rather than the approximate Pareto front. For the aggregation method, it requires the input from a decision maker the relative importance of the objectives. This task is always difficult to a user in choosing an appropriate set of weighing to generate a reasonable solution as the objectives may be inter-conflicting and incomparable.
Pareto-based approaches make use of the concept of Pareto dominance explicitly in assigning fitness to individual solution with reference to the whole population. The assignation of fitness for individual solution does not draw any direct reference to the objective function values. In terms of the Pareto-based approaches, three Pareto-based schemes are primarily employed by the multi-objective evolutionary algorithms. Following the terminology of Zitzler et al. (2002), these three Pareto-based schemes are named as dominance rank, dominance count and dominance depth. Details of these three schemes are as follow:

1. Dominance rank: The number of solutions in the population that dominate the solution under consideration.
2. Dominance count: The number of solutions in the population that are dominated by the solution under consideration.
3. Dominance depth: The group of the solution from the non-dominated sorting of the population.

Dominance depth, in the terminology of Zitzler et al., is one of the most favored within the research community of the multi-objective optimization evolutionary algorithms. The scheme is commonly named as the non-dominated sorting scheme. This strategy, which focused on the non-dominance level with reference to the whole population, was proposed by Goldberg (1989). The idea is to classify individual solutions into different levels based on the non-dominance relationship. This fitness assignment scheme segregates the population into different ranks by first finding the current non-dominated individuals in the population, and labels these solutions as the first level or assigned the fitness as 1. These individuals are removed and the next group of non-dominated individuals in the remaining population is then identified, and assigned as the next level and removed. The process continues until the entire population has been classified into different levels.
This approach was subsequently adopted by several algorithms including NSGA (Srinivas & Deb, 1994). The major shortfall of this approach is that this assignment method usually results with a large group of solution having the same fitness value which hinders the comparison between them. In NSGA-II (Deb et al., 2000), density information is brought into the fitness assignment as a secondary factors to increase the comparability of the non-dominated solutions obtained. Crowding distance assignment method is applied to introduce density information into the fitness assignment. Thus if both of the solutions refer to the same non-dominance level, NSGA-II prefers the solution that are located in a region with fewer number of individuals nearby.

The scheme of dominance count is proposed by Fonseca and Fleming (1993). Their proposed algorithm MOGA adopts the dominance count as the fitness assignment function by counting the number of individual in the current population that dominates that respective individual. All non-dominated individuals are assigned with fitness value 1, and the remaining individuals are penalized according to the number of individual that dominated it. The authors noted that this scheme can assist in preserving the diversity of the population. Individual is usually dominated by contiguous solutions in the objective space. The higher the number of contiguous solutions the higher is the chance that the individual will be penalized by the contiguous solutions. Based on the dominance count scheme proposed in MOGA, Lu and Yen (2003) further refined this counting method and proposed an automatic accumulated ranking strategy for the fitness assignment in RDGA. A scheme of dominance count is used to calculate the rank of each individual and the fitness of individual solution is the sum of the ranks of the solutions that dominate it. Bad solutions are further penalized by this scheme.

Non-dominated individuals in general are assumed to be of the same importance in the evolution process, so they are usually assigned with the same fitness value. Zitzler and Thiele (1999) introduced the dominance rank in an attempt to assign different importance to the non-dominated individuals so that the penalization imposed on different non-
dominated individuals would be different. Such an approach was implemented in SPEA. This approach takes into account both dominance count and dominance rank in the fitness assignment. For each individual, both dominating and dominated solutions for the respective individual are considered. Each non-dominated solution is assigned a strength value, which at the same time represents its fitness value. The strength value is referenced to the number of solution that is dominated by the respective individual relative to the size of the population. The fitness of individual in the population is calculated by summing the strength value of all non-dominated solutions that dominate the respective individual. According to (Zitzler et al., 2001), the potential weaknesses of such fitness assignment approach is that individuals that are dominated by the same archive members have identical fitness value regardless of whether they dominate each other or not. As a consequence, the selection pressure is decreased substantially. This fitness assignment strategy is further refined in SPEA2. Besides density information being introduced in the fitness calculation, the strength value is calculated for the whole population and the fitness calculated are referenced to the whole population rather than only the individuals in the archive. These refinements aimed to avoid the situation that individuals that are dominated by the same archive have identical fitness values and all non-dominated individual are assigned with fitness ‘0’ rather than the strength value so as to be inline with the definition of non-dominance. However, this method has a known shortfall. This fitness assignment strategy has inherent bias on solutions that are dominated more individuals in the population. This bias actually is in contrast to the definition of non-dominance where the non-dominated solution should have equal value (Deb, 1999).

Generally, most of the recent algorithms employ the density information as a secondary factor in the fitness assignment process to direct the evolutionary process. Some algorithms have put forward the use of preference information for refining the fitness assignment scheme. Zitzler and Kunzil (2004) tried to integrate the preference information into the multi-objective search and thus proposed the Indicator-based Evolutionary Algorithm (IBEA). The main idea is first to input the preference
information in the form of binary indicator by the user and these binary indicators are then used as a measure in the fitness assignment process that directs the selection process. Deb et al. (2006) modified NSGA-II to include the preference information in the search process. They employed the preference-based strategy with the NSGA-II methodology and thus proposed the reference-point-based NSGA-II (R-NSGA-II). The algorithm requires the user first to specify a reference point before the evolutionary process of the multi-objective search. The diversity information used in the algorithm domain is substituted by the preference operator. Solutions that are close to the reference point are emphasized in the evolutionary process. However this type of algorithm will have similar problems as the classical approach that required the input from a user about the priority on the objectives. Such a priority is very difficult to estimate in most cases.

There is currently no clear evidence for the merits of any of these approaches over another. However, the complexity issue of the assignment process has to be noted. In Pareto-based approach, generally, no preference information is required in the algorithm. However individual fitness cannot be valued individually as it is related to the whole population. The method of dominance requires one evaluation of all population for each individual. Complexity of SPEA fitness scheme is higher because it has to measure the strength and the fitness separately which both require an evaluation of all population or part of the population. The complexity of the Goldberg method in non-dominance level is also difficult to estimate. It requires several round of evaluation. The difference between the best case and the worst case is quite large. Generally speaking, Pareto-based approaches involve significantly more computation overhead than other approaches. The fitness assignment strategy is still a major consideration in the algorithm design for the search of near-optimal and near-complete Pareto fronts with reasonable complexity.

- **Variation Scheme**

Variation operators are applied to the pool of individual solutions that are selected from the selection for reproduction operation. The aim of the application is to vary individual
solution by changing parts of the associated decision variables in an attempt to create new child solution that is better in certain extent than the original parent solutions.

Two variation operators are usually employed by the variation schemes – the mutation and recombination operators. The mutation operator is usually employed to bring in random factors into the creation of child solutions. It modifies the candidate solution by changing part of the decision variables according to a predefined mutation rate. The variation is of random nature which helps prevent premature convergence. It also ensures that the probabilities of reaching any point in the search space is never zero and hence ensures the theoretical convergence of the algorithm towards the optimal solutions. In contrast to the mutation operator, recombination operator is usually employed because of its potential to exploit similar solution within the search space. The recombination operation is a technique for sharing information among individual solutions. To mimic the stochastic nature of evolution, the recombination operator takes in a certain number of parents and generates a number of child solutions by merging parts of the associated vector of the parent solutions. Detailed studies of recombination and mutation operators can be found in (Deb et al., 2007; Herrera et al., 1998; Raghuwanshi & Kakde, 2004).

Most of the proposed algorithms focus on the design domain of the algorithm. These algorithms are designed in a way that any scheme of mutation and recombination operator can be deployed in a straightforward manner as their corresponding variation scheme. In case when these algorithms benchmark their study with others, common variation operators are employed to increase the fairness of the comparisons between algorithms.

In generally, polynomial mutation and simulated binary crossover are introduced as the mutation and recombination operators for the variation scheme repeatedly. The major reason behind the choice of variation scheme is that these two strategies have been
adopted in many other evolutionary algorithms including most of the algorithm used for the comparison in the benchmark study. The following explanation of the technique is based on the description in (Khare, 2002).

The simulated binary crossover imitates the working principle of a single point crossover operation for solution of binary coding. Similar to the polynomial mutation operation, individual solutions undergo the simulated binary crossover operation have to be real coded. It generally takes in two parent solution and generates two child solutions based on the parent solution. This operator has biases on solutions which are nearer to the parent solutions.

- **Population Diversity**

In designing a multi-objective optimization algorithm, the aim is to find out the Pareto-optimal solutions, and at the same time these solutions are uniformly distributed in the optimal set. Unfortunately, it is known that the diversity of the population is lost during the evolution process because of the randomness nature. This effect is commonly known as genetic drift in the literature (Mahfoud, 1994). An evolutionary strategy generally converges to one single solution rather than a set of distributed solutions. The loss of diversity owing to the genetic drift has to be restrained.

Fitness sharing technique is one of the widely used approach to maintain the diversity and prevent the genetic drift in multimodal function optimization (Goldberg, 1989). NPGA is one of the first algorithms in multi-objective optimization to directly address the diversity of the solution set (Horn et al., 1994). In NPGA, a niche radius is first chosen beforehand, and then the niche count is calculated by summing the individual within the niche circle defined by the niche radius. Fitness sharing technique is deployed when the fitness of the intended solutions are the same. The algorithm in this case thus has a bias towards the less crowded area. This approach is based on the idea that individual solutions in a particular niche have to share the existing resources with other individuals. Therefore the
fitness value should be degraded when more and more individuals are located near the
individual solution. This proximity is defined in terms of a distance measure.

This approach was subsequently adopted with in some variance form by several
algorithms including MOGA (Fonseca & Fleming, 1993), and NSGA (Srinivas & Deb,
1994). The major difference is on the niche measurement. The major shortfall of this
technique is the need for specifying a sharing parameter. This parameter needs to be set
carefully as the parameter greatly influence the actual diversity of the resulting solution
set. However, the setting of this parameter requires prior knowledge about the problem to
be optimized which is almost impossible in multi-objective optimization. Moreover, the
sharing technique may further slow down the speed of the population to evolve to a
correct direction to the optimal solution (Loughlin & Ranjithan, 1997).

Recent evolutionary algorithms maintain the diversity by directly bias the search
process to the less crowded region through incorporating density information into the
fitness assignment. The fitness of a solution located in a crowded area is degraded and
thus reduced the chance of being selected. Many advanced algorithms including SPEA2,
NSGA-II, and PAES use some form of density dependent fitness assignment method to
bias the search (Deb et al., 2000; Knowles, 1999; Zitzler et al., 2001). The major
difference rests on the method of density estimation within the algorithms. In NSGA-II,
density estimation takes into account the size of the largest cuboid enclosing the
individual without including any other individual in the population whereas the density
information is only used when the fitness of the individuals having the same fitness value
like NSGA and NPGA. Nearest neighbor technique is used in SPEA2, the fitness of an
individual is degraded by the reciprocal of the distance to its nearest neighbor (Zitzler et
al., 2001). The crowding strategy in PAES works by forming hyper-grid which divides
the search space into numbers of hyper-boxes. Each individual is assigned with a squeeze
factor which is equal to the number of other individuals in the archive within the box.
Squeeze factor is only used in archive update where the choice of removing an individual
in the archive is based mainly on this factor.

- **Elitism and Memory Archiving**

Elitism was adopted by the researchers in the field of evolutionary algorithm to prevent the loss of promising solution in the evolution process. This is due to the stochastic nature of the evolutionary algorithm where good solution is not guarantee to remain in the evolution population until termination. Elitism is usually implemented with the setup of a memory population external to the evolution process which composed of the non-dominated solutions found so far. Memory population is also named as the secondary population, external population or archive in some other research studies.

Study on the comparison between the elitist algorithms and the non-elitist algorithms had been conducted on variety of multi-objective problems (Zitzler et al., 2000). The study had proved that algorithm with elitism outperformed other algorithms. Because of this favorable study towards the adoption of memory population, the use of elitism is very popular in recent year. Thus the use of such external population is of no doubt important, but the question is then rest on how best to utilize such external population. The memory population is mainly effected on two aspects of the algorithm. First, they act as the major storage in the algorithm for the non-dominated solutions. These solutions later represent the results of the evolutionary process. Other studies put a step forward where this storage is further employed as a pool of elite to direct the subsequence evolution process. With this aim, the members of the memory population are generally involved in the selection process for the next population where they will then subject to variation.

Memory population is originally adopted as storage space for all non-dominant solutions found so far in the evolution process. As the number of non-dominated solution in the continuous objective spaces is of an infinite size, the capacity of the memory population should be ideally infinite to store all non-dominated solutions. Such design is not realistic in practical situation. Thus in practical consideration, the capacity of the memory
population must be bounded in size so that a realistic number of promising solutions are maintained in the evolution process. Certain archiving algorithms were therefore proposed to reduce the number of solutions in the memory population. The memory population, as a representation of the results of the evolutionary process, should itself being a good approximation to the true Pareto-optimal set and as the size is bounded; the memory population should also be a well-distributed set of solution which covers the whole Pareto-optimal set comprehensively. As a result the archiving algorithms applied should achieve both tasks simultaneously, they should enable the convergence in the memory population and they should also focus on having a good distribution of solutions within the set. In addition, as the memory population is sometimes used to influence the evolution process, non-dominated solution having a higher potential to exploit Pareto-optimal set should ideally be recognized and retained in the memory population to direct the search.

Most elitist algorithms try to make use of a combination of convergence and diversity consideration to choose individual to retain in the memory population at each evaluation. The convergence consideration is usually applied with the dominance criterion that only non-dominated solutions found so far can be chosen to join or to retain in the memory population. This means dominated members within the memory population are removed from the memory population immediately. Most of the archiving algorithms proposed in the literature, including those in PAES, PESA and SPEA2, implemented such dominance criterion as the first basis of which solutions should be kept in the memory population. However, the dominance criterion is in fact not sufficient individually to restrict the size of the memory population. Therefore, additional information is taken into account to limit the size of the memory population further. The elitist algorithms address especially the diversity of the memory solutions by different mechanisms as the second criterion to restrict the size of the memory population. Such an adoption aimed to maintain a well-distributed solution sets in the memory population so as to achieve the diversity consideration for the archiving.
A simple archiving approach is adopted by HEMO (Hu et al., 2003) where the diversity criterion is based only on crowding factor. If the memory population exceeds its maximum size, the crowding factors are recalculated for all non-dominated individual. Individuals with a higher crowding factor are removed to bring back the size back to the maximum value. Such an approach is easy to understand and simple to implement.

Elitist evolutionary algorithm for multi-objective optimization called PAES was proposed by Knowles (1999) where a region-based concept was introduced for the diversity criterion in the archiving. The approach was later modified by Knowles and Corne (Knowles & Corne, 2004a), and renamed as the Adaptive Pareto Archiving and being adopted by the PESA, PESA-II, $\mu$ GA and $\mu$ GA2 (Coello Coello & Pulido, 2001; Corne et al., 2001; Corne et al., 2000; Pulido & Coello Coello, 2003). In this approach, if the memory population exceeds its maximum size, then the acceptance of new child solutions depends on the region of the solution in the objective space. Solutions situated in less crown region are allowed to enter into the memory population whereas solutions in the crowded region are removed from the memory population. The objective space is divided into several divisions for each objective forming a number of hyperboxes in the space. Child solution that is non-dominated can only join the memory population if the hyperbox in which the solution situates is less populated than the most populated hyperbox in the current memory population. In this case, the child solution is included and a solution which is randomly selected within the most populated hyperbox is discarded from the memory population. This “less then” requirement in the original concept is amended to “less then or equal to” for PESA and PESA-II to increase the chance of alteration in the memory population. The number of subdivision in the objective space requirement in the original approach is determined by pre-defined parameters. These parameters are adjusted to be adapted to the evolution process in $\mu$ GA2. The value is set such that on average it never exceed three individuals per region and is never less than 1.5 individuals per region. The archiving algorithm proposed by Zitzler and Thiele (Zitzler & Thiele, 1999) applied different mechanisms for the diversity
consideration. If the number of non-dominated solutions exceeds the pre-defined capacity, clustering technique is adopted in SPEA to reduce further members of the memory population. Each individual is assigned into cluster. Clusters are then merges if the distance between clusters is the minimal within the current population. The merging process terminates when the number of clusters is smaller than or equal to the capacity of the memory population. One representative individual, which is the individual with the minimum average distance to all other solutions within the cluster, is selected from each cluster to form the new memory population. Such a mechanism is computationally expensive and the outer solutions which should be kept for comprehensive reason are always removed. Base on the problems identified, different memory updating mechanism is implemented in SPEA2 (Zitzler et al., 2001). An individual having the minimum distance to another random selected non-dominated individual is selected iteratively to remove from the memory population. Boundary solutions are thus preserved in the population. The run-time complexity is still a problem as it involves several numbers of iterations to complete the update process. This mechanism is later being applied to some other studies (Lu & Yen, 2003).

It is clear from the above discussion that most of the existing elitist approaches focus mainly on achieving a good distribution of solutions. The convergence criterion is fulfilled by just the dominance criterion to retain only the non-dominated solutions. There is no attempt to further evaluate the convergence aspect of the memory population. Approaches which focus only on distribution are in fact in danger of deterioration of the quality of solutions. The diversity preservation operations always emphasize on the less crowded regions of the non-dominated solutions and aim to maintain a spread among solutions. In fact, there is no way for these approaches to know which solutions are already Pareto-optimal. Promising solutions may actually be removed by the diversity preservation operations.

The proposed archiving algorithm by Laumanns et al. (2002) adopts adaptive
approximate value in the approximate dominance concept. This approach attempts to adjust the convergence criterion which is normally applied. If the memory population exceeds its maximum size, pairwise comparison is applied to the whole group of non-dominated solutions for approximate dominance concept with pre-defined approximate dominance value. However, such a strategy cannot control the capacity of the resulting solution set after the comparison as the suggested value of the approximate dominance value in the study is over-conservative leading to the small size of the resulting solution set. The size is always much smaller than the capacity of the memory population.

Furthermore, existing archiving methods often center on achieving a good representation for the Pareto-optimal. There does not appear any attempt to retain promising solutions that are later be used to direct the searching process. Well-distributed individuals may not be good individuals for evolution. Methods should be included in the archiving to retain individuals having a higher potential to evolve to better quality solutions. In the study for elitism in directing the evolution process (Laumanns et al., 2001) the authors suggested that the effectiveness of elitism in searching depends significantly on the mutation strength. Strong elitism together with a high mutation rate is required to achieve the best performance. Right combination of elitism and mutation rate is therefore the decisive factor for good performance.

2.4 Simulated Annealing Algorithm

Simulated Annealing (SA) was introduced by Kirkpatrick (1983) [3]. Simulated annealing is a generic probabilistic metaheuristic for the global optimization problem of locating a good approximation to the global optimum of a given function in a large search space [18]. It is a method of obtaining good solutions to difficult optimization problems [19]. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities). For certain problems, simulated annealing may be more efficient than exhaustive enumeration — provided that the goal is merely to find an acceptably good solution in a fixed amount of time, rather than the best possible solution [20].
The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

By analogy with this physical process, each step of the SA algorithm attempts to replace the current solution by a random solution (chosen according to a candidate distribution, often constructed to sample from solutions near the current solution). The new solution may then be accepted with a probability that depends both on the difference between the corresponding function values and also on a global parameter $T$ (called the temperature), that is gradually decreased during the process. The dependency is such that the choice between the previous and current solution is almost random when $T$ is large, but increasingly selects the better or "downhill" solution (for a minimization problem) as $T$ goes to zero. The allowance for "uphill" moves potentially saves the method from becoming stuck at local optima—which are the bane of greedier methods [20].

2.4.1 The basic Iteration

At each step, the SA heuristic considers some neighbouring state $s'$ of the current state $s$, and probabilistically decides between moving the system to state $s'$ or staying in state $s$. These probabilities ultimately lead the system to move to states of lower energy. Typically this step is repeated until the system reaches a state that is good enough for the application, or until a given computation budget has been exhausted [20].

2.4.2 The neighbors of the state

The neighbours of a state are new states of the problem that are produced after altering a given state in some particular way. For example, in the traveling salesman problem, each state is typically defined as a particular permutation of the cities to be visited. The
neighbours of permutation are the permutations that are produced for example by interchanging a pair of adjacent cities. The action taken to alter the solution in order to find neighbouring solutions is called a "move" and different moves give different neighbours. These moves usually result in minimal alterations of the solution, as the previous example depicts, in order to help an algorithm to optimize the solution to the maximum extent and also to retain the already optimum parts of the solution and affect only the suboptimum parts. In the previous example, the parts of the solution are the city connections.

Searching for neighbours of a state is fundamental to optimization because the final solution will come after a tour of successive neighbours. Simple heuristics move by finding best neighbour after best neighbour and stop when they have reached a solution which has no neighbours that are better solutions. The problem with this approach is that the neighbours of a state are not guaranteed to contain any of the existing better solutions which means that failure to find a better solution among them does not guarantee that no better solution exists. This is why the best solution found by such algorithms is called a local optimum in contrast with the actual best solution which is called a global optimum. Metaheuristics use the neighbours of a state as a way to explore the solutions space and can accept worse solutions in their search in order to accomplish that. This means that the search will not get stuck to a local optimum and if the algorithm is run for an infinite amount of time, the global optimum will be found [20].

2.4.3 Acceptance probabilities

The probability of making the transition from the current state \( s \) to a candidate new state \( s' \) is specified by an acceptance probability function \( P(e, e', T) \), that depends on the energies \( e = E(s) \) and \( e' = E(s') \) of the two states, and on a global time-varying parameter \( T \) called the temperature. States with a smaller energy are better than those with a greater energy. The probability function \( P \) must be positive even when \( e' \) is greater than \( e \). This
feature prevents the method from becoming stuck at a local minimum that is worse than the global one.

When $T$ tends to zero, the probability $P(e, e', T)$ must tend to zero if $e' > e$ and to a positive value otherwise. For sufficiently small values of $T$, the system will then increasingly favor moves that go "downhill" (i.e., to lower energy values), and avoid those that go "uphill." With $T = 0$ the procedure reduces to the greedy algorithm, which makes only the downhill transitions.

In the original description of SA, the probability $P(e, e', T)$ was equal to 1 when $e' < e$—i.e., the procedure always moved downhill when it found a way to do so, irrespective of the temperature. Many descriptions and implementations of SA still take this condition as part of the method's definition. However, this condition is not essential for the method to work, and one may argue that it is both counterproductive and contrary to the method's principle.

The $P$ function is usually chosen so that the probability of accepting a move decreases when the difference $e' - e$ increases—that is, small uphill moves are more likely than large ones. However, this requirement is not strictly necessary, provided that the above requirements are met.

Given these properties, the temperature $T$ plays a crucial role in controlling the evolution of the state $s$ of the system vis-a-vis its sensitivity to the variations of system energies. To be precise, for a large $T$, the evolution of $s$ is sensitive to coarser energy variations, while it is sensitive to finer energy variations when $T$ is small [20].

### 2.4.4 The Annealing Schedule

The name and inspiration of the algorithm demand an interesting feature related to the temperature variation to be embedded in the operational characteristics of the algorithm.
This necessitates a gradual reduction of the temperature as the simulation proceeds. The algorithm starts initially with $T$ set to a high value (or infinity), and then it is decreased at each step following some *annealing schedule*—which may be specified by the user, but must end with $T = 0$ towards the end of the allotted time budget. In this way, the system is expected to wander initially towards a broad region of the search space containing good solutions, ignoring small features of the energy function; then drift towards low-energy regions that become narrower and narrower; and finally move downhill according to the steepest descent heuristic [20].

### 2.4.5 Selecting the parameters

In order to apply the SA method to a specific problem, one must specify the following parameters: the state space, the energy (goal) function $E(\cdot)$, the candidate generator procedure $\text{neighbour}(\cdot)$, the acceptance probability function $P(\cdot)$, and the annealing schedule temperature ($\cdot$) and initial temperature. These choices can have a significant impact on the method's effectiveness. Unfortunately, there are no choices of these parameters that will be good for all problems, and there is no general way to find the best choices for a given problem. The following sections give some general guidelines [20].

### 2.4.6 Diameter of the search graph

Simulated annealing may be modeled as a random walk on a *search graph*, whose vertices are all possible states, and whose edges are the candidate moves. An essential requirement for the neighbour () function is that it must provide a sufficiently short path on this graph from the initial state to any state which may be the global optimum. (In other words, the diameter of the search graph must be small.) For example in the traveling salesman for instance, the search space for $n = 20$ cities has $n! = 2,432,902,008,176,640,000$ (2.4 quintillion) states; yet the neighbour generator function that swaps two consecutive cities can get from any state (tour) to any other state in at most $n(n - 1) / 2 = 190$ steps [20].
2.4.7 Transition probabilities

For each edge \((s,s')\) of the search graph, one defines a transition probability, which is the probability that the SA algorithm will move to state \(s'\) when its current state is \(s\). This probability depends on the current temperature as specified by \(\text{temp}()\), by the order in which the candidate moves are generated by the \(\text{neighbour}()\) function, and by the acceptance probability function \(P()\). (Note that the transition probability is not simply \(P(e,e',T)\), because the candidates are tested serially.)

2.4.8 Efficient candidate generation

When choosing the candidate generator \(\text{neighbour}()\), one must consider that after a few iterations of the SA algorithm, the current state is expected to have much lower energy than a random state. Therefore, as a general rule, one should skew the generator towards candidate moves where the energy of the destination state \(s'\) is likely to be similar to that of the current state. This heuristic tends to exclude "very good" candidate moves as well as "very bad" ones; however, the latter are usually much more common than the former, so the heuristic is generally quite effective.

In the traveling salesman problem above, for example, swapping two consecutive cities in a low-energy tour is expected to have a modest effect on its energy (length); whereas swapping two arbitrary cities is far more likely to increase its length than to decrease it. Thus, the consecutive-swap neighbour generator is expected to perform better than the arbitrary-swap one, even though the latter could provide a somewhat shorter path to the optimum (with \(n - 1\) swaps, instead of \(n(n - 1)/2\)).

A more precise statement of the heuristic is that one should try first candidate states \(s'\) for which \(P(E(s),E(s'),T)\) is large. For the "standard" acceptance function \(P\) above, it means that \(E(s') - E(s)\) is on the order of \(T\) or less. Thus, in the traveling salesman example above, one could use a neighbour () function that swaps two random cities, where the probability of choosing a city pair vanishes as their distance increases beyond \(T\) [20].
2.4.9 Barrier Avoidance

When choosing the candidate generator neighbour () one must also try to reduce the number of "deep" local minima — states (or sets of connected states) that have much lower energy than all its neighbouring states. Such "closed catchment basins" of the energy function may trap the SA algorithm with high probability (roughly proportional to the number of states in the basin) and for a very long time (roughly exponential on the energy difference between the surrounding states and the bottom of the basin).

As a rule, it is impossible to design a candidate generator that will satisfy this goal and also prioritize candidates with similar energy. On the other hand, one can often vastly improve the efficiency of SA by relatively simple changes to the generator. In the traveling salesman problem, for instance, it is not hard to exhibit two tours A, B, with nearly equal lengths, such that (0) A is optimal, (1) every sequence of city-pair swaps that converts A to B goes through tours that are much longer than both, and (2) A can be transformed into B by flipping (reversing the order of) a set of consecutive cities. In this example, A and B lie in different "deep basins" if the generator performs only random pair-swaps; but they will be in the same basin if the generator performs random segment-flips [20].

2.4.10 Cooling schedule

The physical analogy that is used to justify SA assumes that the cooling rate is low enough for the probability distribution of the current state to be near thermodynamic equilibrium at all times. Unfortunately, the relaxation time—the time one must wait for the equilibrium to be restored after a change in temperature—strongly depends on the "topography" of the energy function and on the current temperature. In the SA algorithm, the relaxation time also depends on the candidate generator, in a very complicated way. Note that all these parameters are usually provided as black box functions to the SA algorithm.
Therefore, in practice the ideal cooling rate cannot be determined beforehand, and should be empirically adjusted for each problem. The variant of SA known as thermodynamic simulated annealing tries to avoid this problem by dispensing with the cooling schedule, and instead automatically adjusting the temperature at each step based on the energy difference between the two states, according to the laws of thermodynamics [20].

2.4.11 Restarts

Sometimes it is better to move back to a solution that was significantly better rather than always moving from the current state. This process is called **restarting** of simulated annealing. To do this we set \( s \) and \( e \) to \( s_{best} \) and \( e_{best} \) and perhaps restart the annealing schedule. The decision to restart could be based on several criteria. Notable among these include restarting based a fixed number of steps, based on whether the current energy being too high from the best energy obtained so far, restarting randomly etc [20].

2.4.12 Pseudocode of Simulated Annealing Algorithm

The Pseudocode of Simulated Annealing Algorithm is given as below [21]:

1: Generate initial solution \( x^c \), initialize \( R_{max} \) and \( T \)
2: for \( r = 1 \) to \( R_{max} \) do
3: while stopping criteria not met do
4: Compute \( x^n \in N (x^c) \) (neighbour to current solution)
5: Compute \( \Delta = f(x^n) - f(x^c) \) and generate \( u \) (uniform random variable)
6: if \( (\Delta < 0) \) or \( (e^{-\Delta/T} > u) \) then \( x^c = x^n \)
7: end while
8: Reduce \( T \)
9: end for
2.4.13 Flowchart of Simulated Annealing Algorithm

![Flowchart of Simulated Annealing Algorithm]

Figure 2.5 Flowchart of Simulated Annealing Algorithm
2.5 Threshold Acceptance Algorithm

Threshold acceptance (TA) is a local search method and was first described by Dueck & Scheuer and Moscato & Fontanari. Threshold acceptance uses a similar approach, but instead of accepting new points that raise the objective with a certain probability, it accepts all new points below a fixed threshold. The threshold is then systematically lowered, just as the temperature is lowered in an annealing schedule. Because threshold acceptance avoids the probabilistic acceptance calculations of simulated annealing, it may locate an optimizer faster than simulated annealing [22].

2.5.1 Neighbourhood and Thresholds

To implement TA, three points need to be specified:

1. The objective function $f$: This function is generally given by the problem at hand.

2. The neighborhood definition (the function $N$): Given a candidate solution $x^c$, one needs to define how to move from this solution to an alternative, but ‘close’ solution $x^n$.

3. The thresholds: Given a neigbourhood definition, one needs to determine the magnitude of the deterioration in the objective function that the algorithm should still accept for a new solution [22].
2.5.2 Pseudocode of Threshold Acceptance Algorithm

The Pseudocode of Threshold Acceptance Algorithm is given as below [22]:

1: Initialize nRounds and nSteps
2: Compute threshold sequence $T_r$
3: Randomly generate current solution $x^c \in X$
4: for $r = 1$: nRounds do
5: for $i = 1$: nSteps do
6: Generate $x^n \in N(x^c)$ and compute $\Delta = f(x^n) - f(x^c)$
7: if $\Delta < T_r$ then $x^c = x^n$
8: end for
9: end for
10: $x^\text{sol} = x^c$
2.5.3 Flowchart of Threshold Acceptance Algorithm

![Flowchart of Threshold Acceptance Algorithm](image)

Figure 2.6 Flowchart of Threshold Acceptance Algorithm
2.6 Objectives

Motivated by above discussion, we set three main objectives for this work.

1. To do comparative analysis between simulated annealing algorithm, threshold acceptance algorithm and genetic algorithm by considering nine different test functions.

2. To design a Stock Ranking Model for financial application.

3. To design a financial application that gives the optimized Stock Portfolio Selection using Knapsack Problem by genetic algorithm.

2.7 Methodology

In this work,

First, Genetic Algorithm (GA) is compared with two different optimization algorithms i.e. Simulated Annealing Algorithm (SA) and Threshold Acceptance Algorithm (TA). Results show that genetic algorithm is far better compared to simulated annealing algorithm and threshold acceptance algorithm in all the cases for all nine test function optimization problems.

Secondly after finding that Genetic Algorithm is far better among these three algorithms in all the cases for all nine test function optimization problems, finally genetic algorithm is applied on financial application of Stock Portfolio Selection Problem which finds out the optimized stock portfolio.