Chapter 1

Introduction

In daily life, so many examples are observed where the aim is to maximize or minimize a certain function of one or more parameters. From a shopkeeper to multi-billionaire gigantic corporations, the goal is to maximize profits. Engineers working in the aviation and car industries try to minimize air drag, and electronic engineers try to minimize the size of basic electronic components. Medical surgeons are always working on new techniques to maximize their patients' life span. When a task is performed to maximize or minimize a certain objective, it is known as *optimizing* that objective. For example, there may be a significantly large number of ways to organize a factory production schedule. The problem is to determine which schedule gives the best throughput, and thus which schedule is *optimal*.

Optimization is a significant topic in a variety of areas, including engineering, science, medicine, and business. In many of these disciplines, optimization simply means "doing better". In the context of this thesis, however, optimization refers to the process of finding the best possible solution to an optimization problem within a given time limit. This thesis considers a specific optimization problem, namely

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

network topology design. A number of techniques and their variants are investigated for their applicability to a specific case of the above-mentioned problem, namely *topology design of distributed local area networks* (or DLAN topology design). The focus is specifically on the following optimization techniques: simulated evolution (SimE) [148], stochastic evolution (StocE) [216], simulated annealing (SA) [186], ant colony optimization (ACO) [42], and particle swarm optimization (PSO) [140].

1.1 Motivation

A variety of difficult network topology design problems are categorized according to the objective(s) to be optimized. Presence of constraints further amplifies the complexity of these problems. The problems have received significant attention in order to find efficient approaches to solve them [57, 58, 75, 80, 98, 111, 142, 151, 152, 208]. However, many of these approaches have not proven to be fully able to address the problem under consideration [80, 142, 151, 208].

Local search techniques have been frequently used to optimize network topology design problems [80, 142, 151, 208, 267]. However, these techniques generally do not perform well enough when multiple objectives need to be optimized and/or constraints are present [142, 268, 271, 272]. The specific DLAN topology design problem considered in this thesis is a multi-objective combinatorial optimization problem which tends to have a solution space that grows exponentially with the problem size. There are somewhat simpler versions of the DLAN topology design problems which are NP-hard [75, 79, 98], and hence the DLAN topology design problem can be classified as an NP-hard problem. The problem has a number

of objectives that need to be optimized simultaneously, in presence of constraints. Hence, iterative heuristics, such as evolutionary algorithms or swarm intelligence techniques, seem to be appropriate approaches to solve the problem. Iterative heuristics have a tendency to escape a local optimum and can often find a global optimum solution in a reasonable amount of computational time. The iterative heuristics mentioned above have proven to be successful for a number of NP-hard problems [19, 22, 76, 95, 143, 153, 179, 197, 211, 214, 271, 273], hence providing motivation to apply these algorithms to topology design of distributed local area networks. One of these algorithms, namely SA, has been used by researchers for optimization problems for more than thirty years, yet research is still going on to further improve its search capabilities. Others, such as SimE and StocE, are relatively new, and have not been exploited by researchers.

One important feature of SA, SimE, and StocE is that they operate on a single solution, as compared to genetic algorithms, which maintain and operate on a population of solutions. Furthermore, genetic algorithms perform complex operations such as crossover and mutation. The convergence time for SA, SimE, and StocE is much less than that of genetic algorithms [220]. All these aspects of these "single-solution" heuristics make them a strong candidate for application to topology design of distributed local area networks. In addition, research has revealed that hybridization of heuristics with each other has generally proven to be more efficient and effective [144, 194, 234, 249, 270, 271]. This particular aspect provides the motivation to develop hybrid heuristics for the DLAN topology design problem. SA, SimE, and StocE have a number of parameters to be initialized by the user. Since the best values for these parameters are problem-dependent, trial runs of the

heuristics are required to find appropriate values for these parameters, which is a time-consuming process. Thus, the motivation arises to propose ways such that user intervention in finding the appropriate values for these parameters is reduced or eliminated.

Swarm intelligence (SI) techniques, which include ant colony optimization and particle swarm optimization, are also new in the field of optimization methods, and are currently being analyzed by researchers through extensive application of these techniques to a variety of NP-hard problems to find out their capabilities and limitations [37, 45, 52, 65, 78, 81, 150, 171, 201, 225, 227, 251]. The success of these SI algorithms provides motivation for a study on their applicability to the DLAN topology design problem.

A great deal of research has been dedicated to address issues related to multi-objective optimization [31, 51, 125, 206, 209, 275]. Among many other approaches [29, 39, 50, 96, 97, 112, 122, 130, 173], fuzzy logic [276] has been used to solve multi-objective optimization problems. Therefore, the motivation also arises to utilize fuzzy logic in the above algorithms to address the multi-objective nature of the DLAN topology design problem.

1.2 Objectives

The primary objectives of this thesis are summarized as follows:

1. To address the multi-objective nature of the DLAN topology design problem by using fuzzy logic.

2. To show that the stochastic evolution, simulated evolution, simulated anneal-

ing, ant colony optimization, and particle swarm optimization can be successfully used to solve the DLAN topology design problem.

3. To propose a multi-objective simulated evolution algorithm and its hybrid variant for the DLAN topology design problem and to analyze the performance of the algorithm.

4. To propose a multi-objective simulated annealing algorithm and its hybrid variants for the DLAN topology design problem and to analyze the performance of the algorithm.

5. To propose a multi-objective stochastic evolution algorithm and its hybrid variant for the DLAN topology design problem and to analyze the performance of the algorithm.

6. To propose approaches which can reduce user intervention in setting the parameters in simulated annealing, simulated evolution, and stochastic evolution.

7. To propose a multi-objective ant colony optimization algorithm for the DLAN topology design problem and to analyze the performance of the algorithm.

8. To propose a multi-objective particle swarm optimization algorithm for the DLAN topology design problem and to provide a preliminary analysis of the performance of the algorithm.

9. To compare the relative performance of each of the above algorithms and to find out which algorithm(s) perform the best.

1.3 Methodology

The algorithms proposed in this thesis are first presented and discussed. Since the DLAN topology design problem is a very specific case of network topology design, no well-known benchmark cases exist. Therefore, test cases given in [271, 268, 270, 272, 269] are used to quantify performance of the proposed algorithms.

For each of the algorithms, values of control parameters are optimized to produce best performances.

The performances of the different variants of simulated annealing, simulated evolution, and stochastic evolution are empirically compared. The three algorithms are also compared with each other.

For ant colony optimization and particle swarm optimization, empirical results for the two algorithms are compared with each other.

Due to the stochastic nature of the proposed algorithms, results are generally reported in terms of averages and standard deviations over several simulations. However, since the simulations are computationally expensive, averages are calculated for thirty runs, and the average run time for the thirty runs is reported wherever appropriate. However, the performance of an algorithm is evaluated based on the quality of solutions produced. The results are also statistically validated through t-tests.

1.4 Contributions

The main contributions of this thesis are:

1. An approach based on fuzzy logic is proposed to deal with the multi-objective

nature of the DLAN topology design problem. The proposed approach employs fuzzy logic to combine multiple objectives into a single objective function.

2. A new fuzzy operator – the unified And-Or (UAO) operator – is developed, together with both a theoretical and empirical study of its characteristics. The purpose of this operator is to aggregate the multiple objectives into a single objective function. The UAO operator is compared with the well-known ordered weighted average operator [259, 261]. The comparison is done by applying the two operators to all the proposed algorithms.

3. The following algorithms are developed to solve the multi-objective DLAN topology design problem, and these algorithms are analyzed using the proposed fuzzy objective function:

   (a) stochastic evolution,

   (b) simulated evolution,

   (c) simulated annealing,

   (d) ant colony optimization, and

   (e) particle swarm optimization.

4. Hybrid algorithms for the DLAN topology design problem using the fuzzy objective function are developed and analyzed. More specifically:

   (a) A hybrid version of SimE that incorporates tabu search characteristics into the algorithm is developed and analyzed.

   (b) A hybrid variant of StocE that incorporates tabu search characteristics into the algorithm is developed and analyzed.

   (c) Two hybrid variants of SA are developed and analyzed. The first variant incorporates characteristics of tabu search in the SA algorithm, while the second incorporates tabu search and SimE characteristics in the SA algorithm.

5. An approach to dynamically determine the control parameters in simulated evolution, stochastic evolution, and simulated annealing is developed and empirically evaluated. More specifically,

   (a) a dynamic bias B in SimE is proposed and evaluated,

   (b) a dynamic factor R in StocE is proposed and evaluated, and

   (c) a dynamic length of the Markov chain M in SA is proposed and evaluated.

1.5 Organization of Thesis

Chapter 2 provides a general overview of optimization methods. The chapter starts with a short discussion on optimization. This is followed by an elaborate discussion on multi-objective optimization. Another focus of this chapter is the background on fuzzy logic, with respect to its use in multi-objective optimization and well-known operators. This is followed by a discussion on some iterative optimization algorithms, which are the focus of this thesis. In this context, detailed discussions on simulated evolution, stochastic evolution, simulated annealing, ant colony optimization, and particle swarm optimization are provided. Genetic algorithms and tabu search are briefly discussed.

Chapter 3 reviews the DLAN topology design problem addressed in this thesis

in sufficient detail. This includes a formal description of the problem, notation, assumptions, terminology, cost functions, and computation of objective values.

Chapter 4 discusses the proposed unified And-Or (UAO) operator. This includes the definition and mathematical representation of the operator, and its mathematical properties. The chapter also discusses the application of the UAO operator to the DLAN topology design problem as well as the use of preferences of objectives in the context of multi-objective optimization.

Chapter 5 provides details on the implementation of the multi-objective fuzzy StocE algorithm for the DLAN topology design and how the algorithm has been modified to incorporate tabu search characteristics. The fuzzy StocE and its tabu search based variant are mutually compared through empirical results. A dynamic value of R is proposed, in order to eliminate the user-defined value of the parameter, and empirical results are provided and discussed.

Chapter 6 describes the proposed multi-objective fuzzy SimE algorithm for the DLAN topology design. The chapter focusses on the basic SimE algorithm for DLAN topology design, and also on its hybrid variant resulting from incorporating tabu search characteristics. The chapter discusses how to reduce user intervention to control the value of the bias, B. Empirical results are provided and discussed.

Chapter 7 presents a DLAN topology design approach that is based on SA. The implementation details of this fuzzy multi-objective SA algorithm, as well as its two hybrid variants, are discussed. The two variants incorporate tabu search and simulated evolution characteristics into the SA algorithm. Furthermore, an approach is proposed to dynamically determine the value of Markov chain, M, where the approach reduces user intervention in setting up an appropriate value for this

parameter. The proposed SA algorithms are empirically compared.

Chapter 8 presents the proposed multi-objective fuzzy ACO algorithm for the DLAN topology design. The details on the implementation are provided. The algorithm is empirically analyzed.

Chapter 9 discusses the proposed multi-objective fuzzy PSO algorithm for the DLAN topology design. The details on the implementation are provided, and the algorithm is empirically analyzed.

Chapter 10 summarizes and compares the results obtained in Chapters 6 to 9. The focus of this chapter is to determine which of the proposed algorithms performs the best.

Chapter 11 highlights the conclusions of this thesis and provides directions for future research.

The appendices provide a list of symbols used in this thesis and a list of publications derived from the work discussed in this thesis.

Chapter 2

Optimization and Optimization Approaches

This chapter provides a brief overview of optimization. The chapter covers both single-objective and multi-objective optimization, with emphasis on the latter. Another focus of this chapter is the background of fuzzy logic, with respect to its use in multi-objective optimization and some well-known operators. This is followed by a discussion on iterative optimization algorithms. In this context, detailed discussions are given regarding the fundamentals of simulated evolution, stochastic evolution, simulated annealing, ant colony optimization, and particle swarm optimization. Genetic algorithms and tabu search are also discussed briefly.

2.1 Optimization

In its simplest definition, optimization is the process of trying to find the best possible solution to an *optimization problem* within a given amount of time [44].

The objective of optimization is to determine the values of a set of parameters such that the objective function is maximized or minimized, subject to certain constraints [244]. A *feasible solution* is defined as a solution that satisfies all design constraints. A feasible solution results from an assignment of values to design parameters. An *optimal solution* is defined as a feasible solution that results in the optimum value of the objective function(s) among the set of other feasible solutions. In other words, if a pool of feasible solutions exists, then the optimum solution is the one which produces the optimum values of the objective functions (either minimum or maximum, depending on the nature of the problem). Associated with the optimum solution are the *optimum values of parameters*.

Optimization techniques are constantly employed in many disciplines, since many real-world problems are optimization problems. Optimization techniques have been applied in industry, business, engineering, science, and medicine, with applications such as planning, resource allocation, timetabling, decision making, and structural design.

Optimization problems can be classified as *unconstrained* or *constrained* problems. In unconstrained problems, the aim is to minimize or maximize the function without any conditions or constraints imposed on the values of design parameters. Thus, all values of the variables within the domain of the function are considered in searching for the optimum value. Since this thesis deals with a maximization problem, the terms optimization and maximization will be used interchangeably. An unconstrained maximization problem is formally defined as follows [189]:

$$\text{Given } f: \Re^n \to \Re$$
$$\text{find } \mathbf{x}^* \in \Re \; for \; which \; f(\mathbf{x}^*) \geq f(\mathbf{x}) \; \forall \; \mathbf{x} \in \Re^n \tag{2.1}$$

In Equation (2.1), vector $\mathbf{x}^*$ is called the *global maximizer* while $f(\mathbf{x}^*)$ is called the *global maximum* value of f. The process of seeking a global maximum is called *global optimization* [107]. In contrast to global optimization, the term *local optimization* refers to an optimum value within the *local neighborhood* of a solution. In other words, a global optimum is the maximum value within the complete search space, while the local optimum is the maximum value within a sub-region, $B \subseteq S$, of the search space. Thus, for multi-modal problems, it can be inferred that there exist many local optima within the global search space (this is not necessary for unimodal problems, which have only one optimum). It should also be noted that every global optimum is also a local optimum, but a local optimum is not necessarily a global optimum, as illustrated in Figure 2.1. In this figure, $\mathbf{x}^*$ is the global optimum, while $\mathbf{x_b}$ is a local optimum.

In contrast to unconstrained problems, the goal in constrained optimization is to optimize the objective function subject to certain constraints. These constraints often make certain points in the search space invalid. These points might otherwise be global optima. Formally, a constrained maximization problem is defined as follows:

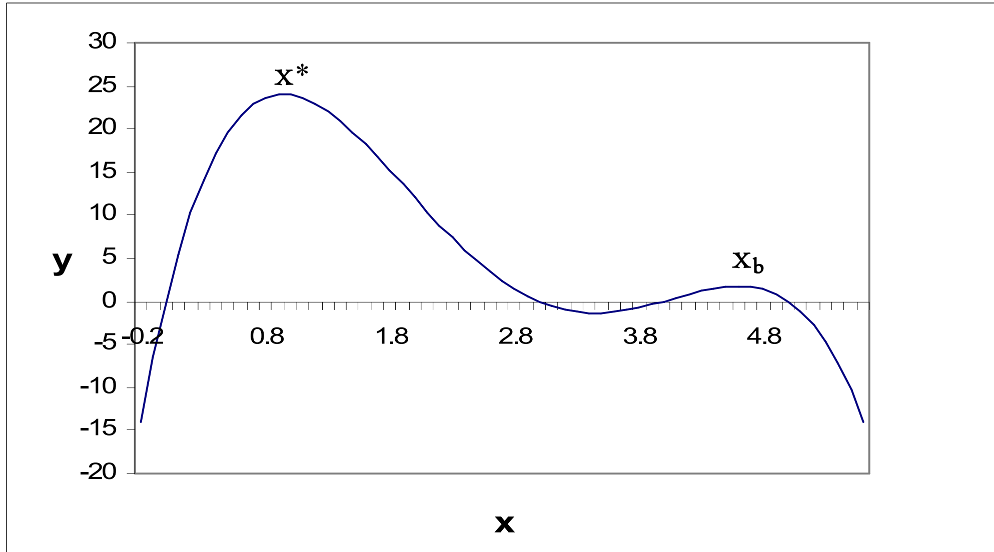

Figure 2.1: Example of global maximum $\mathbf{x}^*$ and local maximum $\mathbf{x_b}$

$$\text{Given } f : \Re^n \rightarrow \Re$$

$$\text{find } \mathbf{x}^* \in \Re \; for \; which \; f(\mathbf{x}^*) \geq f(\mathbf{x}) \; \forall \; \mathbf{x} \in S \subseteq \Re^n \tag{2.2}$$

$$\text{subject to } \; \mathrm{g}_m(\mathbf{x}) \leq 0, \quad m = 1, \cdots, n$$

$$\mathrm{h}_m(\mathbf{x}){=}0, \quad m = n + 1, \cdots, n + n_h$$

Optimization problems can be further categorized based on the number of objectives to be solved. Many problems, whether constrained or unconstrained, require only one objective to be optimized. Solving this type of optimization problem is referred to as *single objective optimization* (SOO). For example, in an organization,

say ABC, the objective could be to maximize productivity. The definition of SOO can be further extended to many objectives which are "non-conflicting". That is, if there are a number of objectives, and if the aim is to maximize all of them, and if maximizing one objective automatically maximizes others, then it will also be a case of SOO. For example, in organization ABC, if one objective is to maximize the profitability, then maximizing productivity will implicitly maximize profitability at the same time, since the two objectives are directly proportional to each other. However, problems arise when the optimization objectives are "conflicting". That is, optimizing one objective could result in degradation of the other objectives. For example, if productivity is to be maximized, while the workforce is to be minimized, then the two objectives are conflicting, since variation in one objective will adversely affect the other. This is where *multi-objective optimization* (MOO) comes into the picture. In this type of situation, a trade-off is needed to obtain a "balanced" solution; a solution (or rather a set of solutions, referred to as the *Pareto front*) that has the best possible value of all objectives. Single objective optimization is useful when insights into the nature of the problem are sought by decision-makers [223]. However, SOO is generally not capable of providing a set of comparable solutions that trade different objectives against each other. This capability is provided by MOO [223]. Since the focus of this thesis is on constrained MOO, a detailed discussion on the subject is given below.

2.2 Constrained Multi-objective Optimization

In multi-objective decision-making problems, a possible compromise between several conflicting objectives needs to be found by evaluating these objectives [173]. Many multi-objective optimization problems are also constrained. Therefore, any optimization technique applied to solve these problems must ensure that all constraints are satisfied by the set of optimum solutions [257]. Multi-objective optimization problems are usually solved by scalarization (also referred to as weighted aggregation); the problem is converted into a single or a family of single objective problems, which can then be solved using single objective optimizers. Mathematically, a MOO problem can be stated as follows:

$$\text{Optimize}: \quad \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \cdots, f_K(\mathbf{x})) \tag{2.3}$$

$$\text{subject to} \quad \mathrm{g}_m(\mathbf{x}) \leq 0, \quad m = 1, \cdots, n$$

$$\mathrm{h}_m(\mathbf{x}) = 0, \quad m = n+1, \cdots, n+n_h$$

where $\mathbf{x} \in S$, and there are at least two (i.e. $K \geq 2$) conflicting objective functions, f_k, that need to be optimized simultaneously. Here $\mathbf{x} = (x_1, x_2, ..., x_D)$ is called the vector of decision variables, S is defined as the feasible region, $\{g_m(\mathbf{x})\}$ is the set of inequality constraints, and $\{h_m(\mathbf{x})\}$ is the set of equality constraints. Due to the contradiction of objectives, there does not exist a single solution that would optimize all the objectives simultaneously. In multi-objective optimization, vectors are regarded as optimal if their components cannot be improved without

deterioration of any one of the other components [173]. This is usually referred to as *Pareto optimality*. Presence of multiple objectives in an optimization problem usually gives rise to a set of optimal solutions, commonly known as Pareto-optimal solutions. Pareto-optimal solutions are also referred to in the literature as non-dominated, non-inferior, or Pareto-efficient. A non Pareto-optimal solution is a solution where one optimization criterion can be improved without degrading any others. This solution is known as a dominated or inferior solution. Mathematically, the MOO problem is considered to be solved when a Pareto-optimal set is found. This is also known as vector optimization. MOO also has a relationship with multi-modal optimization. The slight difference between the two is that MOO generally results in Pareto-optimal solutions, which are global optimal solutions. In multi-modal optimization, the set of solutions includes multiple optimum solutions, but many of these solutions are local optimal solutions. Niching methods have also been adopted by researchers to maintain diversity in the population of solutions as well as to allow a population-based iterative algorithm to find many optima in parallel [231].

A number of approaches to handle constraints have been reviewed by Fonseca *et al.* [87]. Among them, two common approaches to handle constraints have been utilized for use with Pareto-based ranking methods. The first employs penalization of the rank of infeasible individuals, while the second considers transformation of constraints to objectives [257].

In the design or planning stages of an optimization problem, the consideration of many objectives provides three major improvements to the procedure that directly supports the decision-making process [41]:

1. A multi-objective methodology usually identifies a wider range of alternatives.

2. In planning and decision-making processes, the roles of "analyst" or "modeler" and "decision-maker" are more appropriately promoted by considering multiple objectives. An analyst or modeler generates alternative solutions, and a decision-maker uses the solutions generated by the analyst to make informed decisions.

3. More realistic models of a problem are elaborated if many objectives are considered.

Several methods for handling the multi-objective aspects by finding a Pareto set of solutions have been reported in the literature. Some of the popular methods are summarized below.

2.2.1 Weighted Sum Method

The weighted sum method [96, 275] is one of the simplest MOO approaches. This method was extended to address constrained optimization problems [7, 232], where the aim is to solve the following problem:

$$\text{Maximize} \sum_{i=1}^{K} w_i f_i(\mathbf{x}) \tag{2.4}$$

$$\text{subject to } \ \mathbf{x} \in S$$

$$g_m(\mathbf{x}) \leq 0, \quad m = 1, \cdots, n$$

$$\text{h}_m(\mathbf{x})=0, \quad m = n+1, \cdots, n+n_h$$

where $w_i \geq 0$ for all $i = 1, ..., K$, and $\sum_{i=1}^{K} w_i = 1$. The solution to the above equation is weakly Pareto optimal. Weak Pareto optimal solutions are points where all criteria cannot be simultaneously improved. It is Pareto optimal if $w_i > 0$ for all $i = 1, ..., K$ or if the solution is unique [173]. The values of w_i are generally set by the user and therefore require proper adjustment to obtain the desired results. The purpose of these weights is to define relative importance of the individual objectives during the optimization process. A larger weight assigned to one objective as compared to the others would guide the search into a region where this particular objective achieves relatively better optimization than the other objectives.

Advantages and Disadvantages

One drawback of the weighting method is that not all the Pareto optimal solutions can be found unless the problem is convex. An algorithm for generating different weights automatically for convex problems to produce an approximation of the Pareto optimal set is proposed in [26]. The weighted sum method has several other weaknesses. For example, a small change in weights may result in big changes in the objective vectors $f_1(\mathbf{x})$, $f_2(\mathbf{x})$,$\cdots$, $f_K(\mathbf{x})$. Moreover, significantly different weights may produce nearly similar objective vectors. The main advantage of the method is its computational efficiency, which makes the method a strong candidate for generating a strongly non-dominated solution that can be used as an initial solution for other techniques [39].

Some Applications

The weighting sum method has been used quite extensively, with new improvements to the method developed continuously. One such application is by Jakob *et al.* [125] who used a weighted sum of several objectives involved in a task planning problem. Another example is by Jones *et al.* [129], where the authors used weights for their genetic operators in order to reflect the effectiveness of these operators when a GA was applied to generate hyperstructures from a set of chemical structures.

2.2.2 ε-Constraint Method

The ε-Constraint method [112] is based on the minimization of one objective function (the most preferred or primary, as chosen by the decision-maker), and considering the other objectives as constraints bound by some allowable levels. Hence, a single-objective minimization is carried out for the most relevant objective function subject to additional constraints on the other objective functions. The upper bounds of these constraints are obtained through the ε-vector, and by varying the ε-vector, the exact Pareto front can theoretically be generated. The method optimizes one of the objective functions in the form (considering a minimization problem):

$$
\begin{aligned}
\text{Minimize} \quad & f_l(\mathbf{x}) \\
\text{subject to} \quad & f_j(\mathbf{x}) \leq \varepsilon_j \quad \text{for all} \quad j = 1, ..., K, \quad j \neq l \\
& \mathbf{x} \in S
\end{aligned}
\tag{2.5}
$$

where $l \in \{1, ..., K\}$, and ε_j are upper bounds for the objectives $f_j \neq f_l$. The upper bounds are user defined. The solution to Equation (2.5) is weakly Pareto

optimal since the main objective is optimized while satisfying other objectives within a certain bound. However, $\mathbf{x}^* \in S$ is Pareto optimal if and only if Equation (2.5) is solved for every $l = 1, ...K$, where $\varepsilon_j = f_j(\mathbf{x}^*)$ for $j = 1, ...K$, and $j \neq l$ [173]. Thus, it is not a necessary condition that the problem be convex in order to find any Pareto optimal solution. To ensure Pareto optimality in this method, either K different problems have to be solved, or a unique solution has to be obtained. However, it is generally not easy to verify uniqueness [173].

Advantages and Disadvantages

The most prominent disadvantage of the above approach is that it is time-consuming [39]. Also, coding of the objective functions may be difficult or even impossible for certain problems, particularly if there are many objectives [39]. Furthermore, the method may not be appropriate in some applications, since the method tends to find weakly non-dominated solutions [39]. However, the main strength of the technique is its simplicity, making it attractive to optimization practitioners [39].

Some Applications

The ε-Constraint method has been used in many applications. Quagliarella *et al.* [209] used this technique in combination with a hybrid GA to solve multi-objective optimization problems. Loughlin *et al.* [160] applied ε-Constraint method to a real-world air quality management problem having two conflicting objectives: to maximize the amount of emissions reduction and to minimize the cost of controlling air pollutant emissions.

2.2.3 Lexicographic Ordering

Lexicographic ordering [39] ranks the objectives in order of importance. The domain expert (modeler) assigns the importance to objectives. The optimum solution, $\mathbf{x}^*$, is then obtained by optimizing the objective functions. The most important objective is optimized first, after which the other objectives are optimized according to the assigned order of their importance.

Let the subscripts of the objectives denote the objective function number as well as the priority of the objective. Therefore, $f_1(\mathbf{x})$ and $f_K(\mathbf{x})$ represent the most and least important objective functions, respectively. Then, the first problem is formulated as [39],

$$\begin{aligned} \text{Maximize} \quad & f_1(\mathbf{x}) \\ \text{subject to} \quad & g_j(\mathbf{x}) \leq 0 \qquad j = 1, 2, ..., m \\ & \mathrm{h}_j(\mathbf{x}) \ = 0 \qquad j = m+1, \cdots, m + m_h \end{aligned} \tag{2.6}$$

The solution of Equation (2.6) is referred to as $\mathbf{x_1^*}$. The second problem is then formulated as:

$$\begin{aligned} \text{Maximize} \quad & f_2(\mathbf{x}) \\ \text{subject to} \quad & g_j(\mathbf{x}) \leq 0 \qquad j = 1, 2, ..., m \\ & \mathrm{h}_j(\mathbf{x}) \ = 0 \qquad j = m+1, \cdots, m + m_h \\ \textit{and} \quad & \mathrm{f}_1(\mathbf{x}) = \mathbf{x}^* \end{aligned} \tag{2.7}$$

The solution of Equation (2.7) is referred to as $\mathbf{x_2^*}$. This procedure is then continued

until solutions to all K objectives are found. The solution, $\mathbf{x}^*_K$, obtained at the end is the desired solution, $\mathbf{x}^*$.

Advantages and Disadvantages

The main disadvantage of lexicographic ordering is that it tends to favor certain objectives when many are present, since randomness is involved in the process. This in turn may cause the solutions to be concentrated in a particular part of the Pareto front rather than the complete front [36]. This particular situation is undesirable, since a diverse spread of solutions in the Pareto front is required. Moreover, the decision-maker may find it difficult to specify an absolute order of importance [173]. However, an advantage of the technique is its simplicity which makes it competitive with the weighted sum approach [39].

Some Applications

Prasad and Kuo [206] used lexicographic ordering to solve redundancy allocation problems in coherent systems. Shou and Guo [229] used the method in engineering management, where lexicographic ordering is used to select research and development projects subject to resource constraints. Another application of the process has been reported by Jégou *et al.* [124], where the method has been used in developing data compression techniques.

2.2.4 Goal Programming

Goal programming [29, 122] is one of the first methods exclusively developed for multi-objective optimization [173]. The technique was developed for linear models

and has an effective role in industrial optimization problems [39]. Targets, or goals that need to be achieved, are assigned by the decision-maker. Values associated with these goals are then incorporated into the optimization problem as additional constraints [39]. The decision-maker specifies aspiration levels (i.e. the ideal values of the objectives), T_i $(i = 1, ..., K)$, for the objective functions and absolute deviations from these aspiration levels are minimized to the best possible extent [173]. For a maximization problem, goals are of the form $f_i(\mathbf{x}) \geq T_i$. The simplest form of goal programming is formulated as [71],

$$\text{Maximize} \quad \sum_{i=1}^{K} |f_i(\mathbf{x}) - T_i| \tag{2.8}$$

$$\text{subject to} \quad \mathbf{x} \in S$$

$$g_m(\mathbf{x}) \leq 0, \quad m = 1, \cdots, n$$

$$\mathrm{h}_m(\mathbf{x})=0, \quad m = n+1, \cdots, n + n_h$$

The objective is to minimize the sum of the absolute values of the differences between target values and actually achieved values [39]. A more generalized form of the goal programming objective function is a weighted sum of the pth power of the deviation $|f_i(\mathbf{x}) - T_i|$[53], referred to as generalized goal programming [54, 71].

Advantages and Disadvantages

The main advantage of goal programming is its computational efficiency, provided that the target values are known, and if the goals are in a feasible region [39]. Goal programming would generally produce a dominated solution if the target point is chosen in a feasible region [71]. However, if the targets are wrong, then a feasible

region is difficult to approach, in which case goal programming could be very inefficient. Nevertheless, goal programming may prove useful in situations where a linear or piecewise-linear approximation of the objective functions can be made, because of the availability of excellent computer programs for such approximations, along with the possibility of eliminating dominated goal points easily [39]. However, for non-linear cases, this approach may not be a viable option and other approaches may be more efficient.

Some Applications

Goal programming has been used in many applications. Some recent applications include the works of Li *et al.* [156], Johnson *et al.* [205], and Dawande and Gupta [51]. Li *et al.* [156] used goal programming for proposing a generalized varying-domain optimization method with multiple priorities. Johnson *et al.* [205] applied goal programming for project time/cost tradeoff analysis and decision making, while considering quality issues. Dawande and Gupta [51] used goal programming to solve bi-criteria multicasting problems in optical networks.

2.2.5 Goal Attainment

The goal attainment method [97] involves expressing a set of design goals $T_1, T_2, ..., T_K$, associated with a set of objectives, $f_1, f_2, ..., f_K$. The problem is formulated in such a way that objectives are allowed to be over-achieved or under-achieved by using a vector of weights, $w = (w_1, w_2, ...w_K)$. This vector of weights is provided by the decision-maker, and enables him/her to be relatively imprecise about the initial goals. In order to find the best solution, $\mathbf{x}^*$, the following problem is solved [39]:

$$\text{Minimize} \quad \alpha_{goal}$$

$$\begin{aligned} \text{such that} \quad & T_i + \alpha_{goal}.w_i \geq f_i(\mathbf{x}) \quad i = 1, 2, ..., K, \quad \mathbf{x} \in S, \quad w \in \Lambda \\ \text{subject to} \quad & g_j(\mathbf{x}) \leq 0 \qquad j = 1, 2, ..., m \\ & \mathrm{h}_j(\mathbf{x}) = 0 \qquad j = m+1, \cdots, m+m_h \end{aligned} \tag{2.9}$$

where α_{goal} is a scalar variable unrestricted in sign and $\Lambda = \{w \in R^n \ s.t. \ w_i \geq \Im, \sum_{i=1}^{K} w_i = 1, \text{ and } \Im \geq 0\}$

The term $\alpha_{goal}.w_i$ introduces a degree of slackness into the problem, which would otherwise require that the goals be rigidly met. The weight vector w enables the decision-maker to quantitatively express the tradeoffs among the objectives. For smaller w_i, the i_{th} objective prefers a smaller function value.

Given the vectors T_i and w, the direction of the $T_i + \alpha_{goal}.w_i$ vector can be determined. Therefore, the problem in Equation (2.9) is equivalent to finding a feasible solution, which is nearest the origin, on this vector in objective space. During the optimization, α_{goal} is varied, changing the size of the feasible region.

It is worth mentioning that whether the goals are attainable or not depends on the value of α_{goal}. A negative value of α_{goal} implies that the goal of the decision-maker is attainable and an improved solution can be obtained. Otherwise, if $\alpha_{goal} > 0$, then the goal is unattainable [39].

Advantages and Disadvantages

The most prominent disadvantage of the goal attainment method is that the technique can generate misleading results in some cases [255]. For example, if there are

two candidate solutions having the same value in one objective function but different in the other, then the solutions could still have the same goal-attainment value for their two objectives. This means that none of the solutions will be better than the other [39]. The main advantages of the technique are its computational efficiency and simple implementation.

Some Applications

A number of applications of goal attainment have been reported in the literature. Mueller *et al.* [184] used the goal attainment technique for analog circuit design, considering circuit parameters such as transistor lengths and widths, high gain, and low power consumption. Chen and Huang [31] utilized the technique for bi-objective power dispatch optimization, considering fuel cost and environmental impact of multiple emissions. Liao *et al.* [157] adopted the technique for optimal multi-objective filter planning in industrial distribution systems.

2.2.6 Other Approaches

Several other approaches for handling multiple objectives have also been reported in the literature. Some of these approaches include the method of weighted metrics [173], the weighted min-max approach [130], normal boundary interaction [50], and the use of game theory [188].

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

2.3 Fuzzy Logic and Multi-objective Optimization

Apart from the techniques described earlier in this chapter, fuzzy logic is another technique that has been used for multi-objective optimization. In recent years, the use of fuzzy logic for MOO has gained some momentum, with applications in various areas, such as analog circuit design [191], war resource allocation [195], direct current electromagnet design [32], and facility location selection [133]. Since one of the focus areas of this thesis is on fuzzy logic, a detailed overview of the technique is given in this section.

The theory of fuzzy sets [276, 277] is based on a multi-valued logic wherein a statement can be partly true and partly false at the same time. In fuzzy logic, the degree of truthfulness of a statement is expressed by a *membership function*, μ, in the range [0,1]. A value of $\mu = 0$ indicates that the statement is false, while $\mu = 1$ indicates that the statement is true. The fuzzy logic approach differs from binary logic, in that binary logic allows a statement to be either false or true.

The fuzzy logic approach replaces the vector-based objective function with a scalar function [230]. Although it is possible to describe uncertainties in terms of conditional probabilities, it is difficult to do so for the majority of practical cases [230]. A framework for representing uncertainties is conveniently provided by fuzzy logic, thus giving a strong reason to consider a fuzzy logic approach to MOO problems.

Another reason to advocate the use of fuzzy logic in MOO is due to the nature of algorithms used for solving MOO problems. Many MOO problems are proven to

be NP-hard in nature. The situation becomes even more complex in the presence of design constraints. To solve these NP-hard problems, heuristics are employed, which are based on human knowledge acquired through experience and understanding of problems [230]. Natural language, which provides the foundation of fuzzy logic, has a more convenient approach for expressing such knowledge.

2.3.1 Fuzzy Set Theory

A crisp set, X, is normally defined as a collection of elements or objects, $x \in X$, that can be finite, countable or uncountable. Each single element can either belong to a set or not. However, in real-life situations, objects do not have crisp (1 or 0) membership to sets. Fuzzy set theory (FST) aims to represent vague information, such as 'low load', 'high load', or 'low latency', etc., which are difficult to represent in classical (crisp) set theory. A fuzzy set is characterized by a membership function which provides a measure of the degree of membership for every element to the fuzzy set [169, 278]. A fuzzy set, A, of a universe of discourse, X, is defined as $A = \{(x, \mu_A(x)) | \ \forall \ x \ \in \ X\}$, where $\mu_A(x)$ is a membership function of x with respect to fuzzy set A. Figure 2.2 shows an example of a membership function.

As for crisp sets, set operations such as union, intersection, and complement, are also defined on fuzzy sets. There are many operators for fuzzy union and fuzzy intersection. For fuzzy union, the operators are known as **s-norm** operators (denoted as $\oplus$). The s-norm operators are also known as "ORing" functions since they implement the OR operation between the membership functions under consideration. Some examples of **s-norm** operators are given below (where A and B are fuzzy sets of universe of discourse, X) [169]:

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

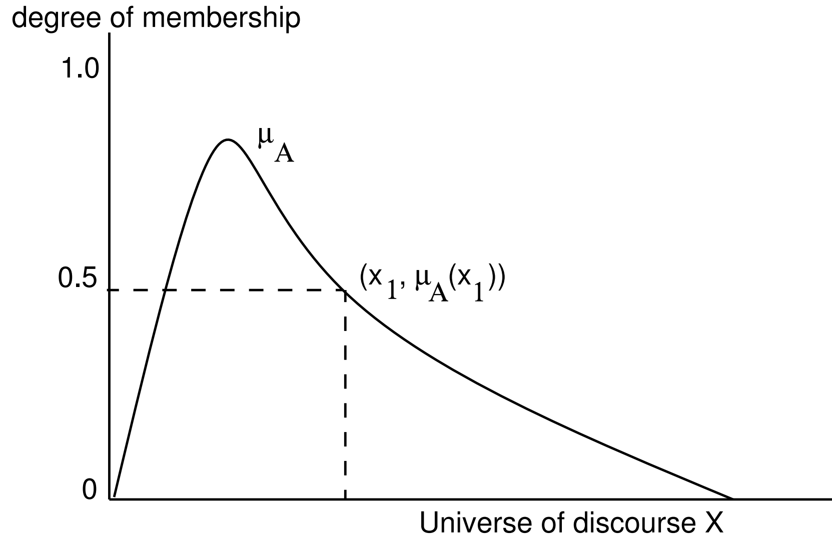

Figure 2.2: Membership function for a fuzzy set A

- Maximum operator: $\mu_{A \bigcup B}(x) = \max\{\mu_A(x), \mu_B(x)\}$.
- Algebraic sum operator: $\mu_{A \bigcup B}(x) = \{\mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x)\}$.
- Bounded sum operator: $\mu_{A \bigcup B}(x) = \min\{1, \mu_A(x) + \mu_B(x)\}$.
- Drastic sum operator: $\mu_{A \bigcup B}(x) = \mu_A(x)$ if $\mu_B(x) = 0$, $\mu_{A \bigcup B}(x) = \mu_B(x)$ if $\mu_A(x) = 0$, or $\mu_{A \bigcup B}(x) = 1$ if $\mu_A(x), \mu_B(x) > 0$.

An **s-norm** operator satisfies the commutativity, monotonicity, associativity, and $\mu_{A \bigcup 0} = \mu_A$ properties.

Fuzzy intersection operators are known as **t-norm** operators (denoted as *). The t-norm operators possess the "ANDing" property since they implement the AND operation between the membership functions under consideration. Examples of fuzzy intersection operators are [169]:

- Minimum operator: $\mu_{A \bigcap B}(x) = \min\{\mu_A(x), \mu_B(x)\}$.

- Algebraic product operator: $\mu_{A \bigcap B}(x) = \{\mu_A(x)\mu_B(x)\}$.

- Bounded product operator: $\mu_{A \bigcap B}(x) = \max\{0, \mu_A(x) + \mu_B(x) - 1\}$.

- Drastic product operator: $\mu_{A \bigcap B}(x) = \mu_A(x)$ if $\mu_B(x) = 1$, $\mu_{A \bigcap B}(x) = \mu_B(x)$ if $\mu_A(x) = 1$, or $\mu_{A \bigcap B}(x) = 0$ if $\mu_A(x), \mu_B(x) < 1$.

t-norms also satisfy the commutativity, monotonicity, associativity, and $\mu_{A \bigcap 1} = \mu_A$ properties.

Additionally, the membership function for the fuzzy complement operator is defined as

$$\mu_{\overline{B}}(x) = 1 - \mu_B(x)$$

2.3.2 Fuzzy Reasoning

Fuzzy logic [277] is a mathematical discipline invented to express human reasoning in rigorous mathematical notation. Unlike classical reasoning in which a proposition is either true or false, fuzzy logic establishes an approximate truth value of a proposition based on linguistic variables and inference rules. A *linguistic variable* is a variable whose values are words or sentences in natural or artificial language [276]. An expert can form *rules* with linguistic variables, by using hedges, e.g. ‘more’, ‘many’, ‘few’, and connectors such as AND, OR, and NOT. These rules will be used by an inference engine to facilitate approximate reasoning.

2.3.3 Linguistic Variables

As defined by Zadeh [277], a linguistic variable is a variable whose values are words or sentences in a natural or artificial language. A linguistic variable is characterized by a quintuple $(\Omega,\ T(\Omega),\ X,\ G,\ N)$, where

- Ω is the name of the linguistic variable;
- $T(\Omega)$ is the term-set of Ω, i.e. the collection of its linguistic values;
- X is a universe of discourse;
- G is a syntactic rule which generates the terms in $T(\Omega)$; and
- N is a semantic rule which associates a meaning with each linguistic value.

$N(\omega)$ denotes a fuzzy subset of X for each $\omega \in$ T(Ω). Consider the following example to clarify the meaning of a linguistic variable. Let X be the universe of *network average delay*, A is the fuzzy subset *network average delay near 0.05 seconds*, and $\mu_A(\bullet)$ is the membership function for A. Here, *network average delay* is a linguistic variable, i.e. Ω=*network average delay*. The linguistic values of *network average delay* can be defined as $T(\Omega)$= {*very small delay, small delay, delay near 0.05 seconds, large delay, very large delay*}. Each linguistic value is characterized by a membership function which associates a meaning to that value. The universe of discourse, X, is a possible range of *network average delay*. $N(\omega)$ defines a fuzzy set for each linguistic value, $\omega \in$ T(Ω).

2.3.4 Fuzzy Rules

One of the major components of a fuzzy logic system are *rules*, which are expressed as logical implications. Fuzzy logic rules are "IF-THEN" rules, which define relations between linguistic values of outcome (i.e. the consequent) and linguistic values of condition (i.e. the antecedent) [1]. For example,

IF *monetary cost is low* and *maximum number of hops are low* and *network average delay is low* THEN *the solution is good.*

Here *monetary cost, maximum number of hops, network average delay*, and *solution* are linguistic variables and *low* and *good* are linguistic values.

Rules are a form of propositions. A *proposition* is an ordinary statement involving defined terms. In traditional propositional logic, an implication is said to be *true* if one of the following holds:

- both the antecedent and the consequent are true;
- both the antecedent and the consequent are false; and
- the antecedent is false, and the consequent is true.

Rules may be provided by experts or can be extracted from numerical data. In either case, *engineering rules* are expressed as a collection of IF-THEN statements. The following are important aspects needed to construct a rule [1]:

- understanding of linguistic variables;
- quantifying linguistic variables by using fuzzy membership functions;
- logical connections for linguistic variables;

- implications, i.e. "IF A THEN B"; and
- how different rules can be combined together to form other rules.

2.3.5 Fuzzy Logic System

A fuzzy logic system (FLS) [169], as illustrated in Figure 2.3, is a model of fuzzy based decision making in engineering applications. A FLS consists of the following components:

- A fuzzifier, which accepts crisp data as input and converts the data into fuzzy input sets. The fuzzifier is needed to activate rules which are expressed in terms of linguistic variables.
- An inferencing engine, which is governed by the *rules*. Rules are stored in a knowledge base. The inference engine carries out the decision-making process. The output of the decision-making process is fuzzy sets.
- A defuzzifier, which converts fuzzy output to crisp values. The defuzzifier is used if an application requires crisp output data. Generally, optimization applications do not require crisp output, in which case the defuzzifier is not used.

2.3.6 Common Fuzzy Operators

t-norms play an important role in fuzzy logic and many other areas [13]. Since multi-objective optimization problems require simultaneous optimization of all objectives under consideration, the "AND" operator in a fuzzy rule plays a crucial role in

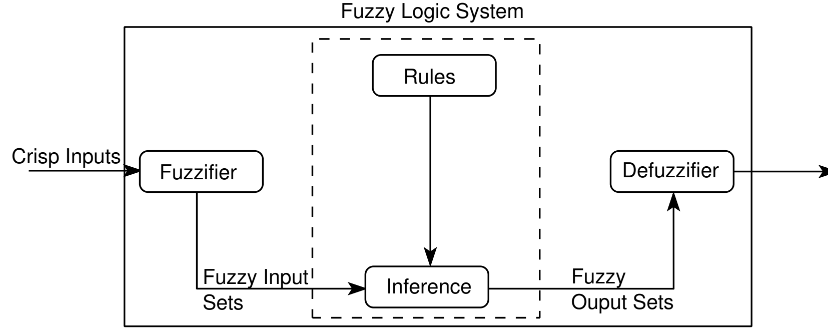

Figure 2.3: Fuzzy logic system

defining that rule. Consequently, a need arises to develop t-norm operators that will effectively handle the multi-objective nature of the problem by efficiently incorporating the characteristics of different objectives (through their membership functions) into one fuzzy rule. Furthermore, since the DLAN topology design problem also deals with simultaneous optimization of objectives, it is necessary to elaborate on different t-norm operators. A number of t-norm operators have been proposed in the literature, including Dombi's operator [61, 62], Einstein's operator [155], Hamacher's operator [113], Frank's operator [89], Weber's operators [252], Dubois and Prade's operator [69, 70], Schweizer's operators [224], Mizumoto's operators [177, 178], and Yager's operators [259, 261]. These operators are defined below.

Dombi : $$D(\mu_A, \mu_B) = \frac{1}{1 + \left(\left(\frac{1}{\mu_A} - 1\right)^{\beta} + \left(\frac{1}{\mu_B} - 1\right)^{\beta}\right)^{\frac{1}{\beta}}}, \quad \beta > 0 \tag{2.10}$$

Einstein : $$E(\mu_A, \mu_B) = \frac{\mu_A \mu_B}{2 - (\mu_A + \mu_B - \mu_A \mu_B)} \tag{2.11}$$

Hamacher : $$H(\mu_A, \mu_B) = \frac{\mu_A \mu_B}{\beta + (1 - \beta)(\mu_A + \mu_B - \mu_A \mu_B)}, \quad \beta \geq 0 \tag{2.12}$$

Frank : $$F(\mu_A, \mu_B) = \log_{\beta}\left(1 + \frac{(\beta^{\mu_A} - 1)(\beta^{\mu_B} - 1)}{\beta - 1}\right), \quad \beta > 0, \beta \neq 1 \tag{2.13}$$

$$\textbf{Weber 1}: W_1(\mu_A, \mu_B) = max \left\{ 0, \left(\frac{\mu_A + \mu_B - 1 + \beta\mu_A\mu_B}{1+\beta} \right) \right\}, \beta > -1 \tag{2.14}$$

$$\textbf{Weber 2}: W_2(\mu_A, \mu_B) = \max \{0, (1+\beta)\mu_A + (1+\beta)\mu_B - \beta\mu_A\mu_B - (1+\beta)\}, \beta > -1 \tag{2.15}$$

$$\textbf{Dubois and Prade}: \quad DP(\mu_A, \mu_B) = \frac{\mu_A\mu_B}{max\{\mu_A, \mu_B, \beta\}}, \quad 0 \leq \beta \leq 1 \tag{2.16}$$

$$\textbf{Schweizer 1}: \quad S_1(\mu_A, \mu_B) = \sqrt[\beta]{max \left\{ 0, (\mu_A^\beta + \mu_B^\beta - 1) \right\}}, \quad \beta > 0 \tag{2.17}$$

$$\textbf{Schweizer 2}: \quad S_2(\mu_A, \mu_B) = 1 - \sqrt[\beta]{(1-\mu_A)^\beta + (1-\mu_B)^\beta - (1-\mu_A)^\beta(1-\mu_B)^\beta}, \quad \beta > 0 \tag{2.18}$$

$$\textbf{Yager}: \ Y(\mu_A, \mu_B) = \beta \times \min\{\mu_A, \mu_B\} + (1-\beta) \times \frac{1}{2}(\mu_A + \mu_B), \quad 0 \leq \beta \leq 1 \tag{2.19}$$

From all the above operators, Yager's operator, also known as the ordered weighted average (OWA) operator, has received considerable attention in the domain of fuzzy multi-objective optimization [21, 175, 185, 218, 219, 242, 262] due to the operator's special characteristics discussed in the following section. One of the objectives of this thesis is to propose a new fuzzy operator, which is based on the properties exhibited by Yager's operator. Therefore, a detailed discussion of Yager's ordered weighted average operator is given below.

Ordered Weighted Averaging Operator

In MOO problems, where sub-objectives are aggregated to form an overall objective function, an important issue is how to form this overall function. Generally,

in MOO problems, here are two extreme approaches in creating a single objective function from sub-objective functions. In the first case, all objectives must be satisfied, which leads to the pure-ANDing operation. At the other extreme, any of the objectives can be satisfied, thus suggesting the pure-ORing operation. However, for many real-world problems, it is not desirable to formulate multi-objective decision functions with pure "ANDing" of t-norm operators nor the pure "ORing" of s-norm operators. The reason for this is the complete lack of compensation of **t-norm** operators for any partial fulfillment and complete submission of **s-norm** operators to fulfillment of any sub-objective. This observation led to the development of the Ordered Weighted Averaging (OWA) operator by Yager [260]. Because of its properties, the OWA operator has been quite popular among researchers working on multi-objective decision-making, and has been applied to many problems [21, 175, 185, 218, 219, 242, 262]. The OWA operator allows easy adjustment of the degree of "ANDing" and "ORing" in the function aggregating sub-objectives. "OR-like" and "AND-like" OWA for two fuzzy sets A and B are implemented as follows:

$$\mu_{A \cup B}(x) = \beta \times \max(\mu_A, \mu_B) + (1 - \beta) \times \frac{1}{2}(\mu_A + \mu_B) \tag{2.20}$$

$$\mu_{A \cap B}(x) = \beta \times \min(\mu_A, \mu_B) + (1 - \beta) \times \frac{1}{2}(\mu_A + \mu_B) \tag{2.21}$$

where μ represents the membership to the fuzzy set, determined by a membership function. $\beta \in [0, 1]$ is a constant parameter, which represents the degree to which the OWA operator resembles the pure "OR" or pure "AND" respectively.

It was shown by Yager that the OWA operator is a mean operator as it satisfies the monotonicity, symmetry, and idempotency conditions for a function, $\mathbf{F}(\mathbf{x}) =$

$(A_1(\mathbf{x}), A_2(\mathbf{x}), \cdots, A_K(\mathbf{x}))$ [260]. Before giving definitions of these conditions, recall that $A_1(\mathbf{x})$, $A_2(\mathbf{x})$, $\cdots$, $\mathrm{A}_K(\mathbf{x})$ are K objectives of a multi-objective problem, and $\mathbf{x}$ is a candidate solution. Accordingly, the corresponding membership functions of $A_1(\mathbf{x})$, $A_2(\mathbf{x})$, $\cdots$, $\mathrm{A}_K(\mathbf{x})$ are $a_1, a_2, \cdots, a_K$ respectively. With these details in view, the three conditions satisfied by the OWA operator can be defined as follows:

1. **Monotonicity:** $F(\ddot{a}_1, \cdots, \ddot{a}_K) \geq F(a_1, \cdots, a_K)$ if $\ddot{a}_k \geq a_k$, $\forall$ k $= 1, \cdots, K$.

2. **Symmetry (generalized commutativity):** $F(\ddot{a}_1, \cdots, \ddot{a}_K) = F(a_1, \cdots, a_K)$ for every permutation $(a_1, \cdots, a_K)$ of $(\ddot{a}_1, \cdots, \ddot{a}_K)$.

3. **Idempotency:** $F(a_1, \cdots, a_K) = a$, if $a_k = a$, $\forall\ k = 1, \cdots, K$.

A further analysis of Equations (2.20) and (2.21) reveals the importance of factor β. The value of β is crucial in deciding the extent of "ORing" or "ANDing". For example, in Equation (2.20), the more the value of β tends towards 1, the higher is the extent of "pure-ORing". As the value of β is lowered, the extent of pure-ORing is reduced, thus making it soft-ORing. Similarly, in Equation (2.21), a high value of β will incline the function towards pure-ANDing, while a low value of β will move the function towards soft-ANDing.

Figure 2.4 depicts the behavior of the OWA AND-like function of Equation (2.21). When $\beta = 0$ (Figure 2.4(a)), the function takes the average of all the membership values. This behavior is referred to as "pure-aggregation". As the value of β is increased, the "ANDing" between the two membership values also increases. When $\beta = 1$ (Figure 2.4(f)), the behavior of the function is exactly the same as that of pure-AND defined by Zadeh [276].

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

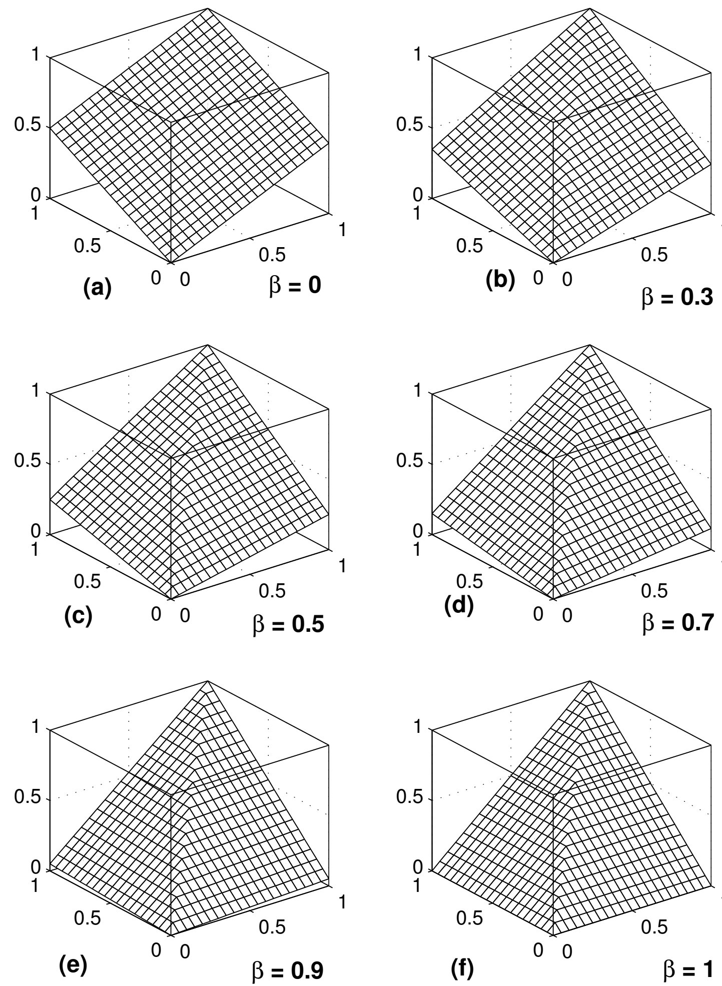

Figure 2.4: Effect of β on OWA-AND function

A similar pattern is observed for Equation (2.20) with respect to ORing. In Figure 2.5, different instances of the OWA OR-like function of Equation (2.20) are depicted. For $\beta = 0$ (Figure 2.5(a)), the behavior of the function is that of pure-aggregation. With increasing value of β, the extent of "ORing" between the two membership values also increases. When $\beta = 1$ (Figure 2.5(f)), the behavior of the function is that of pure-OR, as suggested by Zadeh [276].

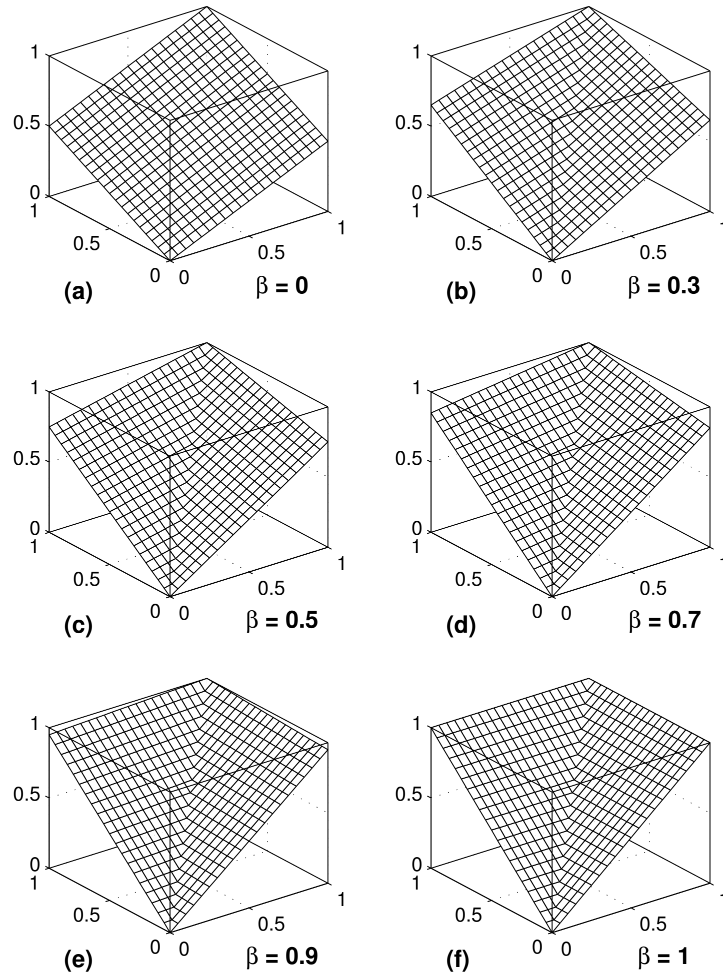

Figure 2.5: Effect of β on OWA-OR function

2.3.7 Role of Preferences in Multi-objective Optimization

Solving of a multi-objective optimization problem involves three phases [38]: measurement, search, and decision-making. In most MOO problems, the focus is on the search for non-dominated vectors, without any insight into the decision-making process. Thus, the decision-maker has to select one of several alternatives obtained. In these MOO problems, all individual objectives are considered as having an equal level of importance, since the decision-maker does not assign preference to an objective or set of objectives [38]. Even if the decision-maker's preferences are given, they are quite often not well stated and the function governing the preferences is imprecise or even arbitrary [47]. Thus, a strong concern is that the decision-maker's preferences are effectively incorporated into the MOO search process to focus on regions of greater interest. A number of proposals to handle preferences have been reported in the literature. For example, Fonseca and Fleming [86] utilized the goal attainment technique to accommodate preferences in the search process. Cvetković and Parmee [47] used binary preference relations which are expressed as words and then translated into weights to narrow the search. Greenwood *et al.* [108] used the idea of *imprecisely specified multi-attribute value theory* (ISMAVT) from imprecise ranking of objectives.

2.4 Optimization Algorithms

Combinatorial optimization (CO) deals with models and methods for optimization over discrete choices [196]. CO involves selection of a solution from a finite set of possible solutions [196]. CO has a strong relationship with discrete mathemat-

ics, probability theory, algorithmic computer science, and complexity theory [196]. Combinatorial optimization algorithms can be categorized into two general classes: (1) *exact algorithms* and (2) *approximation algorithms.* Exact algorithms reach an exact solution, while approximation algorithms seek an approximation that is close to the best solution. Approximation may use either a deterministic or a random strategy. Linear programming, dynamic programming, branch-and-bound, and backtracking are some well-known examples of exact algorithms [119]. Examples of approximation algorithms include local search, constructive greedy methods, and many general iterative heuristics.

A large number of optimization problems are NP-hard. Due to the complexity of NP-hard problems, one cannot resort to exact techniques to solve such optimization problems. For such problems, approximation algorithms, also known as *heuristics*, seem to be more effective. One strong feature of a heuristic is that it explores only a sub-region of the total search space and finds an "excellent" feasible solution rather than the best solution. This feature of heuristics provides an edge over exact techniques with regard to computational complexity: the execution time for a heuristic, in general, is remarkably less than that of an exact algorithm.

Heuristics are divided into two main categories: *constructive* and *iterative* methods. The main difference between the two categories is in the approach adopted in reaching the final solution. Iterative heuristics attempt to improve a complete solution by making controlled stochastic moves, while constructive heuristics construct a solution in a piecewise manner. Although constructive heuristics are faster than iterative heuristics in generating a final solution, the former often do not reach a global optimal solution. For highly constrained problems, constructive heuristics

may even fail to find a feasible solution. Some examples of constructive heuristics are Esau-William's algorithm [80], Prim's algorithm [208], and Kruskal's algorithm [151]. On the other hand, iterative heuristics have proven to be effective for a variety of NP-hard problems in the field of engineering [19, 22, 76, 143, 153, 179, 211, 271] and science [95, 197, 214, 273]. Iterative heuristics include simulated annealing (SA) [186], simulated evolution (SimE) [147, 148], genetic algorithms (GA) [104], stochastic evolution (StocE) [216, 217], and more recently, ant colony optimization (ACO) [42] and particle swarm optimization (PSO) [140]. For many NP-hard problems, these heuristics have the ability to find near optimal solutions when properly engineered, irrespective of the initial solution from which they start the search. An overview of these algorithms is given below.

2.4.1 Genetic Algorithm

Genetic algorithms (GA) are a popular and effective optimization algorithm, which emulates the natural process of evolution as a means of progressing toward an optimum. Initially suggested by Fraser [90], Fraser and Burnell [91], and Crosby [46], and popularized by Holland [118], the GA was inspired by Darwinian theory [49]. The foundation of GA is based on the theory of natural selection, whereby individuals having certain positive characteristics have a better chance to survive and reproduce, and hence transfer their characteristics to their offspring. Since a number of variations of GA exists, the term is generally referred to as GAs (in plural) in the literature.

GAs operate on a set of solutions in parallel. The set of solutions is known as a *population*. Each solution (also referred to as a chromosome) in the population is

represented by a string of symbols. A chromosome comprises individual elements, called *genes*. During each iteration, a new set of chromosomes, called *offspring* is generated. Each offspring is a result of four genetic operators, namely, *selection*, *crossover*, *mutation*, and *inversion* (optional). These operators are repeatedly applied to a collection of solutions to generate the new offspring.

An important issue in applying GAs to solve combinatorial optimization problems is to find an efficient way of representing a solution in the form of a chromosome. Moreover, the *fitness* of each chromosome needs to be evaluated based on some fitness function. The fitness value quantifies the quality of the solution represented by a chromosome. For a maximization problem, the higher the fitness value, the higher the quality of the solution, and vice versa. As evolution progresses, better quality solutions are expected to be produced. It is important that the fitness function is an accurate reflection of the problem domain. An improper selection of fitness function and representation may lead to poor performance. Note that both aspects are problem-dependent.

An overview of the main operators of GAs, namely, selection, crossover, and mutation is presented below.

Selection

The selection operator is a key element of GAs. Selection operators are used to choose a pair of chromosomes (called parents) to produce offspring. The choice of parents plays an important role in generating high-quality offspring. The selection process usually favors chromosomes with high fitness values. The logic behind this is that stronger (i.e. fitter) chromosomes are more likely to produce stronger offspring.

If the process continues, high-quality offspring are expected in each generation. In broader terms, the main objective of selection methods is to exploit the search space [9]. A number of selection methods such as roulette-wheel selection [104], rank selection [107], tournament selection [104], and elitism [107] have been proposed in the literature.

Crossover

The purpose of crossover is to provide a mechanism for producing offspring such that the offspring inherits the characteristics of both the parents. Crossover occurs at a user-specified probability, referred to as the crossover probability. This probability typically ranges between 0.4 and 0.8 [253]. A number of crossover schemes have been proposed in the literature, including simple [104], cyclic [190], order [104], partially mapped [103], uniform [239], arithmetic [104], and heuristic [256] crossover.

Mutation

The basic purpose of mutation is to perturb chromosomes in order to introduce characteristics which are absent in the parent population. This is turn allows more exploration of the search space. Mutation is performed on the offspring produced by the crossover operator. The mutation operation is also applied at a user-specified probability, referred to as the *mutation probability*, p_m. A typical value of p_m is taken as 0.01 [116], since high values p_m of will result in large perturbations in the offspring, which is undesirable. A more appropriate measure of p_m is by setting p_m to the inverse of the number of genes in a chromosome [104]. However, the most suitable value of p_m is problem-dependent [193].

Some applications of GAs

Extensive literature is available on GAs and their use in combinatorial optimization. The interested reader is referred to Goldberg [104]. As for network design problems, genetic algorithms have been widely used. Pierre *et al.* [204] used a GA to solve the topological design problem of distributed packet-switched networks. Dengiz *et al.* [58, 57] used a GA to optimize network topology using cost and reliability as objective functions. Gen *et al.* [98] used a GA for topological network design based on spanning trees. Gen *et al.* optimized the average network message delay and connecting cost. Elbaum *et al.* [75] used a GA for designing LANs with the objective of minimizing the average network delay under the constraint that the flow on any link does not exceed the capacity of that link. Ombuki *et al.* [192] proposed a GA for the 3-connected computer networks design problem. This problem dealt with assigning a set of links to computer sites (nodes) such that every source-destination pair of nodes could successfully communicate with every other via at least one of three diverse paths. The objective was to minimize the total link connection costs while maintaining the 3-connectivity constraint. Xianhai *et al.* [240] proposed a multi-objective GA to design a computer network considering cost, mean path delay, and mean link utilization ratio as the optimization objectives. Mostafa *et al.* [182] proposed a GA to address the issue of joint optimization of capacity and flow assignments. The aim was to find the flows on different links as well as the capacities of the links, such that the total network cost is minimized while keeping the average network delay below a certain upper limit. White *et al.* [254] proposed a GA for ring network design while considering routing, link capacity assignment, and ring determination as three design objectives.

2.4.2 Simulated Evolution

Simulated evolution (SimE) is a general iterative heuristic proposed by Kling and Banerjee [147, 148, 149] (refer to Figure 2.6). SimE is based on the analogy with the principles of natural selection thought to be followed by various species in their biological environments. During the process of biological evolution, organisms tend to develop features that help them in adapting to their habitat.

SimE belongs to the category of algorithms which emphasize the behavioral link between parents and offspring, or between reproductive populations, rather than the genetic link [85]. In other words, SimE follows phenotypic evolution instead of genotypic evolution. SimE combines iterative improvement and constructive perturbation, and prevents itself from getting trapped in local minima by following a stochastic perturbation approach. The algorithm iteratively operates a sequence of evaluation, selection and allocation steps on a single solution until some stopping condition is satisfied. The selection and allocation steps constitute a compound move from the current solution to another feasible solution of the state space. These steps are described below.

Initialization

In this very first step, a *valid* solution (i.e. a solution that satisfies all constraints defined for the problem) is generated. The initial solution may be generated randomly or by using any constructive algorithm.

Evaluation

A solution is seen as a set of movable elements. A movable element is an individual component of the solution. Different arrangements of movable elements result in different unique solutions. Each element, e_i, has an associated *goodness* (fitness) measure, g_i, in the interval [0,1], defined as

$$g_i = \frac{O_i}{C_i} \tag{2.22}$$

where O_i is an estimate of the optimal cost of element e_i and C_i is the actual cost of e_i in its current location. Notice that O_i remains constant throughout iterations. The value of O_i is generally obtained through a mathematical approximation and is set in the initialization phase. However, C_i changes in each iteration, and needs to be recalculated in each call to the evaluation function. C_i is obtained through a user-defined evaluation measure. The element, e_i, is analogous to a *gene* of a *chromosome* in GAs.

Selection

Selection is the third step of the SimE algorithm. This step takes as input the existing solution and the goodness of each element, e_i, estimated during the evaluation step. Elements having a low goodness value are selected for removal from the solution, so that new elements can replace them. The selection is performed using a parameter B, called the *bias* which is used to compensate for inaccuracies in the goodness measure. A random number is generated, and the following check is performed for each e_i:

IF $Random > Min\{g_i + B, 1\}$

Then select e_i for removal

ELSE do not select e_i

where $Random \sim U(0,1)$. The outcome of this operation decides whether an individual is to be removed or not. The higher the goodness value of the element, the higher is its chance of staying in its current location. The lower the goodness, the larger is the probability that the element will be removed. The bias, B, has an important role in selecting the number of elements to be removed. A high bias value inflates the goodness of each element, thus reducing the number of elements selected for removal (and then re-allocation). This speeds-up the algorithm, but at the risk of early convergence to a local optimum. A low (or negative) bias increases the number of elements selected at each iteration, allowing the algorithm to search rigourously at each iteration. This may result in better solutions, but at the expense of higher runtime requirements. If the bias is removed, the algorithm will still work with the default setting (i.e. with g_i only), but the presence of bias alters the default setting in order to extend or reduce the survival chances of elements, thereby controlling the average number of elements per iteration [146]. It is important to mention that the value of bias B is a user-defined value in the range [-1,1] and an appropriate bias value is found by trial and error.

The selection phase is non-deterministic. Therefore, there is still a non-zero probability for an element with a high goodness to be selected for removal from the solution. Due to this characteristic of non-determinism, SimE is capable of escaping local minima [147, 149].

$Simulated_Evolution(B, \Phi_{initial}, StoppingCondition)$
NOTATION
$B=$ Bias value
$\Phi=$ Complete solution
$e_i=$ Individual in Φ
$O_i=$ Optimum cost of e_i
$C_i=$ Current cost of e_i in Φ
$g_i=$ Goodness of e_i in Φ
$H_q=$ Queue to store the selected individual
$ALLOCATE(e_i, \Phi_i)$=Function to allocate e_i in partial solution Φ_i
Begin
Repeat

EVALUATION: **ForEach** $e_i \in \Phi$ **DO**
begin
$g_i = O_i/C_i$
end
SELECTION: **ForEach** $e_i \in \Phi$ **DO**
begin
IF $Random > Min\{g_i + B, 1\}$
THEN
begin
$H_q = H_q \cup e_i; \Phi = \Phi \setminus \{e_i\}.$
end
end
ALLOCATION: **ForEach** $e_i \in H_q$ **DO**
begin
$ALLOCATE(e_i, \Phi_i)$
end

Until *Stopping Condition is satisfied*
Return Best solution.
End $(Simulated_Evolution)$

Figure 2.6: Structure of the simulated evolution algorithm

Allocation

The allocation step is the most important step of the algorithm, and has the most significant impact on the quality of the solution. This step removes the elements selected during the selection step, and *moves* are made for each selected element. For each selected element, trial moves are performed and the cost of a new solution resulting from each move is computed. The move which gives the most optimized cost is accepted. These steps are repeated for each *selected* element. The number of trial moves, as well as the types of moves performed, are problem-specific. For example, for the travelling salesman problem (TSP), a move may consist of swapping vertex i with any other vertex. The number of moves could then be $n - 1$, where n is the number of vertices. However, a smaller number of moves would be more appropriate to keep the runtime under control. At the end of the allocation step, a new solution is obtained.

The objective of the allocation phase is to favor improvements in the quality of the existing solution, without being 'too greedy'. As mentioned above, out of the many trial moves, the 'best move' results in the most optimized cost. However, this cost might be worse than the original cost (i.e., when the element selected for removal was present in the solution). Despite this, the new move with the best cost among all trial moves is accepted. This momentarily causes the solution to go towards a worse solution. However, notice that trial moves are performed for each selected element. Some of the accepted moves might be of lower cost than the original, while other may be of higher cost. The net effect is that the new complete solution (after all selected elements are re-allocated) could be better or worse than the previous solution. Thus, the allocation phase is not very greedy in accepting

only good solutions; the phase also accepts bad solutions.

The allocation step is somewhat similar to the mutation operation in genetic algorithms, but is relatively more complex in nature. As mentioned earlier, mutation and crossover are the two main operations in GAs that contribute strongly to the evolution and inheritance in obtaining better quality solutions. The same specific task has to be performed by the allocation function in SimE. Therefore, it is obvious that allocation should be a well-engineered and sophisticated operation compared to the mutation operation of GAs.

Applications of SimE

Since the SimE algorithm was originally proposed for design automation of very large scale integration (VLSI), and research papers appeared only in related conferences and journals, the algorithm did not receive much attention and many researchers are not aware of the algorithm. SimE has been applied in only a few research articles outside the field of VLSI such as the driver scheduling problem [127], the set covering problem [128], and the operand data type problem [264].

2.4.3 Stochastic Evolution

Stochastic evolution (StocE) is another randomized iterative search algorithm, also inspired from biological evolution [215, 216, 217]. The StocE algorithm seeks to find a suitable location, $Z(e_i)$, for each movable element, e_i, which eventually leads to a lower cost of the whole state, $Z \in \mho$, where $\mho$ is the state space. The basic idea of the algorithm is to reward additional iterations to the algorithm if improvement is observed. A general outline of the StocE algorithm is given in Figure 2.7.

$Stochastic_Evolution(Z_0, p_0, R)$
$NOTATION$
Z_0= Initial solution
ρ= Counter
p= Control parameter
p_0= Initial value of p
p_{incr}= User-defined value
R_c= Stopping criterion parameter
$Ccur$ = Cost of current solution Z
$CBest$ = Cost of best solution
$Cpre$ = Cost of previous solution
Begin
  ZBest = $Z = Z_0$;
  CBest = Ccur = $Cost(Z)$;
  $p = p_0$;
  $\rho = 0$;
  Repeat
    Cpre = Ccur;
    $S = PERTURB(Z, p)$; /* perform a search in the neighborhood of Z */
    Ccur = $Cost(Z)$;
    $UPDATE(p$, Cpre, Ccur); /* update p if needed */
    if (Ccur < CBest)
      ZBest =Z;
      CBest = Ccur;
      $\rho = \rho - R_c$; /* Reward the search with R_c more generations */
    else
      $\rho = \rho + 1$;
    endif
  until $\rho > R_c$
  return (ZBest);
End

Figure 2.7: The stochastic evolution algorithm

The inputs to StocE are an initial solution, a parameter, R_c, for the stopping criterion, and a control parameter, p, used to control the uphill climbs to escape local minima. Another variable, p_{incr}, is used to increment the value of p. A counter, ρ, is used with the main loop of the algorithm, where the loop is iterated until the value of ρ becomes equal to R_c. ρ is a variable, and is updated according to the result of a perturbation. Each time a state is found which is lower in cost than the best cost obtained so far, ρ is decremented by R_c, giving the algorithm a chance to find better solutions by using more execution time. In other words, the algorithm is rewarded with more iterations before terminating. If no improvement is observed for a number of iterations (determined by ρ and R_c), then the algorithm stops since this indicates that StocE has converged.

There are three main steps of the StocE algorithm, namely, initialization, perturbation, and updating. These steps are discussed below.

Initialization

The first step of the StocE algorithm initializes a valid solution, Z. Other algorithm parameters such the control parameter, p, and the counter, ρ, are also initialized.

Perturbation

Following initialization, a **repeat** loop is executed. Within the **repeat** loop, the 'perturb' function (refer to Figure 2.8) is invoked to make a compound move (i.e. a move comprised of multiple single moves) from the current state (i.e. current solution), Z. The objective of 'perturb' is to obtain a new solution by perturbing the current solution. Once a valid move is done, the gain is evaluated by calculating

the difference between the cost of the old solution, Z, and the new solution, Z'. If the gain is greater than a randomly generated integer in the range $[-p, 0]$, the move is accepted and Z' replaces Z as the current solution. Moves with positive gains are always accepted. The new solution generated by 'perturb' is returned to the main procedure as the current solution. Validity of a move is ensured using a sub-function MAKE_STATE which checks whether all design constraints are satisfied. In case a move is invalid, the MAKE_STATE function reverses the move and restores the previous valid state of the solution.

```
FUNCTION perturb(Z, p);
Begin
    for each (q ∈ Q) do    /* according to some apriori ordering */
        Z' = MOVE(Z, q);
        Gain(q) = Cost(Z) − Cost(Z');
        if (Gain(q) > RANDINT(−p, 0)) then
                Z = Z'
        endif
    endfor;
    Z =MAKE_STATE(Z); /* make sure Z satisfies constraints */
    return(Z)
End
```

Figure 2.8: The Perturb function

Updating

Following the perturbation phase, the next stage in the **repeat** loop is the **update** routine (Figure 2.9). The 'update' routine takes the cost the previous current solution and the new current solution and compares the cost. If the two costs turn out to be the same, then there is a possibility that the algorithm has reached a

local minimum. To escape the local minimum, p is increased by p_{incr} to allow uphill moves. Otherwise, p is reset to p_0.

The two key parameters that affect the performance of the StocE algorithm are p and R_c. As said above, the purpose of p is to help the algorithm escape local minima by allowing uphill climbs. The value of p controls the steepness of this uphill climb. In other words, p determines the extent of accepting a solution of worse quality than the quality of the current solution. Initially, p is set to a non-negative value close to zero [217]. Such a choice for p means that only moves with small negative gains are performed. A high value of p will result in moves with large negative gains. Large negative gains are undesired since they increase the runtime of the algorithm. Therefore, it is important to find an appropriate value of p, and by what factor the value of p should be increased. The parameter R_c represents the expected number of iterations needed by the StocE algorithm until an improvement in the cost is achieved with respect to the best solution seen so far. If R_c is too small, the algorithm will not have enough time to improve the initial solution, and if R_c is too large, the algorithm may waste too much time during the later generations. Experimental studies suggest that good results are obtained when R_c is between 10 and 20 [215].

Applications of StocE

As is the case with SimE, StocE has not been exploited much by researchers. The algorithm has been applied to only a few problems, including channel router design [82], register allocation [248], the graph covering problem [48], and a technology mapper for field programmable gate arrays [4].

PROCEDURE update(p, Cpre, Ccur);
begin
  if (Cpre=Ccur) **then** /* possibility of a local minimum */
    $p = p + p_{incr}$; /* increment p to allow larger uphill moves */
  else
    $p = p_0$; /* re-initialize p */
  endif;
end

Figure 2.9: The update procedure for stochastic evolution algorithm

2.4.4 Simulated Annealing

Simulated annealing (SA) is a popular combinatorial optimization algorithm proposed by Kirkpatrick *et al.* [145]. It is derived from the analogy of the physical annealing process of metals. SA works on a single solution. The neighborhood state of the solution is generated by making a move. All good moves are accepted. However, bad moves are stochastically accepted. The acceptance probability of bad moves is controlled by a *cooling schedule*. In the early stage of the search, bad moves are accepted with high probability. However, as the search progresses, the *temperature* of the cooling schedule decreases and so does the probability of accepting bad moves. In the last part of the search, SA behaves as a greedy algorithm, accepting only good moves. The algorithm is depicted in Figure 2.10. SA has two main stages: initialization and the metropolis procedure. These stages are described below.

Initialization

The first step of the SA algorithm initializes a valid solution, and values are assigned to the SA control parameters. These parameters include the initial temperature,

$Simulated_Annealing(Z_0, T_0, \alpha_{SA}, \beta_{SA}, M, MaxTime)$
  Z_0 = Initial solution
  T_0 = Initial temperature
  α_{SA} = Cooling rate
  β_{SA} = A constant
  $MaxTime$ = Total allowed time for the annealing process
  M = Time until the next parameter update
Begin

  $T = T_0$;
  $Z = Z_0$;
  $Time = 0$;
  Repeat
    Call $Metropolis(Z, T, M)$
    $Time = Time + M$;
    $T = \alpha_{SA} \times T$;
    $M = \beta_{SA} \times M$;
  Until (Time $\geq MaxTime$);
  Output best solution found
End

$Metropolis(Z, T, M))$
Begin
  Repeat
    $Z' = neighbor(Z)$;
    Δh = Cost(Z') – Cost(Z);
    if$((\Delta h < 0)$ **or** $(random < e^{-\Delta h/T}))$ **then** $Z = Z'$; {accept the solution}
    $M = M - 1$;
  Until $(M = 0)$;
End $(Metropolis)$

Figure 2.10: Structure of the simulated annealing algorithm

T_0, the cooling rate, α_{SA}, the constant, β_{SA}, the maximum time for the annealing process, $MaxTime$, and the length of the Markov chain, M, which represents the time until the next parameter update.

Metropolis Procedure

The metropolis procedure is the core of the annealing algorithm and is performed repeatedly until a predefined number of iterations is reached. The metropolis procedure uses a function *neighbor* to generate a local neighbor, Z', of any given solution Z. The function *neighbor* performs the following steps: a single move is made. The move is accepted or rejected based on the constraints of the problem, and the new cost is calculated according to the metropolis procedure. The function **cost** returns the *overall cost* of the given solution Z. If the overall cost of Z' is better than the cost of Z, then Z' is definitely accepted, otherwise Z' is accepted probabilistically based on the *metropolis criterion*. The metropolis criterion is given by $P(random < e^{-\Delta h/T})$, where $random$ is a random number in the range 0 to 1, Δh represents the difference in the overall goodness of Z and Z', and T represents the *annealing temperature*.

The control parameters have an impact on the convergence of the SA algorithm. Inappropriate values of these parameters can significantly affect the quality of solution produced by SA. One such parameter is the initial temperature, T_0. The initial temperature should be set to an appropriate value, so that all transitions (i.e. moves) are accepted initially. A very high initial temperature will unnecessarily increase the algorithm execution time, since the algorithm would navigate the search space blindly (thus increasing exploration). In other words, the algorithm will then

implement a blind search without any intelligence. On the other hand, a very low value of T_0 will favor too much exploitation, leading to premature convergence, and the algorithm will reject bad solutions even in the early steps of the search. Therefore, a suitable value of T_0 should be chosen, depending on the nature of the problem being solved. A number of approaches have been reported in the literature to find a suitable value for T_0 [33, 126, 145, 167, 202].

The cooling rate, α_{SA}, also has an impact on the performance of the algorithm. With a high value of α_{SA}, the temperature will decrease slowly. This implies that the capability of the algorithm for accepting bad solutions will persist for a considerable amount of time, which may help the algorithm to escape from local minima (thus favoring exploration). If α_{SA} is very low, then the algorithm will quickly lose the tendency of accepting a bad solution. This may cause the algorithm to become stuck in a local minimum. Since small changes in the solution are desired, a value close to unity is chosen for α_{SA}, typically ranging from 0.8 to 0.99 [154].

Another important parameter that affects the convergence of the algorithm is the length of the Markov chain, denoted by M, which represents the number of times the algorithm makes perturbations at a particular temperature. Determination of an appropriate value of M depends on the fact that a minimum number of transitions should be accepted in each iteration. This minimum number is user-defined and depends on the nature of the problem. The value of M should neither be very high nor very low. A very high value of M increases the execution time, since the algorithm performs more transitions than necessary. For example, if $M = 25$, then for an arbitrary temperature, the algorithm will attempt 25 moves (i.e. transitions). However, the same quality of solution might be achieved with $M = 10$. Unnecessary

moves are therefore made for $M = 25$. If M is too small, the solution might not be perturbed enough to search for better solutions in the current neighborhood. Kirkpatrick *et al.* [145] proposed that M should be chosen such that a specific number of solution transitions are accepted. This implies that it is possible to define the maximum value of M equal to the neighborhood size. However, to save the execution time, it is more appropriate to take M equal to some subset of the neighborhood, rather than the entire neighborhood [145].

It is worth mentioning that, during the course of execution of the algorithm, parameter β_{SA} (where $\beta_{SA} > 1$) is used to increase the value of M as temperature is decreased. The basic idea behind using β_{SA} is that the number of moves increases as temperature decreases. In the initial stages of the algorithm, a few moves are performed. These moves are sufficient to escape local minima, since the algorithm possesses the tendency to accept bad moves. As the temperature is decreased, the algorithm's tendency to accept bad moves is reduced, thus reducing the chances of escaping local minima. Under this condition, more moves are needed at a particular temperature to escape local minima, thus enhancing the chances of converging towards a better solution.

Some applications of SA

Simulated annealing has been used extensively for a variety of problems in different disciplines, including network design problems. Miyoshi *et al.* [176] investigated the use of SA for topological designs of multicast networks, and proposed a new method for finding an effective initial solution to the problem of reducing the computational time of SA. Harmatos *et al.* [114] proposed a heuristic planning algorithm for opti-

mizing tree-topology access networks. The algorithm is a combination of an adaptive version of the SA meta-heuristic and a local improvement strategy. Thompson and Bilbro [241] empirically compared a GA and SA on the problem of optimizing the topological design of a network. In addition to the usual problem of optimizing only the placement of links, the number and placement of concentrators were also decision variables for a class of problems using a real set of concentrators, links, and traffic. The results found by the GA and SA were comparable for all test cases. Ersoy *et al.* [79] used SA for topological design of interconnected LANs/MANs. The main objective was to minimize the average network delay. Fetterolf [83] used SA to design LAN-WAN computer networks with transparent bridges. SA was used to generate sequences of neighboring spanning trees, and to evaluate design constraints based on maximum flow, bridge capacity, and end-to-end delay. Atiq *et al.* [12] proposed a SA algorithm for reliability optimization. Similarly, Dengiz *et al.* [56] used SA to design computer communication networks, with reliability as the optimization objective. The results were almost of the same quality when compared with a GA.

SA has not been exploited well for multi-objective optimization problems. Research includes that of Venanzi *et al.* [250] where a multi-objective SA algorithm was used to design optimal wind-excited structures (such as masts and lattice towers). Chattopadhyay *et al.* [30] developed a multi-objective optimization procedure based on SA to simultaneously optimize the synthesis of structures/controls and the actuator-location problem for the design of intelligent structures. Bandyopadhyay *et al.* [14] proposed a multi-objective SA algorithm and tested it on a number of mathematical benchmarks.

2.4.5 Tabu Search

Tabu search (TS) is another single solution iterative heuristic used for solving combinatorial optimization problems. The algorithm was first proposed by Glover [100, 101]. The algorithm is biologically inspired by "memory" - the ability to use past experiences to improve current decision-making. There are two key features of the tabu search algorithm, namely the tabu list and the aspiration criterion. The tabu list is a mechanism through which the algorithm prevents cycling. In essence, the tabu list controls the memory component of the algorithm, by reminding the algorithm which moves have already been undertaken in the recent past. The tabu list maintains a record of recently visited solutions, and no moves leading to tabu solutions are allowed. However, the tabu status of a solution is overridden when *aspiration criteria* are satisfied. Aspiration criteria are defined on the basis of the nature of the problem being solved.

The tabu search algorithm works as follows: the search starts with a valid initial solution, labelled as the current solution. Then, the neighborhood of this current solution is generated and explored, and the best solution in that neighborhood is selected as the new solution, even if this best solution is worse in quality than the existing current solution. However, acceptance of this new solution in the neighborhood is subject to the condition that the solution is not in the tabu list. If it is in the tabu list, then it satisfies the aspiration criteria. If the above conditions fail, then the next trial solution is examined. This process is repeated until a stopping condition is met. The best solution found is returned as the result of the TS algorithm. The pseudo-code of the algorithm is depicted in Figure 2.11. Tabu search has two main stages: initialization and the tabu search procedure. Both are described

below.

Ω	:	Set of feasible solutions
S	:	Current solution
S^*	:	Best admissible solution
$Cost$	:	Objective function
$\aleph(S)$	:	Neighborhood of $S \in \Omega$
$\mathbf{V}^*$	:	Sample of neighborhood solutions.
	:	Tabu list
AL	:	Aspiration Level

Begin
Start with an initial feasible solution $S \in \Omega$;
Initialize tabu lists and aspiration level;
For fixed number of iterations
    Generate neighbor solutions $\mathbf{V}^* \subset \aleph(S)$;
    Find best $S^* \in \mathbf{V}^*$;
    IF move S to S^* is not in tabu list **THEN**
        Accept move and update best solution;
        Update tabu list and aspiration level;
        Increment iteration number
    else
        if $Cost(S^*) < \mathbf{AL}$ **then**
            Accept move and update best solution;
            Update tabu list and aspiration level;
            Increment iteration number
        endif
    endif
endfor
End

Figure 2.11: Algorithmic description of tabu search

Initialization

The first step of the TS algorithm initializes a valid solution, Z. The tabu list and aspiration level are also initialized in this step.

Tabu Search Procedure

A neighborhood, N(Z), is defined for Z. A subset of all neighbor solutions, V*(Z) $\subset$ N(Z), is generated. Solutions in V*(Z) are evaluated, and the best (in terms of an evaluation function), call it Z^*, is considered to be the next solution. A list of *attributes* of accepted moves is maintained by the algorithm in the tabu list. The size of the tabu list determines the number of iterations for which the move would remain tabu. An *attribute* is some characteristic associated with a move which is saved in the tabu list. The reason for saving only the attribute instead of the whole solution is that the solution cannot be stored when the solution representation is large or complex. If the move leading to Z^* is not defined as tabu in the tabu list, then Z^* is accepted as the new solution, even if it results in a worse solution compared to the current solution in terms of the evaluation function. However, if the tabu list defines the move leading to Z^* as tabu, then the solution is not accepted until it has one or more features that allow the algorithm to accept it (i.e. the solution) by overriding its tabu status. An *aspiration criterion* is used to check whether the tabu solution is to be accepted or not [102]. The tabu search loop is repeated until a stopping condition is satisfied.

The key factor that affects the performance of the tabu search algorithm is the size of the tabu list. As mentioned above, the basic role of the tabu list is to prevent cycling. If the size of the list is too small, then the algorithm might not be able to prevent cycling efficiently. Conversely, a very long list creates too many restrictions on the visited solution, thereby barring the algorithm from exploring the search space freely. For any optimization problem, it is very difficult to find a tabu list size that prevents cycling and also does not excessively restrict the search for all

instances of the problem of a given size [221]. Therefore, an appropriate size needs to be determined for effective performance of the tabu search algorithm.

Some applications of TS

Tabu search has been applied to a variety of optimization problems, including applications to various areas of network design. Fortin *et al.* [88] proposed a mathematical model for the dimensioning of a 3G multimedia network, and designed a tabu search heuristic to solve the dimensioning problem. Subrata *et al.* [237] used a genetic algorithm (GA), tabu search (TS), and an ant colony algorithm (ACA) to solve the reporting cells planning problem. The effectiveness of each algorithm was shown for a number of test problems. Tabu search showed the best performance, followed closely by ant colony algorithms. Chamberland *et al.* [28] studied the problem of expanding cellular networks in a cost-effective way, and presented a mathematical formulation of the network expansion problem. A tabu search algorithm for finding "good" solutions is proposed, and results were compared to a proposed lower bound. These results showed that the tabu-based approach produced solutions close to the lower bound. Karasan *et al.* [134] proposed a tabu search based heuristic for the mesh topology design problem in overlay virtual private networks. For all test cases, the tabu search heuristic produced results within 2.5% of the optimum. Pierre *et al.* [203] used the tabu search algorithm for designing computer network topologies with unreliable components. Their simulation results showed that the tabu search algorithm is efficient for designing backbone networks. Similarly Dengiz *et al.* [55] used the tabu search algorithm for computer network design while considering reliability as the optimization function. Their TS outperformed a GA upon comparison.

2.4.6 Ant Colony Optimization

Ant algorithms are multi-agent systems in which the behavior of each agent, called an artificial ant (or ant for short in the following), is inspired by the behavior of real ants [44]. Ant Colony Optimization (ACO) [63] is a relatively new meta-heuristic for solving combinatorial optimization problems. ACO has a combination of distributed computation, autocatalysis (positive feedback) and constructive greediness to find an optimal solution for combinatorial optimization problems [65]. This technique tries to mimic the ants' behavior in the real world. Ant algorithms are one of the most successful examples of swarm intelligent systems [20], and have been applied to many types of problems, ranging from the classical travelling salesman problem, to routing in telecommunications networks.

The inspiration for the development of the ACO algorithms came from the experiments conducted by Goss *et al.* [105] using a colony of real ants. One important observation from the experiments was that real ants were able to select the shortest path between their nest and food resource, in the presence of alternative paths between the two points. The ants made this search possible by an indirect communication mechanism known as *stigmergy*. In this process, ants deposit a chemical substance called *pheromone* on the ground while travelling. When a point comes where there are multiple paths, and ants have to make a decision, the choice of path is probabilistic. This choice is based on the intensity of pheromone encountered on the paths. This behavior has a rippling effect for ants to follow, due to the fact that choosing a path increases the probability that the same path will be chosen again by future ants, since the higher pheromone deposit on the path will enhance the probability of choosing the same path (despite the fact that pheromone evapo-

ration on the paths also take place). Thus, new pheromone will be released on the chosen path when following ants visit the path, which consequently makes it more attractive for future ants. Eventually, all ants will be using the same path.

The meta-heuristic consists of an initialization step and three algorithmic components, as depicted in the generic specification in Figure 2.4.6. These algorithmic components undergo a loop that consists of:

1. the construction of solutions by all ants,

2. the (optional) daemon actions, and

3. the update of the pheromones.

Construction of Solutions

For the ACO meta-heuristic, the optimization problem is formulated as a graph, $G = (C, L)$, where C is the set of components of the problem, and L is the set of possible connections or transitions among the elements of C. The solution is expressed in terms of feasible paths on the graph, G, with respect to a set of given constraints. An ant is defined as a simple computational agent, which iteratively constructs a solution for the problem to solve [66]. A colony of ants concurrently and asynchronously moves through adjacent states of the problem by building paths on G. The movement is done through application of a stochastic local decision policy that makes use of pheromone trail τ and heuristic value η. Through their movement, ants incrementally construct solutions to the optimization problem. Each ant moves from a state ι to state ψ, corresponding to a more complete partial solution. At each step t, each ant k computes a set of next feasible steps from its current state, and

probabilistically moves to one of these states, according to a probability distribution identified as follows.

For ant k, the probability, $p^k_{\iota\psi}$, of moving from state ι to state ψ depends on the combination of two values:

1. The attractiveness $\eta_{\iota\psi}$ of the move. The attractiveness is defined as the desirability of a move for getting accepted and becoming part of the solution. Attractiveness is computed by a heuristic indicating the *a priori* desirability of that move.

2. The pheromone trail level $\tau_{\iota\psi}$ of the link, indicating how effective it has been in the past to make that particular move. The pheromone concentration indicates an *a posteriori* condition on the desirability of that move. The pheromone trails encode a long-term memory about the whole ant search process that is updated by the ants themselves [66].

Once an ant has constructed a solution, or while the solution is being constructed, the ant evaluates the (partial) solution and deposits pheromone trails on the components or connections used by the ant. This pheromone information later directs the search of the future ants.

The heuristic value represents *a priori* information about the problem instance definition or run-time information provided by a source different from the ants [66]. Generally, η is the cost, or an estimate of the cost, of extending the current state. These values are utilized by the ants' heuristic rule to make probabilistic decisions on how to move on the graph [68].

The exact rules for the probabilistic choice of solution components vary across

different ACO variants. The best known rule is the one of ant system (AS) [66, 163, 164]:

$$p^k_{\iota\psi}(t) = \frac{[\tau_{\iota\psi}(t)]^{\alpha_{ant}}[\eta_{\iota\psi}]^{\beta_{ant}}}{\sum_{l\in N_\iota}[\tau_{\iota\psi}(t)]^{\alpha_{ant}}[\eta_{\iota\psi}]^{\beta_{ant}}} \tag{2.23}$$

where N_ι is the set of neighbors of node ι, $p^k_{\iota\psi}(t)$ is the probability of selecting a link l between nodes ι and ψ for the kth ant, $\tau_{\iota\psi}$ is the pheromone on link l, and $\eta_{\iota\psi}$ is the heuristic value of link l. α_{ant} and β_{ant} represent the influence of pheromone content and heuristic respectively.

Daemon Actions

The daemon actions are optional processes. Daemon actions can be used to implement centralized actions that cannot be performed by single ants. Examples of daemon actions are the activation of a local optimization procedure, or the collection of global information that can be used to decide whether it will be useful to deposit additional pheromone to bias the search process from a non-local perspective [42, 65].

Pheromone Update

The purpose of pheromone update is to increase the pheromone values associated with good solutions, and to decrease those that are associated with bad ones. Usually, this is achieved by decreasing all the pheromone values through pheromone evaporation, and by increasing the pheromone levels associated with a chosen set of good solutions. Pheromone evaporation is a crucial process since it is needed to avoid too rapid convergence of the algorithm on a sub-optimal region. The evapo-

ration process favors exploration of new areas of the search space by implementing a useful form of forgetting.

A number of pheromone updating schemes have been proposed in the literature [66, 161, 235]. One update scheme is to make use of elitist ants. It is important to provide a little description of the approach, since the elitist ant approach is used in this thesis. The elitist ant approach, proposed by Dorigo *et al.* [67], is based on the assumption that the trail of the best tour will direct the search of all the other ants in probability towards a solution composed by some edges of the best tour itself [67]. Other variations of this elitist approach have been applied to optimization problems. For example, an elitist ant is treated as the ant that has found the best solution so far, and the pheromones on links of following ants are updated on the basis of the edges utilized by this elitist ant, as adopted by Gu *et al.* [109]. Alternatively, the elitist ant represents the best solution found in a particular iteration, and only the links of this solution have the pheromone updated. Thus, the elitist ant potentially changes in each iteration (there my be chance that it does not change, but will change more frequently than the global best ant). This approach has been used by Merkle *et al.* [170], Ho *et al.* [117] and Angus [10].

Some Variants of the ACO Meta-heuristic

Following are some of the well-known ACO algorithms.

Ant System

The Ant System (AS) was the first ACO algorithm which was proposed by Dorigo *et al.* and applied to the travelling salesman problem (TSP) [65]. AS has

served as the prototype of many following ACO algorithms with which many other combinatorial problems can be solved successfully [187]. In AS, the probability of moving from node ι to node ψ is found using Equation (8.1). Engelbrecht [77] described that the transition probability used by AS is a balance between pheromone intensity (i.e. history of previous successful moves), $\tau_{\iota\psi}$, and heuristic information (expressing desirability of the move), $\eta_{\iota\psi}$. This efficiently balances the exploration-exploitation trade-off [77]. The best balance between exploration and exploitation is achieved through selection of suitable values of the parameters α_{ant} and β_{ant}. For $\alpha_{ant} = 0$, no pheromone information is used, i.e. previous search experience is neglected, resulting in a stochastic greedy search [77]. If $\beta_{ant} = 0$, the attractiveness (or potential benefit) of moves is neglected [77].

The pheromone is updated using

$$\tau_{\iota\psi}(t+1) = (1-\varrho)\tau_{\iota\psi}(t) + \Delta\tau_{\iota\psi}(t) \tag{2.24}$$

where ϱ is a user-defined coefficient known as evaporation/forgetting constant, and $\Delta\tau_{\iota\psi}$ represents the sum of the contributions of all ants that used the link '$\iota\psi$' to construct their solutions.

Dorigo *et al.* [66] developed three variants of AS, namely, Ant-cycle AS, Ant-density AS, and Ant-quantity AS. The three variants differ in the way $\Delta\tau_{\iota\psi}$ is calculated.

Ant Colony System

The ant colony system (ACS), developed by Dorigo and Gambardella [161], was

the first major improvement in AS. ACS has three major differences than AS [161]: (i) a different state transition rule is used (ii) the global updating rule is applied only to edges which belong to the best ant tour, and (iii) while ants construct a solution, a local pheromone updating rule is applied.

The ACS use the so called *pseudo-random-proportional* rule [94] developed to explicitly balance the exploration and exploitation abilities of the algorithm [77]. For ant k currently located at node ι, selection of the next node ψ to move to is directed by the following rule [77]:

$$\psi = \begin{cases} \arg\max_{l \in N_\iota^k(t)} \{\tau_{\iota l}(t)\eta_{\iota l}^{\beta}(t)\} & \text{if } r \leq r_0 \\ \Psi & \text{if } r > r_0 \end{cases} \tag{2.25}$$

where $r \sim U(0,1)$, and $r_0 \in [0,1]$ is user-defined parameter, $N_\iota^k(t)$ is a set of valid nodes to visit, and $\Psi \in N_\iota^k(t)$ is a node randomly selected according to probability

$$p_{\iota\Psi}^k(t) = \frac{\tau_{\iota\Psi}(t)\eta_{\iota\Psi}^{\beta}(t)}{\sum_{l \in N_\iota^k} \tau_{\iota l}(t)\eta_{\iota l}^{\beta}(t)} \tag{2.26}$$

The parameter r_0 is used to balance exploitation and exploration. A value of $r \leq r_0$ biases the algorithm towards exploitation by favoring the best edge. However, a value of $r > r_0$ favors exploration [77]. Also note that that the transition rule is the same as that of AS when $r > r_0$.

Unlike AS, only the globally best ant (i.e. the ant that constructed the optimum path, $x^+(t)$) is permitted to deposit pheromone on the links of the corresponding best path [77]. Pheromone is updated according to the following global update rule,

$$\tau_{\iota\psi}(t+1) = (1-\varrho_1)\tau_{\iota\psi}(t) + \varrho_1 \Delta\tau_{\iota\psi}(t) \tag{2.27}$$

where

$$\Delta\tau_{\iota\psi}(t) = \begin{cases} \frac{1}{f(x^+(t)} & \text{if } (\iota,\psi) \in x^+(t) \\ 0 & \text{otherwise} \end{cases} \tag{2.28}$$

The ACS global update rule results in more directed search, since ants are encouraged to search in the proximity of the best solution found thus far. This strategy supports exploitation, and is applied after all ants have constructed a solution [77].

Pheromone evaporation in ACS follows an approach slightly different than that of AS. Referring to Equation (2.27), for small values of ϱ_1, the current pheromone concentrations on links evaporate slowly, thus dampening the influence of the best route [77]. On the other hand, for large values of ϱ_1, previous pheromone concentrations evaporate rapidly, but the influence of the best path is emphasized [77].

In addition to global updating rule, a local updating rule is also used in ACS

$$\tau_{\iota\psi}(t+1) = (1-\varrho_2)\tau_{\iota\psi}(t) + \varrho_2\tau_0 \tag{2.29}$$

where $0 < \varrho_2 < 1$ is parameter, and τ_0 is a small constant.

Some applications of ACO

Since its advent, ACO has been applied to a variety of problems. Some important applications are the travelling salesman problem (TSP), which was the first problem used to evaluate the performance of the AS [65]. Using the TSP as a test case, a number of modifications were proposed, such as the ant-cycle within AS [44], the

```
Algorithm ACO_Meta-Heuristic();
    while (termination criterion not satisfied);
            ant_generation_and_activity();
            pheromone_update();
            daemon_actions(); optional
    end while
end Algorithm
```

Figure 2.12: Pseudo-code of the ant colony optimization meta-heuristic

Max-Min ant system [235], the Ant-Q [92], the elite and ranking ant systems [25], and multiple ant colonies [135]. Another important application of ACO was on the quadratic assignment problem (QAP). Several variants of the ACO to deal with QAP were proposed [64, 150, 162, 163, 164, 226, 236]. The ACO was also applied to job shop scheduling [43]. Vehicle routing is another well-known application of ACO algorithms [23, 24]. Other applications of ACO algorithms include the shortest common supersequence [171, 172], graph coloring [45], sequential ordering [81], set covering [52], and logic circuit optimization [37]. With respect to the network design problems, ACO is relatively unexplored. However, ACO meta-heuristic has been applied successfully to topological networks design by Premprayoon *et al.* [207].

ACO and Multi-objective Optimization

ACO algorithms have been adapted to solve multi-objective optimization problems. Alaya *et al.* [5] have differentiated between the multi-objective ACO approaches on the basis of the following three aspects:

1. **Pheromone trails**

   The quantity of pheromone deposited on a link symbolizes the past experience

of the colony with respect to choosing this link. In case of a single objective, this past experience is defined with respect to this objective. However, with multiple objectives, two different strategies may be considered. The first strategy employs a single pheromone structure [15, 99, 106, 165, 168, 263]. In this approach, the amount of pheromone deposited by ants is defined by an aggregation of the multiple objectives. The second strategy is to consider several pheromone structures [11, 23, 59, 60, 93, 123, 258]. In this case, an individual colony of ants is associated with each different objective.

2. **Solutions to reward**

   When updating pheromone trails, it should be decided which ants are allowed to influence pheromone concentrations. One possible way of deciding is to reward solutions that find the best values for each objective in the current iteration [59, 60, 93]. Another possibility is to reward every non-dominated solution of the current iteration. Either all the solutions in the Pareto set may influence pheromone update [15], or only the new non-dominated solutions that enter the set in the current iteration [123].

3. **Definition of heuristic factors**

   As mentioned earlier in this section, when constructing a solution, a candidate at each step is selected on the basis of the transition probability. This probability depends on a pheromone factor and a heuristic factor. The definition of the pheromone factor depends on how the pheromone trails have been defined, as discussed in point (1) above. The heuristic factor can be defined based on two different strategies. The first strategy suggests an aggregation of the mul-

tiple objectives into a single heuristic information matrix [23, 59, 106, 222]. The other strategy considers a separate heuristic matrix for each objective [15, 23, 93, 123, 165]. There is also multi-colony approach [27, 258] where local and global sharing strategies are used.

2.4.7 Particle Swarm Optimization

The particle swarm optimization (PSO) is an optimization heuristic proposed by Kennedy and Eberhart [73, 138]. As with ACO, the PSO is also inspired from nature. The PSO algorithm is based on the sociological behavior associated with bird flocking [138]. The algorithm can be used to solve a variety of continuous and binary optimization problems.

In PSO, a population of potential solutions to the problem under consideration is used to explore the search space [200]. Each individual of the population is called a 'particle'. A particle has an adaptable velocity (step size), according to which the particle moves in the search space. Moreover, each particle has a memory, remembering the best position it has ever visited in the search space [74]. This best position is termed as personal best, or *pbest*. The fitness value associated with the *pbest* position is also stored. Another "best value" that is tracked by the global version of the particle swarm optimizer is the overall best value, and the associated best location, obtained so far by any particle in the population. This location is called the *gbest* particle. Thus, a particle's movement is an aggregated 'acceleration' towards its best previously visited position (the cognitive component) and towards the best individual (the social component) of a topological neighborhood. Since the "acceleration" term was mainly used for particle systems in particle physics [212],

the pioneers of this technique decided to use the term "particle" for each individual, and the name "swarm" for the population, resulting in the name "particle swarm" [138].

The particle swarm optimization algorithm consists of, at each time step, changing the velocity (accelerating) of each particle toward its *pbest* and *gbest* locations in the global version of the PSO. Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward *pbest* and *gbest* locations.

Each particle in the swarm maintains the following information:

- $\mathbf{x_i}$: the *current position* of the particle;
- $\mathbf{v_i}$: the *current velocity* of the particle;
- $\mathbf{y_i}$: the *personal best position* of the particle; and
- $\mathbf{\hat{y}_i}$: the *neighborhood best position* of the particle.

The velocity update step is specified separately for each dimension, $j \in 1...N$, where $v_{i,j}$ represents the jth dimension of the velocity vector associated with the ith particle. The velocity of particle i is updated using

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1,j}(t)[y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)[\hat{y}_j(t) - x_{i,j}(t)] \qquad (2.30)$$

where w is the *inertia weight*, c_1 and c_2 are *acceleration coefficients*, and $r_{1,j}, r_{2,j} \sim U(0,1)$ are two independent random sequences. These random sequences induce a stochastic component in the search process. Apart from $v_{i,j}(t)$, Equation (2.30) has two other main components: the cognitive component, $c_1 r_1(t)[\mathbf{y_i(t)} - \mathbf{x_i}]$, and

Algorithm PSO();
  For each particle $i \in 1, ..., s$ **do**
      Randomly initialize $\mathbf{x_i}$
      Initialize $\mathbf{v_i}$ to zero
      Set $\mathbf{y_i} = \mathbf{x_i}$
  end For
  Repeat
    For each particle $i \in 1, ..., s$ **do**
      Evaluate the fitness of particle i
      Update $\mathbf{y_i}$
      Update $\mathbf{\hat{y}_i}$
      For each dimension $j \in 1, ..., N$ **do**
        Apply velocity update using Equation (2.30)
      end For
      Apply position update using Equation (2.31)
    end For
  Until some convergence criterion is satisfied
end Algorithm

Figure 2.13: Pseudo-code of the basic particle swarm optimization algorithm

the social component, $c_2 r_2(t)[\mathbf{\hat{y}_i}(\mathbf{t}) - \mathbf{x_i}]$. The cognitive component represents the particle's own experience of the best solution found by the particle. The social component represents the belief of the neighborhood regarding the position of best solution in the neighborhood.

The position $\mathbf{x_i}$ of a particle i is updated using

$$\mathbf{x_i}(t+1) = \mathbf{x_i}(t) + \mathbf{v_i}(t+1) \tag{2.31}$$

Figure 2.13 lists pseudo-code of the basic PSO. There are many main groups of PSO algorithms. Based on the neighborhood topology used, two early versions of PSO have been developed [200]: the global best (*gbest*) PSO, and the local best (*lbest*) PSO. These models are briefly discussed below.

gbest PSO

For the global best (*gbest*) PSO, the neighborhood of each particle is the entire swarm. Thus, each particle is attracted to single “best solution” called the global best particle. All particles will converge on a point on the straight line that connects the global best position with the personal best of the particle [244]. It is also important to mention that even if all particles converge to the global best particle, there is no guarantee that the *gbest* is even a local minimum [247]. If *gbest* is not updated regularly, the swarm may converge prematurely [244]. However, one advantage of the *gbest* model is that it offers a faster rate of convergence [74].

lbest PSO

Unlike the *gbest* model, the *lbest* model maintains multiple attractors. A subset of particles, known as the neighborhood, N_i, is defined for each particle, usually based on particle’s index number. However, topological neighborhoods [238] such as *ring* and *Von Neumann* can also be used [137, 141]. For each particle, a neighborhood best position is selected as the best particle in its neighborhood. In the case of ring topology, the neighborhood best is referred to as the local best (*lbest*), and the corresponding algorithm is referred to as the *lbest* PSO. Neighborhoods overlap, which in the end allows convergence to one point. Particles selected to be in N_i have no relationship to each other in the search space domain, because selection of neighbors is based purely on the particle’s index number. This is done for two reasons: it is computationally inexpensive, since no spatial clustering has to be performed, and it helps to promote the spread of information regarding good solutions to all particles, regardless of their current location in search space.

The *gbest* model is a special case of the *lbest* model with the entire swarm as the only neighborhood. Note that the *lbest* model can still prematurely converge due to the same reasons as for *gbest*, but there is smaller probability of becoming trapped in a local minimum [74, 78].

PSO Parameters

The standard PSO algorithm/model consists of several parameters that have an influence on the performance of the algorithm [139]. These include

- **Dimensionality of the particles**

  In some cases, dimensionality is considered an important parameter in determining the hardness of a problem. PSO has been shown to perform very well on a wide variety of hard, high-dimensional benchmark functions such as the De Jong suite and other hard problems including the Schaffer's f6, Griewank, Ackley, Rastrigin, and Rosenbrock functions [8, 138, 228]. Angeline [139] found that PSO actually performs relatively better on higher-dimensional versions of some test functions than on versions of the same functions in fewer dimensions.

- **Number of particles (i.e. swarm size)**

  Swarm size is another important factor in PSO. Increasing population size generally causes increase in computational complexity per iteration, but favors higher diversity, and therefore, may take less iterations to converge [139]. Generally, there is an inverse relationship between the size of the population and the number of iterations required to find the optimum of an objective function [139]. This relationship is more prominent for the *gbest* versions of the

algorithm, with the entire population considered as the neighborhood, than for some *lbest* versions.

- **Inertia weight w**

  The inertia weight w was a modification to the standard PSO, proposed by Shi and Eberhart [228], to control the impact of the previous history of velocities on the current velocity, thus influencing the trade-offs between global (wide-ranging) and local (nearby) exploration abilities of the particles. A larger value of w facilitates exploitation (searching new areas), thus increasing diversity. A smaller value of w tends to facilitate local exploration to fine-tune the current search area.

- **Acceleration coefficients $\mathbf{c}_1$ and $\mathbf{c}_2$**

  The acceleration coefficients, c_1 and c_2, associated with the cognitive and social components play an important role in the convergence ability of the PSO. Varying these parameters has the effect of varying the strength of the pull towards the two bests (i.e. personal best and neighborhood best). Values of $c_1 = c_2 = 0$ means both the cognitive and social components are absent, and particles keep moving at their current speed until they hit a boundary of the search space (assuming no inertia) [77]. With $c_1 > 0$ and $c_2 = 0$, each particle searches for the best position in its neighborhood, and replaces the current best position if the new position is better [77]. However, with $c_2 > 0$ and $c_1 = 0$, the entire swarm is attracted to a single point, $\hat{\mathbf{y}}$. Furthermore, having $c_1 >> c_2$ causes each particle to get attracted to its own personal best position to a very high extent, resulting in excessive wandering. On the other hand,

$c_2 >> c_1$ results in particles getting more strongly attracted to the global best position, thus causing particles to rush prematurely towards optima [77].

Van den Bergh [247] showed that the relation between acceleration coefficients and inertia weight should satisfy the following equation to have guaranteed convergence:

$$\frac{c_1 + c_2}{2} - 1 < w \tag{2.32}$$

- **Velocity clamping $\mathbf{V_{max}}$**

  Since there was no actual mechanism for controlling the velocity of a particle, it was necessary to impose a maximum value, V_{max}, on it [74]. V_{max} restricts the step size, i.e. the amount by which velocity is updated. This upper limit on step sizes prevents individuals from moving too rapidly from one region of the problem space to another, overshooting good regions of the search space. V_{max} proved to be crucial, because large values could result in particles moving past good solutions, while small values could result in insufficient exploration of the search space due to too small step sizes. The value assigned to V_{max} is not arbitrary, and should be optimized for each problem. It is recommended to set V_{max} to a value that is determined by the domain of the variables [139].

Some applications of PSO

The applications of PSO are in diverse fields. One of the first applications of PSO is to train neural networks [72, 78, 243, 245, 246]. The PSO was also applied to variations of the travelling salesman problem [227, 225, 251]. Yoshida *et al.* [265, 266]

applied PSO to power systems. Yuan *et al.* [274] proposed a PSO for multicast routing in sensor networks. Ai-ling *et al.* [3] used PSO algorithm to solve a vehicle routing problem. PSO was applied to the field of antennas by Pérez and Basterrechea [201].

PSO and Multi-objective Optimization

PSO was also adapted to solve MOO problems. Reyes-Sierra and Coello-Coello [213] provided a detailed classification of current MOO approaches for PSO, as discussed below:

1. **Aggregating approaches**

   This category considers approaches that "aggregate" all the objectives of the problem into a single one. In other words, the multi-objective problem is converted into a liner combination of the sub-objectives. PSO aggregation approaches were proposed by Parsopoulos and Vrahatis [199] and Baumgartner *et al.* [17].

2. **Lexicographic ordering**

   Lexicographic ordering (discussed in Section 2.2.3) has also been applied to multi-objective PSO [120, 121].

3. **Sub-awarm approaches**

   Sub-swarm approaches use one swarm for each objective. That is, each swarm optimizes one of the sub-objectives. An information exchange mechanism is used to balance the trade-offs among the different solutions generated for the objectives that were separately optimized [35, 198, 213].

4. **Pareto-based approaches**

   Pareto-based approaches involve "leader selection" techniques based on Pareto dominance. In MOO PSO, the leaders are the personal best positions (local leaders) and neighborhood best positions (global leaders). The basic idea is to select leaders to the particles that are non-dominated with respect to the rest of the swarm [16, 40, 84, 117, 181, 183, 210, 213].

2.5 Conclusion

This chapter provided a brief overview of optimization methods with emphasis on evolutionary and swarm intelligence techniques. Most of the discussed methods are used in this thesis to solve the distributed local area network topology design problem. The problem is modelled as a multi-objective optimization problem using fuzzy logic. A formal definition of this problem is given and discussed in the next chapter.