Design and Analysis of Evolutionary and Swarm Intelligence Techniques for Topology Design of Distributed Local Area Networks

by

Salman A. Khan

Submitted in partial fulfillment of the requirements for the degree Philosophiae Doctor in the Faculty of Engineering, Built Environment, and Information Technology

University of Pretoria

Pretoria

July 2009

Design and Analysis of Evolutionary and Swarm Intelligence Techniques for Topology Design of Distributed Local Area Networks

by
Salman A. Khan

Abstract

Topology design of distributed local area networks (DLANs) can be classified as an NP-hard problem. Intelligent algorithms, such as evolutionary and swarm intelligence techniques, are candidate approaches to address this problem and to produce desirable solutions. DLAN topology design consists of several conflicting objectives such as minimization of cost, minimization of network delay, minimization of the number of hops between two nodes, and maximization of reliability. It is possible to combine these objectives in a single-objective function, provided that the trade-offs among these objectives are adhered to. This thesis proposes a strategy and a new aggregation operator based on fuzzy logic to combine the four objectives in a single-objective function. The thesis also investigates the use of a number of evolutionary algorithms such as stochastic evolution, simulated evolution, and simulated annealing. A number of hybrid variants of the above algorithms are also proposed. Furthermore, the applicability of swarm intelligence techniques such as ant colony optimization and particle swarm optimization to topology design has been investigated. All proposed techniques have been evaluated empirically with respect to their algorithm parameters. Results suggest that simulated annealing produced the best results among all proposed algorithms. In addition, the hybrid variants of simulated annealing, simulated evolution, and stochastic evolution generated better results than their respective basic algorithms. Moreover, a comparison of ant colony optimization and particle swarm optimization shows that the latter generated better

results than the former.

Keywords: Optimization, Local area networks, Fuzzy logic, Simulated annealing, Simulated evolution, Stochastic evolution, Swarm intelligence, Ant colony optimization, Particle swarm optimization, Unified And-Or operator.

Thesis Supervisor: Prof Andries P. Engelbrecht
Department of Computer Science
Degree: Doctor of Philosophy

Dedicated to my beloved parents

Acknowledgements

All praise be to God Almighty, for his limitless blessing and guidance. It is only because of his will and mercy that this thesis was made possible.

I would like to express my profound gratitude and appreciation to my supervisor, Professor Andries P. Engelbrecht, for his guidance, patience, and sincere advice throughout this thesis. I acknowledge his valuable time and constructive criticism. Each moment I spent working with him in this research was enjoyable and exciting.

All my family members, especially my parents, were a constant source of motivation and support. Their love and care carried me through some difficult moments in my life. Their prayers, guidance and inspiration led to this accomplishment. I am also very thankful to my sisters Amber and Sahar for their kind support, and to my wife Sobia for her patience and understanding.

Contents

List of Tables

List of Figures

"For the things we have to learn before we can do them, we learn by doing them."

Aristotle

"I learned this, at least, by my experiment; that if one advances confidently in the direction of his dreams, and endeavors to live the life which he has imagined, he will meet with a success unexpected in common hours."

Henry David Thoreau

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

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]:

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

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:

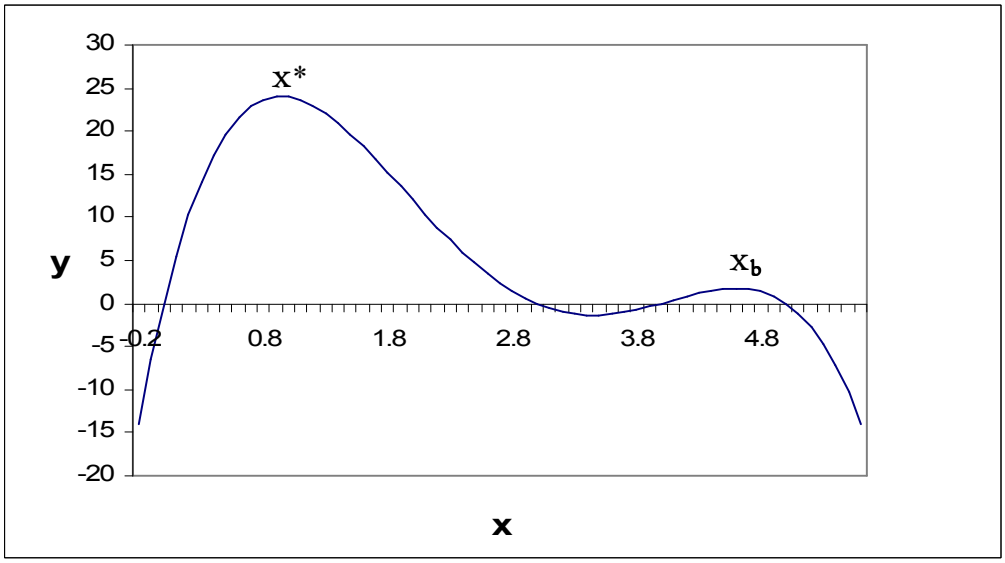


Figure 2.1: Example of global maximum $\mathbf{x}^*$ and local maximum $\mathbf{x_b}$

$$\begin{aligned} &\text{Given } f\colon \Re^n \to \Re \\ &\text{find } \mathbf{x}^* \in \Re \textit{ for which } f(\mathbf{x}^*) \geq f(\mathbf{x}) \;\forall\; \mathbf{x} \in S \subseteq \Re^n \end{aligned} \tag{2.2}$$

$$\begin{aligned} \text{subject to } \; & g_m(\mathbf{x}) \leq 0, \quad m = 1, \cdots, n \\ & h_m(\mathbf{x}) = 0, \quad m = n+1, \cdots, n+n_h \end{aligned}$$

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:

$$\begin{aligned} &\text{Optimize}: \quad \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \cdots, f_K(\mathbf{x})) \\ &\text{subject to} \quad \mathrm{g}_m(\mathbf{x}) \leq 0, \quad m = 1, \cdots, n \\ &\qquad\qquad \mathrm{h}_m(\mathbf{x}) = 0, \quad m = n+1, \cdots, n+n_h \end{aligned} \tag{2.3}$$

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:

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

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

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

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

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],

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

The solution of Equation (2.6) is referred to as $\mathbf{x}_\mathbf{1}^*$. The second problem is then formulated as:

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

The solution of Equation (2.7) is referred to as $\mathbf{x}_\mathbf{2}^*$. 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 p^{th} 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]:

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

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].

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]:

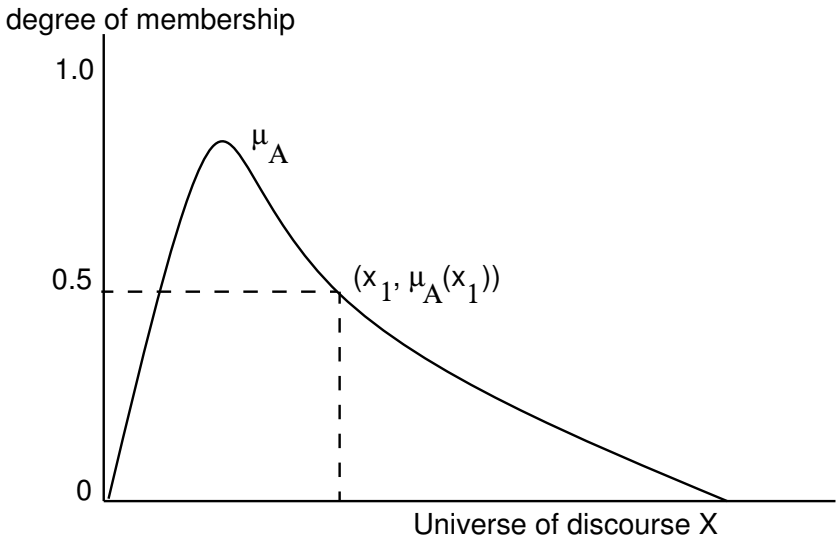


Figure 2.2: Membership function for a fuzzy set A

- 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

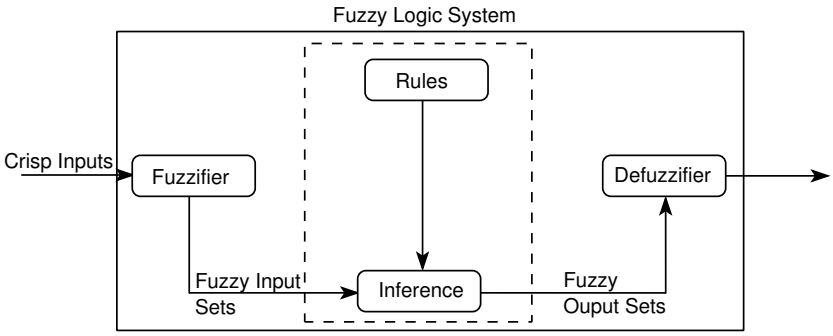


Figure 2.3: Fuzzy logic system

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.

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

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

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

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

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

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

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

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

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

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

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 $A_1(\mathbf{x})$, $A_2(\mathbf{x})$, $\cdots$, $A_K(\mathbf{x})$ are K objectives of a multi-objective problem, and $\mathbf{x}$ is a candidate solution. Accordingly, the corresponding membership functions of $A_1(\mathbf{x})$, $A_2(\mathbf{x})$, $\cdots$, $A_K(\mathbf{x})$ 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].

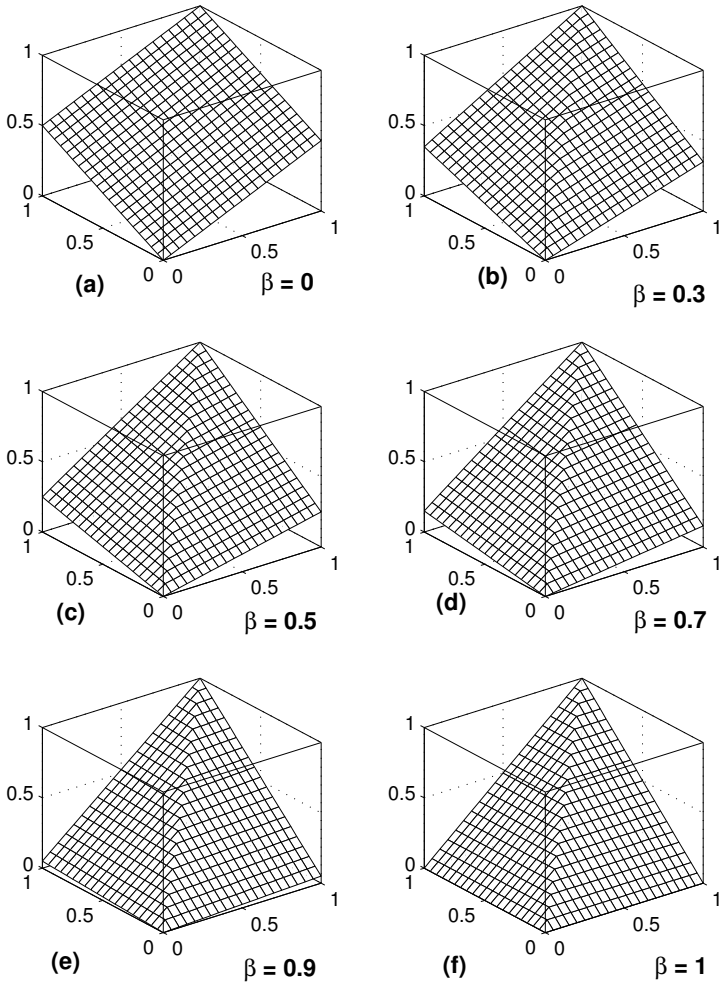


Figure 2.4: Effect of β on OWA-AND function

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].

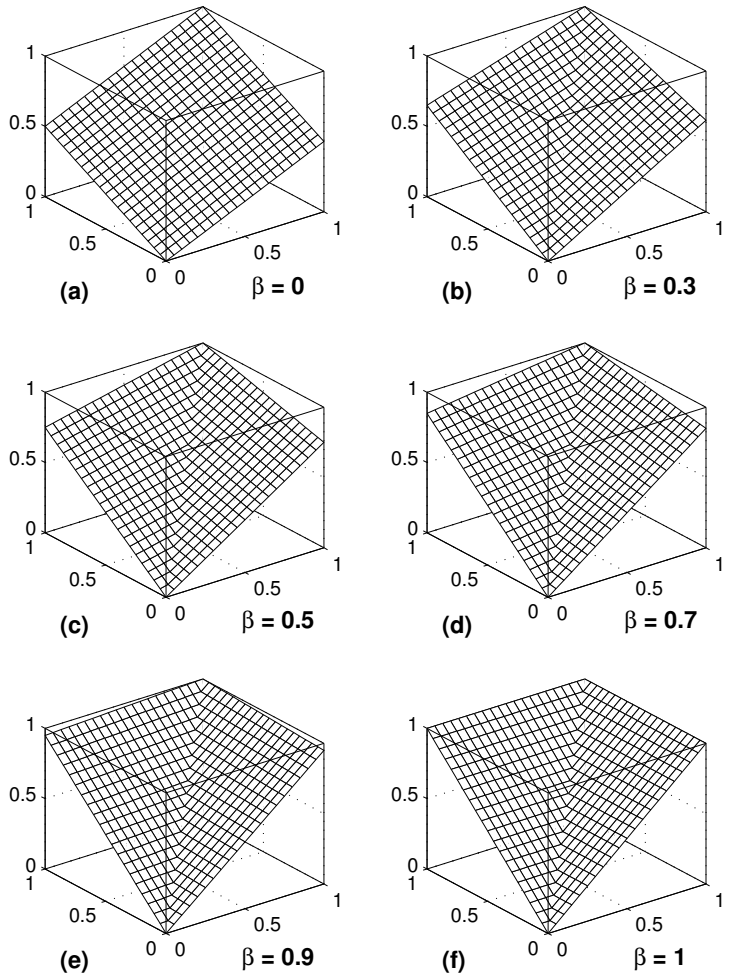


Figure 2.5: Effect of β on OWA-OR function

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
Φ= 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 $(q \in Q)$ **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 $Cost(S^*) <$ **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 k^{th} 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
- $\hat{\mathbf{y}}_\mathbf{i}$: 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 j^{th} dimension of the velocity vector associated with the i^{th} particle. The velocity of particle i is updated using

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

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 $\hat{\mathbf{y}}_\mathbf{i}$
      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, $c_2 r_2(t)[\hat{\mathbf{y}}_\mathbf{i}(\mathbf{t}) - \mathbf{x_i}]$. 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.

Chapter 3

Topology Design of Distributed Local Area Networks

This chapter provides a detailed definition of the problem considered in this thesis. This information is required in order to understand the concepts, terminology, and related work described in subsequent chapters. Firstly, an informal discussion of network topology design and related issues are provided. This is followed by assumptions made for the purposes of this thesis and a formal statement of the distributed local area networks topology design problem. Design objectives and constraints are also discussed. Finally, the fuzzy logic approach for aggregating the individual design objectives is discussed.

3.1 Background

A computer communication network provides communication services to a large number of hosts. Hosts may include mainframe computers, mini systems, worksta-

tions, personal computers, printers, and other peripherals. To interconnect these hosts, network active elements such as routers, switches, and hubs are used. The large number of different hosts and network active elements result in a large number of ways in which hosts and network active elements can be interconnected. Such networks are referred to as internetworks. At an abstract level, a typical computer communication network can be divided into two stages. The first stage is known as the local access network, which allows users access to hosts or local servers. Local access networks are generally designed as *centralized* systems. The second stage consists of a backbone, which is responsible for the delivery of information from source to destination using *switching* elements. The backbone network can be designed as a *distributed* network which relies on switching technology.

In a modern organization, communication services are centered around a distributed local area network, or DLAN. In a DLAN, the backbone interconnects a number of local access networks via routers or layer 3 switches (refer to Figure 3.1). A local access network may be subdivided into smaller sub-groups, called *segments*. Thus, a hierarchically structured network topology is achieved. Such a hierarchical topology comprises a switched backbone that interconnects several local access networks via routers or layer 3 switches, where each local access network is an interconnected collection of segments. Distributed local area networks have several advantages over centralized networks, including

1. Broadcast traffic is confined to a single local access network. This prevents broadcast storms from sweeping across the entire network.

2. Highest network availability and lowest latency are ensured.

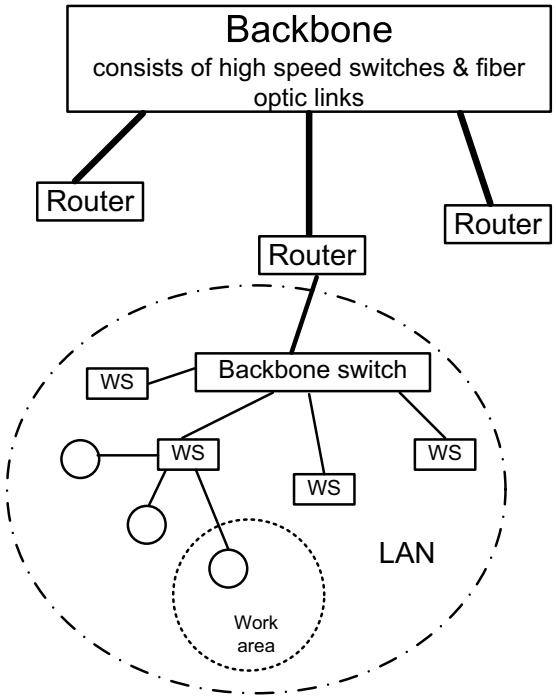


Figure 3.1: A typical distributed local area network (WS represents a workgroup switch)

3. Users are provided with the most appropriate connectivity.

4. Cost of administration is reduced. Equipment movement and changes are carried out more systematically. Moreover, diagnosis and troubleshooting of network problems are easier.

The objective of topological design of computer networks is to achieve a certain performance level through improvement in various parameters such as network delay, cost, reliability, and number of hops between communicating nodes. An over-dimensioned network is easy to design, while it is always difficult to design a cost-optimized network. The prime factors contributing to this difficulty are the size, the constraints, and obviously the cost parameters. Topology optimization usually

involves a tradeoff between performance and cost. For example, management would like to consider financial aspects while designing a network, and would like the monetary costs to be minimized. On the other hand, a user would be more interested in a service where communication delay is minimized while network reliability is maximized. Thus, the notion of optimality becomes vague in the presence of various cost parameters and constraints. A rational approach to high-quality network design is to search for a solution that possesses a set of desirable attributes and does not violate well-established design principles. For example,

- there should be a physical path between any two nodes,
- the number of hops between any two nodes should remain within a defined upper limit, and
- utilization levels of links should always be below a given threshold.

The network topology design problem has received considerable attention by network designers and analysts. Extensive research has been done to develop efficient optimization techniques for this complex optimization problem [57, 58, 75, 98, 110, 111, 152, 204]. Topology design has been categorized as an NP-hard problem [75, 79, 98]. Similarly, topological design of DLAN can be considered as an equally complex problem with a huge search space. For a network with n nodes, there exist as many as $2^{(\frac{n(n-1)}{2}-1)}$ distinct topologies. Even for $n = 10$, there exist more than 10^{13} potential solutions. It is therefore clear that an exhaustive search is not desirable due to its huge computational cost. Rather, heuristic approximation methods are used to search for an optimal topology design. Heuristic methods produce good

feasible solutions in a reasonable amount of time and focus the search on feasible topologies of desirable characteristics.

3.2 Assumptions and Problem Statement

This section describes the necessary assumptions adopted in this thesis, and also provides a formal statement of the DLAN topology design problem and design constraints.

3.2.1 Assumptions

For the purposes of DLAN topology design, the following are assumed:

- A tree topology is considered for network design.
- A "node" refers to a LAN (i.e. a local site). The node represents a layer 2, or above, networking device (i.e. switch, router, or gateway) connecting the LAN to another backbone node.
- The "root" node is a switch acting as a collapsed backbone with given required interfaces.
- Each link is bidirectional.
- The reliability of each link is known.
- Nodes are fault-free. Only links are susceptible to failure.
- Only fiber optic cable is used between two LANs.

- The maximum utilization of any link should remain within a desired threshold (e.g., 60%).
- Fast Ethernet is implemented on the backbone.
- The "capacity" of a network device is equal to the number of ports available on it. These ports are used to connect users to a network device.
- Distances between local sites (nodes) are known. The location of a node can be represented by its Cartesian coordinates with respect to some reference point.
- The internal topology of each LAN (i.e. a local site) is known a priori. That is, each LAN is assumed to already exist.

3.2.2 Problem Statement

The DLAN topology design problem can be stated as follows [268, 269, 270, 271, 272]:

"With the assumption that the internal topology of each LAN is already designed and known, find a quality feasible tree topology under a given set of design objectives and constraints. This tree topology will interconnect all nodes (LANs) in the network, thus forming a backbone topology of a DLAN."

The term "feasible topology" in the above statement refers to a solution that satisfies all design principles and constraints. The term "quality topology" refers to a solution that optimizes the design objectives. In this thesis, the quality of a topology is evaluated on the basis of four design objectives: monetary cost, average network

delay per packet (network latency), maximum number of hops between any source-destination pair, and network reliability. The search targets feasible topologies which minimizes the first three objectives and maximizes the fourth objective.

3.3 Design Objectives and Constraints

This section describes the design objectives and constraints considered in this thesis. All objectives are summarized in Section 3.3.1, while the constraints are discussed in Section 3.3.2.

3.3.1 Design objectives

As mentioned earlier, four design objectives are considered. These objectives are:

Monetary cost

The aim is to find a topology with low cost, while satisfying the design constraints (discussed in Section 3.3.2 below). Since the number of network devices would be the same in any topology, the only entity that affects the monetary cost is the cost of cables. Cost is expressed as

$$cost = length \times c_{cable} \tag{3.1}$$

where *length* represents the total length of cable, and c_{cable} represents the cost per unit of the cable used.

Average Network Delay

The second objective is to minimize the average network delay incurred on a packet during transmission from a source node to a destination node.

To estimate the average network delay, the aggregate behavior of a link and network device is modelled by an M/M/1 queue [75]. If a link connects local sites i and j, then the delay per bit due to the network device feeding this link is $B_{i,j} = b_{i,j}/\omega$, where $b_{i,j}$ is the delay per packet, and ω is the average packet size in bits. If γ_{ij} is the total traffic through the network device between local sites i and j, then the average packet delay due to all network devices is:

$$D_{nd} = \frac{1}{\gamma} \sum_{i=1}^{d} \sum_{j=1}^{d} \gamma_{ij} B_{ij} \tag{3.2}$$

where d is the total number of networking devices in the network, and γ is the total traffic in the network, representing the summation of all γ_{ij}. The total average network delay is the summation of delays over all links and network devices, given as [75]:

$$D = \frac{1}{\gamma} \sum_{i=1}^{L} \frac{\lambda_i}{\lambda_{max,i} - \lambda_i} + \frac{1}{\gamma} \sum_{i=1}^{d} \sum_{j=1}^{d} \gamma_{ij} B_{ij} \tag{3.3}$$

where L is the number of links in the topology, λ_i is traffic on link i in bits per second (bps), and $\lambda_{max,i}$ is the capacity of link i in bps.

Maximum number of hops between any source-destination pair

The maximum number of hops between any source-destination pair needs to be minimized. A hop is counted as the packet crosses a network device. The reason for taking number of hops as an optimization objective is due to the restrictions imposed by the routing information protocol (RIP). RIP uses hop count to measure the distance between the source and a destination node. RIP implements a limit on the number of hops encountered in the path from a source to a destination to prevent routing loops from continuing indefinitely [233]. The maximum number of hops allowed in a path is 15. If the hop count exceeds this number, then the destination is considered unreachable [233].

Network reliability

Network reliability should be maximized. Network reliability can be defined as the probability of occurrence of an event in which each node communicates with every other node in the network [2]. For the purposes of this thesis, the topology is a tree. Thus, the reliability of such a topology is the product of the reliabilities of all links present in that particular topology [12, 136]. Mathematically,

$$R_s = \prod_{i=1}^{L} R_i \tag{3.4}$$

where R_i is the reliability of link i, and R_s is reliability of the network.

3.3.2 Constraints

Three types of constraints are considered in this thesis, namely:

1. The maximum number of nodes attached to network device i must not exceed the capacity, p_i, of that device. That is,

$$\sum_{j=1}^{n} t_{ij} < p_i, \qquad i = 1, 2, ..., n, \qquad \forall i \neq j \tag{3.5}$$

   where n is number of nodes in the network, and t_{ij} represents a connection between device (i.e. a node) i and device j.

2. Link bandwidth is limited. Therefore, a good network will employ "reasonably" utilized links, since links with high utilization levels experience delays, congestion, and packet loss. The traffic flow on any link i therefore must be limited by a threshold value, $\lambda_{max,i}$, as follows:

$$\lambda_i < \lambda_{max,i}, \quad i = 1, 2, ..., L \tag{3.6}$$

3. The last set of constraints are specified by the designer, and is used to enforce certain design guidelines and principles:

   (a) Certain nodes must be leaf/terminal nodes. For example, hubs should generally be placed as leaf nodes.

   (b) Certain nodes must be interior nodes of the tree, for example, nodes designated as switches or routers.

(c) Certain nodes cannot be directly connected to the backbone. For example, hubs should not be directly connected to the backbone (i.e. the root node).

3.4 Fuzzy Logic Approach to the DLAN Topology Design Problem

Topology design of LANs is a complex process which requires simultaneous optimization of a number of design objectives. Important objectives include monetary cost, maximum number of hops between a source-destination pair, network average delay, and network reliability. None of these objectives on their own gives sufficient information to decide the quality of the network topology. It is also the case that some of these objectives, such as the delay, can only be approximated. The complexity of the problem is further amplified by the conflicting nature of some of these objectives. Thus, a trade-off between the conflicting objectives is required. Fuzzy logic comfortably provides a mechanism to handle imprecise information since the logic provides a rigorous algebra for dealing with imprecise information. Furthermore, the logic is a convenient method of combining conflicting objectives and expert human knowledge.

The rest of this section describes how fuzzy logic is employed in combining the four conflicting objectives into a single overall objective. This overall objective estimates the quality of a solution in terms of membership of a given topology to the fuzzy set of quality topologies. For simplicity, the following discussion uses the terms cost, delay, hops, and reliability. The goal is to find a high quality solution,

represented by a linguistic variable 'good topology'. A good topology consists of low cost, small number of hops, low delay, and high reliability, as summarized in Figure 3.2.

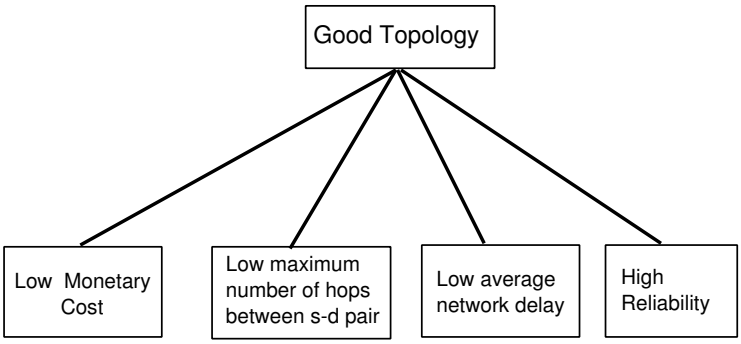


Figure 3.2: Basic components of a good topology

To evaluate the quality of the overall solution using fuzzy logic, the quality of individual objectives needs to be evaluated first through membership functions. Once this is done, fuzzy logic rules can be used to assess the quality of solutions with respect to the individual objectives. For the DLAN topology design problem, a solution should satisfy the four objectives mentioned above. Each of these objectives is evaluated using a membership function. Thus, a membership function needs to be defined for each objective. This process is described below.

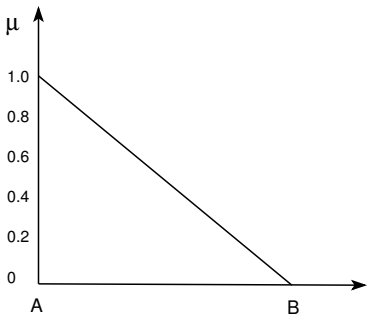


Figure 3.3: Membership function of the objective to be optimized

To find the membership function of cost, two extreme values, the minimum and maximum, are determined first. These values can be found mathematically or from prior knowledge. Figure 3.3 depicts the membership function of the objective to be optimized (cost in this case). In this figure, the two values shown as 'A' and 'B' refer to the minimum cost 'MinC' and the maximum cost 'MaxC'. The membership value for cost of solution x, $\mu_c(x)$, is computed as

$$\mu_c(x) = \begin{cases} 1 & \text{if } Cost(x) \leq MinC \\ \frac{MaxC - Cost(x)}{MaxC - MinC} & \text{if } MinC < Cost(x) \leq MaxC \\ 0 & \text{if } Cost(x) > MaxC \end{cases} \tag{3.7}$$

where the term $Cost(x)$ represents the cost of the solution. The membership function for delay, $\mu_d(x)$, can be defined in a similar way. The two extreme values of delay are 'MinD' and 'MaxD' for minimum and maximum values respectively. In Figure 3.3, 'MinD' corresponds to 'A' and 'MaxD' corresponds to 'B'. The membership value of delay is determined as

$$\mu_d(x) = \begin{cases} 1 & \text{if } Delay(x) \leq MinD \\ \frac{MaxD - Delay(x)}{MaxD - MinD} & \text{if } MinD < Delay(x) \leq MaxD \\ 0 & \text{if } Delay(x) > MaxD \end{cases} \tag{3.8}$$

where the term $Delay(x)$ represents the average delay of the solution. The membership function for number of hops, $\mu_h(x)$, is also illustrated by Figure 3.3, where the minimum (MinH) and maximum (MaxH) values correspond to 'A' and 'B' respectively. The membership value for number of hops is determined as

$$\mu_h(x) = \begin{cases} 1 & \text{if } Hops(x) \leq MinH \\ \frac{MaxH - Hops(x)}{MaxH - MinH} & \text{if } MinH < Hops(x) \leq MaxH \\ 0 & \text{if } Hops(x) > MaxH \end{cases} \tag{3.9}$$

Finally, the membership function for reliability, $\mu_r(x)$, can be determined by finding the maximum and the minimum bounds for the solution reliability. In Figure 3.3, the minimum (MinR) and maximum (MaxR) values correspond to 'B' and 'A' respectively. The membership value for reliability is determined as

$$\mu_r(x) = \begin{cases} 1 & \text{if } Rel(x) \geq MaxR \\ \frac{MaxR - Rel(x)}{MaxR - MinR} & \text{if } MinR < Rel(x) \leq MaxR \\ 0 & \text{if } Rel(x) < MinR \end{cases} \tag{3.10}$$

After obtaining the membership functions of the four objectives, the next phase is to combine these functions into an overall function (i.e. a single objective function) of "good topology" using fuzzy logic. A good topology is one that is characterized by a low cost, low delay, small number of hops, and high reliability. In fuzzy logic, this can be stated by the following fuzzy rule:

Rule 1: **IF** a solution X has *low cost* AND *low delay* AND *low hops*
AND *high reliability* **THEN** it is a *good topology*

The expressions "low cost", "low delay", "low hops", "high reliability", and "good topology" are linguistic values, each defining a fuzzy subset of solutions. For

example, "high reliability" is the fuzzy subset of topologies of high reliabilities. Each fuzzy subset is defined by a membership function μ. The membership function returns a value in the interval [0,1] which describes the degree of satisfaction with the particular objective criterion. The above fuzzy rule can be mathematically represented using the OWA-AND operator:

$$\mu(x)^O = \beta \; min\{\mu_1(x), \mu_2(x), \mu_3(x), \mu_4(x)\} + (1-\beta)\frac{1}{4}\sum_{i=1}^{4}\mu_i(x) \tag{3.11}$$

In Equation (3.11), $\mu(x)^O$ is the membership value for solution x in the fuzzy set *good topology* using the OWA-AND operator. Also in the same equation, μ_i for $i = \{1,2,3,4\}$ represents the membership values of solution x in the fuzzy sets *low cost*, *low delay*, *low hops*, and *high reliability* respectively. The solution which results in the maximum value for Equation (3.11) is reported as the best solution found. However, it is also possible to get a set of best solutions having equal membership values (i.e. Pareto optimal solutions), in which case any one of such solutions is taken as the best solution.

3.5 Characteristics of Test Cases

The test cases used in this thesis have been used in other literature [271, 272, 268, 270, 269]. These test cases were used to evaluate the performance of all algorithms proposed in this thesis. The test cases represent networks consisting of local sites. Traffic generated by each local site for these test cases was collected from real sites, and costs of the network cables were collected from vendors. Other characteristics,

Table 3.1: Network characteristics assumed for experiments.

Parameter	Characteristic
Cost of fiber optic cable	$ 5 per meter
Delay per bit due to networking device	$250\mu sec.$
Maximum traffic on a link allowed	60
Average packet size	500 bytes
Type of networking device	Router, switch, or hub
Number of ports on a networking device	4, 8, or 12

such as the number of ports on a network device and its type are listed in Table 3.1. Five test cases were used for experimental work. For these test cases, the number of local sites ranged between 15 (denoted by *n15*) and 50 (denoted by *n50*). Table 3.2 summarizes the features of these test cases.

Table 3.2: Characteristics of test cases used in experiments. MinC is in US$, MinD is in milliseconds, and traffic is in Mbps.

Test Case	# of Local Sites	MinC	MinD	MaxR	Traffic
n15	15	4640	2.14296	0.868746	24.63
n25	25	5120	2.15059	0.785678	74.12
n33	33	8158	2.15444	0.72498	117.81
n40	40	9646	2.08757	0.675729	144.76
n50	50	11616	2.08965	0.611117	164.12

3.5.1 Upper and Lower Bounds for Objective Values

The extreme values of the objectives can be found as given below. Some of these values, such as MinC, MaxC, MinD, MinH, and MaxR, can be pre-calculated, while others such as MaxD, MaxH, and MinR are computed during the initialization step of the optimization algorithm.

For the cost objective, the minimum value, 'MinC', is found by using the Esau-

Williams algorithm [80], with all the constraints completely relaxed. This guarantees that the minimum possible cost of the topology is obtained. The value of 'MinC' for the five test cases is given in Table 3.2. The maximum value for cost, 'MaxC', is taken to be the cost generated by the initialization step of the optimization algorithm. Once the two extreme values are available, the membership value, μ_c, of cost is computed using Equation (3.7).

The minimum value for the delay objective, 'MinD', is found by connecting all the nodes directly to the root node, ignoring all constraints. The value of 'MinD' for the five test cases is given in Table 3.2. The maximum value of delay, 'MaxD', is taken to be the delay generated by the initialization step. Once these values are obtained, the delay membership value, μ_d, is calculated using Equation (3.8).

For the number of hops objective, the minimum value, 'MinH', is taken to be 1 hop. The maximum value, 'MaxH', is taken to be the maximum number of hops between any source-destination pair generated by the initialization step. The membership value of hops, μ_h, is computed using Equation (3.9).

Finally, for the reliability objective, the maximum reliability, 'MaxR', is found using the Esau-Williams algorithm [80], with all constraints relaxed. This guarantees the maximum possible reliability that could be attained. The value of 'MaxR' for the five test cases is given in Table 3.2. The minimum reliability, 'MinR', is taken to be the reliability of the initial solution. The membership value for reliability, μ_r, is found using Equation (3.10).

3.6 Conclusion

This chapter defined and discussed the problem of topology design of distributed local area networks formulated as a multi-objective optimization problem. Necessary background information along with important concepts, design objectives, and problem constraints were discussed in sufficient detail. Moreover, aggregation of individual objectives into a combined single fuzzy function using the ordered weighted averaging operator was also covered. In addition, this chapter provided characteristics of test cases used to evaluate the performance of each proposed algorithm, along with the upper and lower bounds of the objective values. The next chapter discusses a new fuzzy aggregating operator proposed in this thesis.

Chapter 4

The Unified AND-OR Fuzzy Operator

A new fuzzy operator is proposed and discussed in detail in this chapter. This new operator is shown to have mathematical properties similar to that of the ordered weighted averaging operator. A new preference handling approach is also proposed. To illustrate the effectiveness of the proposed operator and the preference scheme, empirical analysis is performed using some examples, and a comparison with the OWA operator is done.

4.1 Definition of the Unified AND-OR Operator

This section focusses on a new operator proposed in this thesis, namely the *unified AND-OR* operator or UAO. As will be seen, this operator uses a single equation (unlike the two separate equations for AND and OR of Yager's OWA operator), yet it is capable of behaving either as the OWA-AND or the OWA-OR operator. The

behavior is controlled by a variable $\nu \geq 0$, whose value decides whether the function will behave as AND or OR. The operator is defined as

$$f(a,b) = \frac{ab + \nu \max\{a,b\}}{\nu + \max\{a,b\}} = \begin{cases} \mathrm{I}_\star = \mu_{A \cup B}(x) & \text{if } \nu > 1 \\ \mathrm{I}^* = \mu_{A \cap B}(x) & \text{if } \nu < 1 \end{cases} \tag{4.1}$$

where a represents the membership value of μ_A (i.e. $a = \mu_A$), b represents the membership value of μ_B (i.e. $b = \mu_B$), and $f(a,b)$ represents the value of the overall objective function (i.e. $f(a,b) = \mu_{AB}$). I^* represents the AND operation using the UAO operator, and $I_\star$ denotes the OR operation using the UAO operator.

In Equation (4.1), parameter ν is used to orient the equation such that the UAO behaves either as the AND or the OR operator. With $\nu < 1$, the UAO behaves as the OWA-AND operator. A value of $\nu = 0$ gives the pure-AND behavior. On the other hand, a value of $\nu > 1$ shifts the behavior of the UAO towards the OWA-OR operator. As $\nu \longrightarrow \infty$, the ORing becomes more 'rigid'. The UAO behaves as pure-OR when $\nu = \infty$. However, experimentation with different values of ν suggests that the behavior of the UAO is quite similar to that of the OWA-OR with $1 \leq \nu \leq 100$. Figure 4.1 depicts instances of the UAO with different values of ν. As illustrated in Figure 4.1(a), a value of $\nu = 0$ for UAO gives exactly the same behavior as that of OWA-AND with β =1 (see Figure 2.4(f)). Similarly, Figure 4.1(b) and (c) depict almost the same behavior illustrated in Figure 2.4. When ν = 1 (Figure 4.1(d)), the UAO operator behaves almost the same as the OWA-AND with $\beta = 0$ (or equivalently, as OWA-OR with $\beta = 0$).

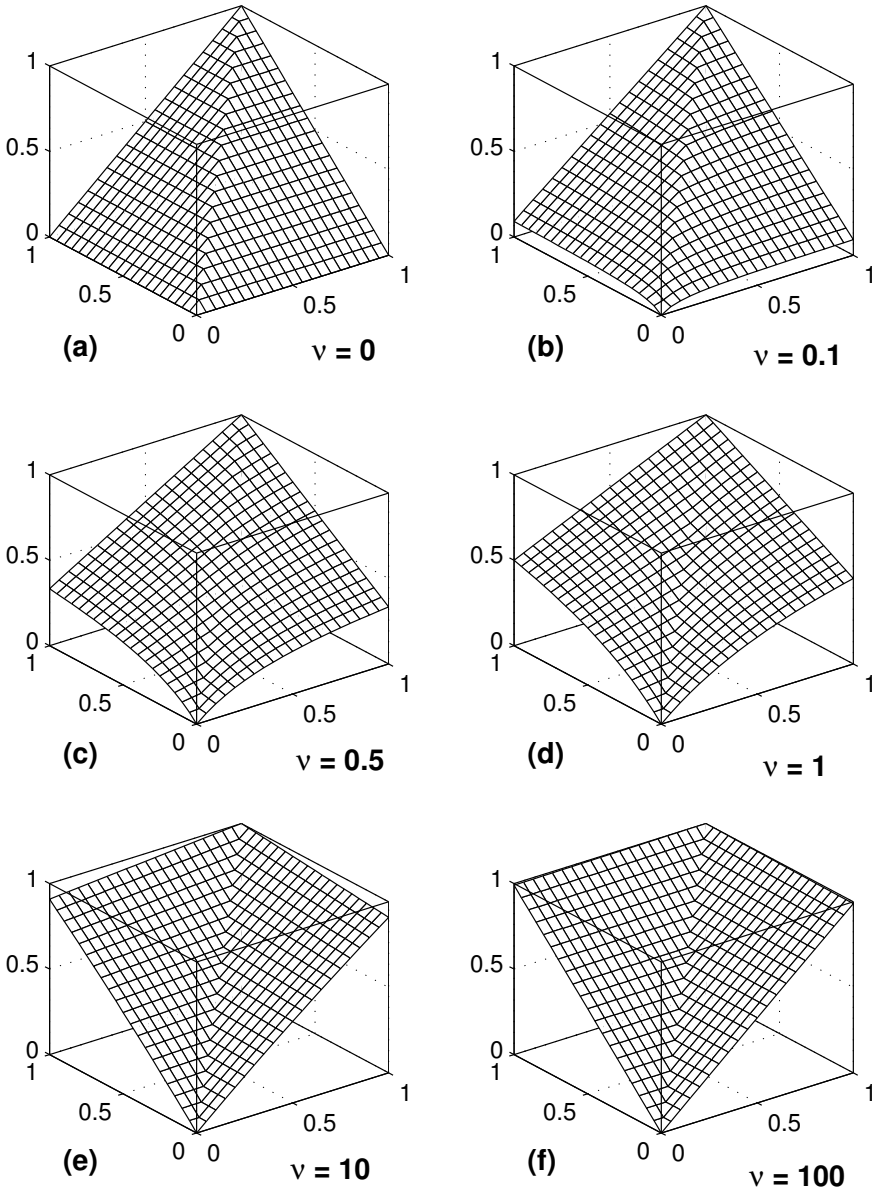


Figure 4.1: Effect of ν on Unified AND-OR operator

4.2 Mathematical Properties

This section proves that the UAO operator also satisfies the properties of monotonicity, symmetry, and idempotency, similar to OWA. Assume a value, $\Delta a > 0$, which represents the change in μ_A (i.e. $\Delta a = \Delta\mu_A$).

1. **Monotonicity:** The claim is that the UAO operator is monotonic. Thus, it is necessary to prove that

$$f(a,b) \geq f(\ddot{a},\ddot{b}) \text{ if } a \geq \ddot{a} \text{ and } b \geq \ddot{b}$$

To prove this, several cases need to be considered. The proof of each case is given below:

Case 1: $a + \Delta a < b \Rightarrow a < b$. It is to be proven that $f(a + \Delta a, b) > f(a, b)$.

Proof: From Equation (4.1),

$$f(a+\Delta a,b)\Big|_{a+\Delta a<b,a<b} = \frac{(a+\Delta a)b+\nu b}{\nu+b}$$

$$= \frac{ab+\nu b}{\nu+b} + \frac{\Delta ab}{\nu+b} \tag{4.2}$$

$$f(a,b)\Big|_{a<b} = \frac{ab+\nu b}{\nu+b} \tag{4.3}$$

Comparing Equations (4.2) and (4.3), and since $\frac{\Delta ab}{\nu+b} > 0$, it is concluded that $f(a + \Delta a, b) > f(a, b)$.

Case 2: $a + \Delta a > b$ and $a > b$. It is to be proven that $f(a + \Delta a, b) > f(a, b)$.

Proof: Since $a > b$, take $a = b + k_2$ where $k_2 > 0$. Also $a + \Delta a > b$, which implies that $a + \Delta a = k_1 + b$, where $k_1 > 0$. We find $\Delta a = k_1 - k_2$. From

Equation (4.1),

$$f(a+\Delta a,b)\Big|_{a+\Delta a>b,\ a>b} = \frac{(a+\Delta a)b+\nu(a+\Delta a)}{\nu+(a+\Delta a)} = \frac{(a+\Delta a)(b+\nu)}{\nu+(a+\Delta a)} \quad (4.4)$$

$$f(a,b)\Big|_{a>b} = \frac{ab+\nu a}{\nu+a} = \frac{a(b+\nu)}{\nu+a} \quad (4.5)$$

Substituting $\Delta a = k_1 - k_2$ and $a = b + k_2$ in Equations (4.4) and (4.5) respectively, yields:

$$f(a+\Delta a,b)\Big|_{a+\Delta a>b,\ a>b} = \frac{(b+k_1)(b+\nu)}{\nu+b+k_1} \quad (4.6)$$

$$f(a,b)\Big|_{a>b} = \frac{(b+k_2)(b+\nu)}{\nu+b+k_2} \quad (4.7)$$

Now, compare Equations (4.6) and (4.7) as follows:

$$f(a+\Delta a,b)\Big|_{a+\Delta a>b,\ a>b} \overset{?}{\gtrless} f(a,b)\Big|_{a>b} \quad (4.8)$$

That is,

$$L.H.S. \overset{?}{\gtrless} R.H.S.$$

$$\Rightarrow \frac{(b+k_1)(b+\nu)}{\nu+b+k_1} \overset{?}{\gtrless} \frac{(b+k_2)(b+\nu)}{\nu+b+k_2} \quad (4.9)$$

$$\therefore \frac{b+k_1}{\nu+b+k_1} \overset{?}{\gtrless} \frac{b+k_2}{\nu+b+k_2} \quad (4.10)$$

Cross multiplication yields:

$$(b+k_1)(\nu+b+k_2) \overset{?}{\gtrless} (b+k_2)(\nu+b+k_1) \quad (4.11)$$

Solving Equation (4.11) results in:

$$\nu(b + k_1) \overset{?}{\gtrless} \nu a \tag{4.12}$$

$$\therefore b + k_1 \overset{?}{\gtrless} a \tag{4.13}$$

Since $a + \Delta a = k_1 + b \Rightarrow a = k_1 + b - \Delta a$, Equation (4.13) becomes

$$(b + k_1) \overset{?}{\gtrless} (b + k_1 - \Delta a) \tag{4.14}$$

Simplifying Equation (4.14) yields:

$$\Delta a > 0 \tag{4.15}$$

From Equation (4.15), since L.H.S. $>$ R.H.S., it is concluded that $f(a + \Delta a, b) > f(a, b)$.

Case 3: $a + \Delta a > b$ and $a < b$. It is to be proven that $f(a + \Delta a, b) > f(a, b)$.

Proof: Since $a < b$, assume $a + k_2 = b$ where $k_2 > 0$. Also $a + \Delta a > b$, which implies that $a + \Delta a = k_1 + b$, where $k_1 > 0$. Therefore $\Delta a = k_1 + k_2$. From Equation (4.1),

$$f(a + \Delta a, b)\Big|_{a+\Delta a>b,\ a<b} = \frac{(a + \Delta a)b + \nu(a + \Delta a)}{\nu + (a + \Delta a)} = \frac{(a + \Delta a)(b + \nu)}{\nu + a + \Delta a} \tag{4.16}$$

$$f(a, b)\Big|_{a>b} = \frac{ab + \nu b}{\nu + b} = \frac{b(a + \nu)}{\nu + b} \tag{4.17}$$

Substituting $\Delta a + a = b + k_1$ and $a = b - k_2$ in Equations (4.16) and (4.17) respectively, yields

$$f(a+\Delta a, b)\Big|_{a+\Delta a>b,\ a<b} = \frac{(b+k_1)(b+\nu)}{\nu+b+k_1} \tag{4.18}$$

$$f(a,b)\Big|_{a>b} = \frac{b[(b-k_2)+\nu]}{\nu+b} \tag{4.19}$$

Now, compare Equations (4.18) and (4.19) as follows:

$$f(a+\Delta a, b)\Big|_{a+\Delta a>b,\ a<b} \overset{?}{\gtrless} f(a,b)\Big|_{a>b} \tag{4.20}$$

That is,

$$L.H.S. \overset{?}{\gtrless} R.H.S.$$

$$\Rightarrow \frac{(b+k_1)(b+\nu)}{\nu+b+k_1} \overset{?}{\gtrless} \frac{b[(b-k_2)+\nu]}{\nu+b} \tag{4.21}$$

Cross multiplication yields:

$$(b+k_1)(\nu+b)^2 \overset{?}{\gtrless} b(b-k_2+\nu)(\nu+b+k_1) \tag{4.22}$$

Taking $b+\nu = b'$ reduces Equation (4.22) to:

$$(b+k_1)(b')^2 \overset{?}{\gtrless} b(b'-k_2)(b'+k_1) \tag{4.23}$$

$$\therefore (b')^2k_1 + bk_2(b'+k_1) \overset{?}{\gtrless} bb'k_1 \tag{4.24}$$

$$\therefore ((b')^2 + bk_2)k_1 + b'bk_2 \overset{?}{\gtrless} b'bk_1. \tag{4.25}$$

Substituting $b' = b + \nu$ back into Equation (4.25) gives:

$$((b+\nu)^2 + bk_2)k_1 + (b+\nu)bk_2 \overset{?}{\geqslant} (b+\nu)bk_1 \tag{4.26}$$

$$\therefore b^2k_1 + \nu^2k_1 + 2\nu bk_1 + bk_1k_2 + b^2k_2 + \nu bk_2 \overset{?}{\geqslant} b^2k_1 + \nu bk_1 \tag{4.27}$$

$$\Rightarrow \nu^2k_1 + \nu bk_1 + bk_1k_2 + b^2k_2 + \nu bk_2 > 0 \tag{4.28}$$

Since L.H.S. $>$ R.H.S. for all b and for $\nu \geq 0$, it is concluded from Equation (4.28) that $f(a + \Delta a, b) > f(a, b)$.

The above proofs considered the case $f(a + \Delta a, b) > f(a, b)$ with different possible scenarios. The case where $f(a, b + \Delta b) > f(a, b)$ can be proven in a similar manner. Moreover, the proof can be extended to more than two variables. Thus, it can be claimed that the UAO operator is monotonic for any number of variables.

2. **Symmetry (generalized commutativity):** It is obvious from the structure of the UAO operator that the order of arguments does not matter. Thus the operator is symmetric.

3. **Idempotency:** It is to be proven that

$$f(a, a) = \frac{a \cdot a + \nu \ max(a, a)}{\nu + max(a, a)} = a \tag{4.29}$$

Proof: From Equation (4.29),

$$f(a, a) = \frac{a^2 + \nu \ a}{\nu + a} = \frac{a(a+\nu)}{\nu + a} = a \tag{4.30}$$

□

4.3 Fuzzy Rules for Topology Design

The four objectives can be combined in a number of ways using fuzzy operators to generate rules with a single consequent, i.e. "good topology". One possibility is the extreme where it is required that all the objectives are simultaneously optimized, thus implying the AND operation among all objectives, as mentioned earlier in Chapter 3. Another extreme is where optimization of any one objective would suffice. A number of combinations exist between these extremes. These cases, as well as the application of the proposed UAO operator on them, are discussed next.

4.3.1 Case 1: Simultaneous Optimization of All Four Objectives

For the extreme case where all objectives have to be optimized, the fuzzy rule is:

- R1: IF cost is *low* AND delay is *low* AND hops is *low* AND reliability is *high* THEN the topology is *good*.

The corresponding mathematical representation using the UAO operator is:

$$\mu_s = I^*(\mu_c, \mu_d, \mu_h, \mu_r) \tag{4.31}$$

4.3.2 Case 2: Simultaneous Optimization of Three Objectives

A number of cases can be defined when any three of the four objectives need to be considered. Some of these cases and their UAO representations are:

- R2a: IF cost is *low* AND delay is *low* AND (hops is *low* OR reliability is *high*) THEN the topology is *good*.

$$\mu_s = I^*(\mu_c, \mu_d, I_\star(\mu_h, \mu_r)) \tag{4.32}$$

- R2b: IF cost is *low* AND (delay is *low* OR hops is *low*) AND reliability is *high* THEN the topology is *good*.

$$\mu_s = I^*(\mu_c, \mu_r, I_\star(\mu_d, \mu_h)) \tag{4.33}$$

- R2c: IF cost is *low* AND hops is *low* AND (delay is *low* OR reliability is *high*) THEN the topology is *good*.

$$\mu_s = I^*(\mu_c, \mu_h, I_\star(\mu_d, \mu_r)) \tag{4.34}$$

- R2d: IF (cost is *low* OR reliability is *high*) AND delay is *low* AND hops is *low* THEN the topology is *good*.

$$\mu_s = I^*(\mu_d, \mu_h, I_\star(\mu_c, \mu_r)) \tag{4.35}$$

4.3.3 Case 3: Simultaneous Optimization of Two Objectives

When only two objectives need to be optimized, a number of different cases can be defined. Some of these cases and their UAO representations are given below.

- R3a: IF cost is *low* AND (delay is *low* OR hops is *low* OR reliability is *high*)

THEN the topology is *good*.

$$\mu_s = I^*(\mu_c, I_\star(\mu_d, \mu_h, \mu_r)) \tag{4.36}$$

- R3b: IF delay is *low* AND (cost is *low* OR hops is *low* OR reliability is *high*) THEN the topology is *good*.

$$\mu_s = I^*(\mu_d, I_\star(\mu_c, \mu_h, \mu_r)) \tag{4.37}$$

- R3c: IF (cost is *low* AND delay is *low*) OR (hops is *low* AND reliability is *high*) THEN the topology is *good*.

$$\mu_s = I_\star(I^*(\mu_c, \mu_d), I^*(\mu_h, \mu_r)) \tag{4.38}$$

- R3d: IF (cost is *low* AND reliability is *high*) OR (delay is *low* AND of hops is *low*) THEN the topology is *good*.

$$\mu_s = I_\star(I^*(\mu_c, \mu_r), I^*(\mu_d, \mu_h)) \tag{4.39}$$

- R3e: IF (cost is *low* OR delay is *low*) AND (hops is *low* OR reliability is *high*) THEN the topology is *good*.

$$\mu_s = I^*(I_\star(\mu_c, \mu_d), I_\star(\mu_h, \mu_r)) \tag{4.40}$$

- R3f: IF (cost is *low* OR hops is *low*) AND (delay is *low* OR reliability is *high*)

THEN the topology is *good*.

$$\mu_s = I^*(I_\star(\mu_c, \mu_h), I_\star(\mu_d, \mu_r)) \tag{4.41}$$

4.3.4 Case 4: Optimization of Any One Objective

For the extreme where any one of the four objectives is optimized, the fuzzy rule is given as:

- R4: IF cost is *low* OR delay is *low* OR hops is *low* OR reliability is *high* THEN the topology is *good*.

The corresponding mathematical representation using the UAO operator is:

$$\mu_s = I_\star(\mu_c, \mu_d, \mu_h, \mu_r) \tag{4.42}$$

4.4 Preferences and UAO

Many of the fuzzy rules mentioned above treat the objectives equally (i.e. giving no preference to any objective). For example, in rule R1, all the objectives get equal preference. This type of situation limits the accuracy in decision-making. To enhance the precision in decision-making, an additional set of rules is necessary. This set can be used to give preference to one criterion or another to emphasize or de-emphasize certain objective(s) involved in the decision-making process.

As an example, consider the set of rules given in Section 4.3.2. In all the rules, the three objectives are equally weighted. If the designer wants to emphasize cost in

all cases, he/she has no way of doing so. However, with the availability of preference rules, it will be easier to give more significance to cost.

To formulate the preference rules, preference terms need to be defined. These terms are associated with the main linguistic terms. For the fuzzy rules defined above, the linguistic terms "low", "high", and "good" have been used. The literature [18, 131, 132] has reported some approaches to find preference terms and preference rules based on membership functions. These approaches map a fuzzy preference relation P to a fuzzy membership function μ_P in the range [0,1] as follows:

$$\mu_P(\mathrm{s}_i, s_j) = \begin{cases} 1 & \text{if } s_i \text{ is definitely preferred to } s_j \\ c \in (0.5, 1) & \text{if } \mathrm{s}_i \text{ is slightly preferred to } s_j \\ 0.5 & \text{if there is no preference (i.e., indifference)} \\ d \in (0, 0.5) & \text{if } \mathrm{s}_j \text{ is slightly preferred to } s_i \\ 0 & \text{if } \mathrm{s}_j \text{ is definitely preferred to } s_i \end{cases}$$

The above representation is impractical for decision-makers since the degree of preference for a certain objective is not precise [166]. For example, the term "slightly" represents indefinite numbers of the numeric preference values which are within the range (0.5, 1) or (0.0, 0.5) [166]. To overcome this problem, it is suggested that the numerical representation be extended by using a wider variety of linguistic terms which includes seven linguistic terms. Thus, the above set of preferences is extended as follows:

$$\mu_{Pm}(\mathrm{s}_i, s_j) = \begin{cases} 1 & \text{if } s_i \text{ is definitely preferred to } s_j \\ c \in (0.917, 1) & \text{if } s_i \text{ is strongly preferred to } s_j \\ d \in (0.834, 0.917) & \text{if } \mathrm{s}_i \text{ is highly preferred to } s_j \\ e \in (0.751, 0.834) & \text{if } \mathrm{s}_i \text{ is considerably preferred to } s_j \\ f \in (0.668, 751) & \text{if } \mathrm{s}_i \text{ is moderately preferred to } s_j \\ g \in (0.585, 0.668) & \text{if } \mathrm{s}_i \text{ is slightly preferred to } s_j \\ h \in (0.5, 0.585) & \text{if } \mathrm{s}_i \text{ is mildly preferred to } s_j \\ 0.5 & \text{if there is no preference (i.e., indifference)} \\ i \in (0.417, 0.5) & \text{if } \mathrm{s}_j \text{ is mildly preferred to } s_i \\ j \in (0.337, 0.417) & \text{if } \mathrm{s}_j \text{ is slightly preferred to } s_i \\ k \in (0.254, 0.337) & \text{if } \mathrm{s}_j \text{ is moderately preferred to } s_i \\ l \in (0.171, 0.254) & \text{if } \mathrm{s}_j \text{ is considerably preferred to } s_i \\ m \in (0.088, 0.171) & \text{if } \mathrm{s}_j \text{ is highly preferred to } s_i \\ n \in (0, 0.088) & \text{if } s_j \text{ is strongly preferred to } s_i \\ 0 & \text{if } s_j \text{ is definitely preferred to } s_i \end{cases}$$

Notice that the preference values are reflective-reciprocal at the point of 'indifference'. The evaluation values can be seen as consisting of two groups: 1) s_i preferred to s_j, and 2) s_j preferred to s_i. The value corresponding to 'indifference' may belong to both groups. Since the two groups are similar, and each group includes seven preference terms (including the 'indifference'), the above representation of preference values can be curtailed to a representation of only seven values, instead of fourteen. Thus, it can be said that $c \iff n$, $d \iff m$, and so on, where "$\iff$" means that the two preference terms are equivalent. The interpretation of the "two terms being

equivalent" is that if the first objective is preferred to the second, then the second objective is not preferred to the first. For example, if we say that s_i is *strongly preferred* to s_j, then it is equivalent to saying that s_j is *strongly not preferred* to s_i, etc. The seven-point evaluation is also advocated by Miller's [174] observation that the human mind can deal with around seven items at a time.

The proposed set of preferences can be seen as a subset of the approach presented by Cvetković *et al.* [47]. The approach in [47] suggests a more comprehensive scheme for the use of preferences in multi-objective optimization. However, the scheme in [47] seems more suitable for situations where the number of objectives is high (the authors have applied their approach to a multi-objective problem with thirteen objectives). Their approach involved more complex steps than are proposed in this thesis.

To continue, preference terms proposed in this thesis can be used to define a number of preference rules. These preference rules are divided into several categories, as described below.

4.4.1 Preference rules involving all four objectives:

Examples of preference rules that contain all four objectives are:

- PR1a: Cost is strongly preferred over the other three objectives
- PR1b: Delay is highly preferred over the other three objectives
- PR1c: Reliability is strongly preferred over the other three objectives
- PR1d: Cost is slightly preferred over the other three objectives

4.4.2 Preference rules involving three objectives:

It is also possible to develop preference rules which include three objectives, for example,

- PR2a: Cost is strongly preferred over hops and reliability
- PR2b: Delay is highly preferred over reliability and cost
- PR2c: Reliability is slightly preferred over hops and delay
- PR2d: Cost is slightly preferred over reliability and hops

4.4.3 Preference rules involving two objectives:

Examples of preference rules involving two objectives are:

- PR3a: Cost is strongly preferred over hops
- PR3b: Delay is mildly preferred over reliability
- PR3c: Reliability is considerably preferred over hops
- PR3d: Hops is slightly preferred over cost
- PR3e: Delay is highly preferred over reliability
- PR3f: Cost is mildly preferred over delay

4.4.4 Combining the main rules with preference rules

Once the main rules and the preference rules have been defined, the next step is to combine them together. This may lead to a number of scenarios, for example:

1. **Ex1**: If rule R1 is used at the first level, and *Cost* is emphasized over the other three objectives with PR1a at the second level, then the value of the objective function using the UAO operator is computed by the following formula:

$$\mu_s = I^*(0.917\mu_c, 0.088\mu_d, 0.088\mu_h, 0.088\mu_r)$$

2. **Ex2**: When rule R1 is used at the first level, and the emphasis is given to *reliability* using PR1c on the second level, then the value of the objective function using the UAO operator is computed by the following formula:

$$\mu_s = I^*(0.088\mu_c, 0.088\mu_d, 0.088\mu_h, 0.917\mu_r)$$

3. **Ex3**: If rule R1 is used at the first level and PR3c is used at the second level, thus giving emphasis to *reliability* over *number of hops*, then the value of the objective function can be computed as follows:

$$\mu_s = I^*((\mu_c, \mu_d, 0.25\mu_h), 0.75\mu_r)$$

4. **Ex4**: When rule R2d is used at the first level and PR2c is used at the second level, thus emphasizing *reliability* over *number of hops*, the value of the objective function can be computed as follows:

$$\mu_s = I^*(0.35\mu_d, 0.35\mu_h, I_\star(\mu_c, 0.65\mu_r))$$

5. **Ex5**: If rule R1 is used at the first level and both PR3b and PR3d are used at the second level, the emphasis is given to *delay* over *reliability* and to *number of hops* over *cost*. In this case the value of the objective function is computed by the following formula:

$$\mu_s = I^*(0.40\mu_c, 0.55\mu_d, 0.60\mu_h, 0.45\mu_r)$$

6. **Ex6**: If rule R3c is used at the first level and both PR3c and PR3f are used at the second level, the emphasis is given to *cost* over *delay* and to *reliability* over *number of hops*. In this case the value of the objective function is computed by the following formula:

$$\mu_s = I_\star(I^*(0.55\mu_c, 0.45\mu_d), I^*(0.20\mu_h, 0.80\mu_r))$$

7. **Ex7**: If rule R1 is used at the first level, PR2a is used at the second level, and both PR3b and PR3c are used at the third level, the emphasis is given to *cost* over *number of hops* and *reliability*. Then, the preference is given to *delay* over *reliability* and to *reliability* over *number of hops*. In this case the value of the objective function is computed by the following formula:

$$\mu_s = I^*(0.95\mu_c, 0.55\mu_d, 0.05 \times 0.20\mu_h, 0.05 \times 0.80 \times 0.45\mu_r)$$

The examples above provide different scenarios of using main rules with preference rules. Examples 1 to 4 suggest using rules at two levels. On the first level, the main rule is used, whereas at the second level one preference rule is used. An exten-

sion of this two level case is observed in examples 5 and 6, where a main rule is used at the first level, and two preference rules are used at the second level. Similarly, example 7 illustrates how rules at three levels might be used. In this example, it is observed that a main rule is used at the first level, followed by a preference rule involving three objectives at the second level, and two preference rules at the third level with each involving two objectives. Apart from the aforementioned scenarios, there exist many other possibilities where multi-level rules are possible and multiple preference rules could be used at a certain level.

4.5 Application of UAO to Topology Design

A multi-objective SimE algorithm for network topology design was proposed in [271, 268]. This algorithm was engineered to optimize three design objectives: cost, delay, and number of hops. The main focus of the algorithm was to incorporate fuzzy logic in the **allocation** phase to compute the overall goodness (or fitness) of the solution.

For the purposes of this chapter, the above SimE algorithm has been adopted to evaluate the performance of UAO and OWA. This evaluation is done by using rules Ex1 and Ex2 presented in Section 4.4.4. Although SimE was used for this experiment, any stochastic optimization algorithm, such as a genetic algorithm, simulated annealing, stochastic evolution, ant colony optimization, or particle swarm optimization, can be used instead.

The main focus of the fuzzy SimE algorithm in [271] and [268] was to incorporate fuzzy logic in the allocation phase to compute the overall goodness (or fitness) of the

solution. This overall goodness was calculated using a fuzzy rule similar to rule R1 (refer to Section 4.3.1). The only difference between the rule in [271] and [268], and R1 is that the latter also includes reliability as an optimization factor along with the other three objectives. The overall goodness is used to compare the quality of two solutions. Results are presented in the following section.

4.6 Empirical Results and Discussion

This section discusses empirical results obtained by applying the UAO and OWA operators to SimE using rules Ex1 and Ex2 given in Section 4.4.4. The two operators were applied to the allocation phase of the SimE algorithm, as described in detail in Chapter 6.

The five test cases as described in Chapter 3 were used for performance evaluation. As mentioned earlier, SimE uses a problem dependent bias parameter, B. An appropriate selection of bias is important for convergence of SimE to near-optimal or optimal set of solutions. Thus, it is essential to find the appropriate bias value. Tables 4.1 to 4.4 provide these values, which were obtained after running several tests with different bias values ranging from 0.0 to 0.4 for each test case. Also, once the appropriate bias value was found, thirty runs were executed for each test case for each of the two operators (i.e. UAO and OWA). The average of these runs is also reported in Tables 4.1 to 4.4 for each case. Moreover, through trial runs it was found that the algorithm converges within 4000 iterations. Each run was therefore executed for 4000 iterations.

4.6.1 Application of UAO and OWA to Ex1

In Ex1 of Section 4.4.4, the rule requires optimization of all four objectives, with strong preference given to cost. Tables 4.1 and 4.2 summarize the results obtained after application of UAO and OWA to SimE using rule Ex1. These results reflect the percentage improvement of the final solution (average of thirty runs) with respect to the initial solution. Simulations were run using values of $\beta = 0.5$ and $\nu = 0.5$ for OWA and UAO respectively. It is observed from these tables that both OWA and UAO demonstrated a noticeable improvement in cost for all test cases, as validated by the t-test for statistical significance. These improvements range from 25.46% to 32.42% for UAO and from 25.26% to 32.57% for OWA. The average cost improvement for UAO was 29.17 %, while OWA showed an improvement of 28.38%. Thus, as far as cost is concerned, both operators performed almost equally well, with OWA having, in general, a slightly higher standard deviation than UAO. However, although strong preference was given to cost, there was a remarkable improvement in reliability, with improvements of 4.53% to 77.74% for UAO and 13.16% to 86.45% for OWA. In general, the improvement in reliability was also statistically significant (as validated by a t-test), with the exception of *n15* when UAO was used. One possible argument for this behavior is that the search space could be constricted such that no further improvement in cost is possible, irrespective of the emphasis placed on the cost objective. However, reliability might have a tendency to improve remarkably in most parts of the search space, with the improvement being more significant in some regions. Thus, for the reliability objective, OWA performed better than UAO with almost the same level of standard deviation for most of the test cases, with the exception of *n50*.

Table 4.1: Results for UAO for Ex1. N= number of local sites in the network, B=Bias, C=Cost, D=Delay, H=Hops, R=Reliability, Avg= Average percentage improvement of the five test cases. Statistically significant improvements are in italics.

Case	N	B	Percentage improvement				Standard deviations			
			C	D	H	R	C	D	H	R
n15	15	0.1	*25.46*	6.67	-6.31	4.53	5.24	11.09	18.56	11.13
n25	25	0.3	*30.85*	-7.60	*-37.20*	*40.14*	7.19	6.16	17.71	15.43
n33	33	0.3	*31.50*	-8.52	*-33.70*	*67.42*	4.38	13.43	15.62	14.30
n40	40	0.2	*32.42*	7.68	*-26.55*	*68.33*	3.82	52.68	26.49	15.54
n50	50	0.3	*25.62*	-17.96	*-30.01*	*77.74*	6.37	28.74	19.16	17.63
Avg			29.17	-3.95	-26.75	51.63				

As far as delay and number of hops are concerned, it should be kept in mind that rule Ex1 requires optimization of all four objectives, and that the objectives are conflicting in nature. Therefore, it is also noticed in Tables 4.1 and 4.2 that improvement in cost (and reliability) is achieved at the price of deterioration in number of hops and delay objectives. However, it is apparent from the tables that, on average, the percentage of deterioration in the delay and number of hops objectives was less with UAO than that of OWA (i.e. -3.95% for UAO versus -24.38% for OWA for delay, and -26.75% for UAO compared with -28.76% for OWA for number of hops). As far as standard deviations are concerned, OWA showed relatively higher values than UAO for the delay, specifically for *n33* and *n50*. For number of hops, the deviations were comparable for both UAO and OWA. Collectively, it can be fairly claimed that UAO showed a better overall performance than OWA.

Table 4.2: Results for OWA for Ex1. N= number of local sites in the network, B=Bias, C=Cost, D=Delay, H=Hops, R=Reliability, Avg= Average percentage improvement of the five test cases. Statistically significant improvements are in italics.

Case	N	B	Percentage improvement				Standard deviations			
			C	D	H	R	C	D	H	R
n15	15	0.4	*25.26*	-3.39	*-6.42*	*13.16*	9.25	7.67	16.45	31.02
n25	25	0.3	*26.55*	-7.91	*-33.43*	*54.02*	7.06	8.70	30.70	11.76
n33	33	0.2	*32.57*	-15.94	*-29.02*	*72.73*	5.69	46.72	24.43	12.18
n40	40	0.0	*29.27*	*-29.67*	*-36.91*	*76.26*	5.09	53.79	15.53	15.28
n50	50	0.1	*28.22*	*-64.97*	*-38.02*	*86.45*	9.03	67.36	10.96	6.06
Avg			28.38	-24.38	-28.76	60.52				

4.6.2 Application of UAO and OWA to Ex2

Rule Ex2 of Section 4.4.4 has a structure similar to that of rule Ex1, since Ex2 also requires the optimization of all four objectives. However, for rule Ex2, strong preference is given to reliability. The results of the application of UAO and OWA to SimE for Ex2 are given in Tables 4.3 and 4.4. Values of $\beta = 0.5$ and $\nu = 0.5$ were used for OWA and UAO respectively. The tables depict the expected behavior of UAO and OWA when reliability was given a strong preference over the other three objectives. For both UAO and OWA, a significant improvement was observed for almost all test cases, with the exception of test case *n15* while UAO was used. For example, the improvement in reliability for UAO ranges from 78.71% (for *n25*) to 93.51% (for *n50*). These improvements were also statistically significant. Only one case (*n15*) deviated from this trend, achieving an improvement of only 7.48%. A similar behavior was observed for OWA, where a statistically significant improvement ranging from 17.85% to 93.56% was achieved for test cases *n15* to *n50* respectively. On average, both UAO and OWA showed the same level of improvement in reliability, with

69.89% for UAO and 72.24% for OWA. The deviations in both cases were almost of the same magnitude. As for the cost objective, the obtained results suggest that both UAO and OWA were able to attain almost the same level of improvement for all test cases. On average, UAO reduced the cost by 11.85% compared to 11.74% by OWA. The standard deviations were also comparable for the cost objective.

For the delay and number of hops objectives, the tables show that, on average, both objectives deteriorated for almost all test cases. However, in general, the level of deterioration for the number of hops objective was not statistically significant. The reason for the deterioration in the delay and number of hops objectives is the same as for Ex1, i.e. the four objectives are conflicting in nature, and improvement in some objectives is achieved at the cost of deterioration in the others. However, average deterioration in delay for UAO (-9.2%) is less than that in OWA (-21.81%). Similarly, for the number of hops objective, the deterioration for UAO (-16.42%) is also less than that of OWA (-28.39%). However, for the hops objective, OWA generally showed slightly higher deviations than UAO. Overall, it could be claimed that UAO showed a better performance than OWA for Ex2.

The analysis of results presented above for Ex1 and Ex2 shows that both UAO and OWA effectively handled the multi-objective aspects of the DLAN topology design problem in presence of preferences. On the one hand, there was a case where strong preference was given to the cost objective, while on the other, the situation required strong preference for reliability. The two examples were specifically chosen to reflect the view that preferences diversified the search into different solution subspaces.

Table 4.3: Results for UAO for Ex2. N= number of local sites in the network, B=Bias, C=Cost, D=Delay, H=Hops, R=Reliability, Avg= Average percentage improvement of the five test cases. Statistically significant improvements are in italics.

Case	N	B	Percentage improvement				Standard deviations			
			C	D	H	R	C	D	H	R
n15	15	0.1	6.88	2.55	*-9.98*	7.48	8.97	6.97	14.70	18.66
n25	25	0.3	*15.54*	-5.78	*-23.33*	*78.71*	11.68	10.61	15.23	6.58
n33	33	0.3	*14.56*	17.90	*-6.64*	*79.35*	4.75	36.74	12.48	7.85
n40	40	0.2	*10.88*	-9.10	*-20.85*	*90.39*	9.59	59.51	17.66	3.29
n50	50	0.3	*11.38*	*-51.60*	*-21.32*	*93.51*	3.17	15.00	16.40	3.76
Avg			11.85	-9.20	-16.42	69.89				

4.7 Conclusions

This chapter presented a new fuzzy operator named the unified And-Or (UAO) operator. The operator was shown mathematically to be similar to the OWA operator. The proposed UAO operator was able to aggregate the four design objectives discussed earlier. Moreover, a new 7-point preference scheme was also proposed to handle preferences between the objectives and facilitate the decision-making process. The performance of UAO along with the preference scheme was evaluated by their application to a simulated evolution algorithm (discussed in detail in Chapter 6) using some test examples. An empirical comparison of UAO was also done with the OWA operator. Results suggest that the UAO operator performed better than the OWA operator. Moreover, the proposed fuzzy rules and 7-point preference rules scheme allowed the integration of individual objectives into a single objective function with a good degree of precise decision-making.

The next chapter proposes and discusses a fuzzy stochastic evolution algorithm for the multi-objective DLAN topology design problem. The effectiveness of the

Table 4.4: Results for OWA for Ex2. N= number of local sites in the network, B=Bias, C=Cost, D=Delay, H=Hops, R=Reliability, Avg= Average percentage improvement of the five test cases. Statistically significant improvements are in italics.

Case	N	B	Percentage improvement				Standard deviations			
			C	D	H	R	C	D	H	R
n15	15	0.4	5.19	-11.88	-20.12	*17.85*	19.86	24.22	26.59	19.12
n25	25	0.3	*15.26*	-5.75	*-24.74*	*72.61*	8.90	17.05	26.64	9.67
n33	33	0.2	*11.78*	-5.56	*-29.93*	*88.24*	5.91	32.21	22.98	3.72
n40	40	0.0	*11.92*	-40.06	*-30.60*	*88.94*	6.48	65.37	14.62	5.72
n50	50	0.1	*14.54*	-45.78	*-36.56*	*93.56*	6.93	48.88	20.20	2.87
Avg			11.74	-21.81	-28.39	72.24				

proposed algorithm is evaluated through empirical study.

Chapter 5

Fuzzy Stochastic Evolution Algorithm for DLAN Topology Design

This chapter proposes a new fuzzy stochastic evolution algorithm (FStocE), specifically for the multi-objective DLAN topology design problem. A variant of the fuzzy stochastic algorithm is also proposed, namely, TFStocE, which incorporates characteristics of tabu search. The effect of the tabu list size on the quality of solutions is investigated. Furthermore, an empirical comparison of the FStocE and TFStocE is also done. This comparison is done using both the OWA and the UAO operator. In addition, a method to dynamically assign an important parameter of FStocE, namely, R_c, is also proposed and analyzed.

5.1 Fuzzy Stochastic Evolution

Chapter 2 discussed the StocE algorithm. The same algorithm is applicable to the DLAN topology design problem, but with some modifications to the cost computation. As described below, this cost computation is done using fuzzy logic.

A valid initial solution (i.e. a solution that satisfies the constraints) is randomly generated. The *PERTURB* function alters the existing solution by making a number of moves, where a move involves removing a link between two nodes in the current solution (i.e. topology) and introducing a new link between these nodes. Selection of links for removal as well as for placement is done randomly. However, insertion of a new link is done under the constraint that the newly placed link must include one of the two nodes from which the previous link was removed. Removing a link divides the topology into two disjoint topologies, as depicted in Figure 5.1. Then, for this removed link, another link has to be introduced in the topology such that the complete tree is restored. There are many possibilities of introducing a new link. As an example, consider the removal of a link between two nodes P and Q in Figure 5.1. Figure 5.2 illustrates links that result in a complete tree to be formed. Note that these links include node Q, which formed part of the removed link (alternatively node P, instead of Q could also be chosen to place a new link). It is important to mention that for each removed link, only one other link is tried. If this results in a valid topology, violating no constraints, the link is made permanent. Otherwise the old link is replaced.

Each iteration of the proposed FStocE algorithm makes ten moves, which results in a new solution. The "overall" cost of this new solution is compared to the cost

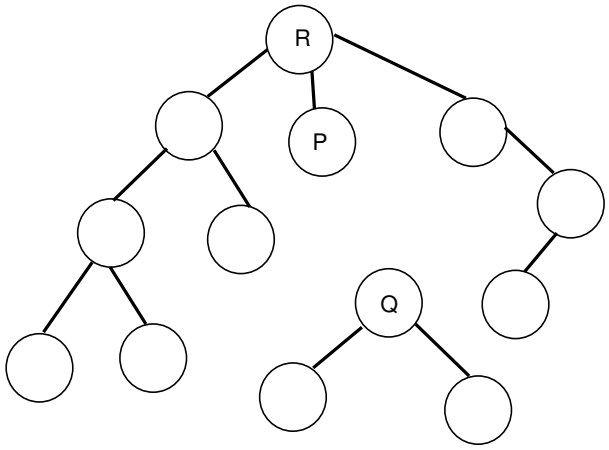


Figure 5.1: Two disjoint trees containing nodes P and Q

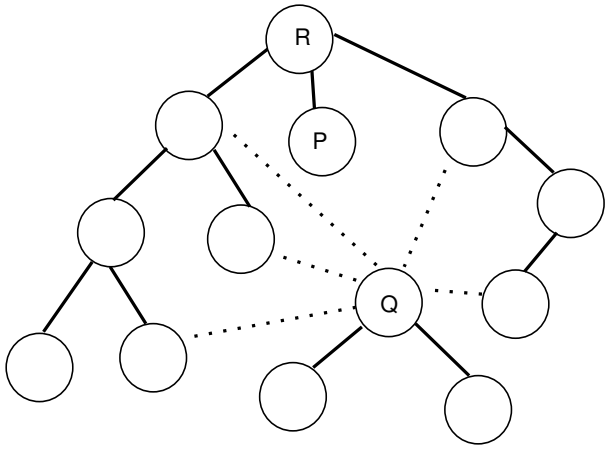


Figure 5.2: Candidate moves (illustrated with dotted lines) that can replace the removed link between P and Q

of the previous solution and the gain is calculated as described in Figure 2.8. This overall cost is computed using the fuzzy rule (Rule 1 in Section 3.4) based on either Equation (3.11) or Equation (4.1).

The FStocE algorithm differs from the standard StocE algorithm in that FStocE assumes a maximization problem, whereas StocE assumes a minimization problem. In StocE, the objective is to minimize cost of the solution, whereas in the FStocE algorithm, the objective is to maximize the overall goodness of the solution based on Equation (3.11) or Equation (4.1). The FStocE algorithm is summarized in Figure 5.3. The algorithm is run iteratively and the solution that gives the maximum value of fuzzy cost function using Equation (3.11) (or Equation (4.1)) is taken as the best solution.

5.2 Tabu Stochastic Evolution

This section proposes a new hybrid fuzzy iterative search technique, namely, tabu stochastic evolution (TFStocE), which introduces features of tabu search in the *PERTURB* function. A move in TFStocE consists of removing a randomly selected link from the current solution and introducing a new feasible link in the solution. This newly accepted link is saved in the *tabu list*. Thus, the *attribute* is the link itself. If the link that had been made tabu produces a higher membership value than the current link in the membership function "good topology", then the aspiration criterion overrides the tabu status of the link, making the link permanent. This strategy prevents the algorithm from repetitively removing the same link and replacing it with a link of equal or worse goodness.

$Stochastic\_Evolution(Z_0, p_0, R_c)$
NOTATION
Z_0= Initial solution
ρ= Counter
p= Control parameter
p_o= Initial value of p
R_c= Stopping criterion parameter
$Goodcur$ = Goodness of current solution Z
$GoodBest$ = Goodness of best solution
$Goodpre$ = Goodness of previous solution
Begin
ZBest = $Z = Z_0$;
GoodBest = Goodcur = $Goodness(Z)$;
$p = p_0$;
$\rho = 0$;
Repeat
Goodpre = Goodcur;
$Z = PERTURB(Z, p)$; /* perform a search in the neighborhood of Z */
Goodcur = $Goodness(Z)$;
$UPDATE(p$, Goodpre, Goodcur); /* update p if needed */
if (Goodcur > GoodBest)
ZBest =Z;
GoodBest = Goodcur;
$\rho = \rho - R_c$; /* Reward the search with R_c more generations */
else
$\rho = \rho + 1$;
endif
until $\rho > R_c$
return (ZBest);
End

Figure 5.3: The fuzzy stochastic evolution algorithm for DLAN topology design

5.3 Experimental Results

The FStocE and TFStocE algorithms proposed in this chapter were tested on the cases and instances described in Chapter 3. For StocE, there are two main parameters: p (which includes p_0 and p_{incr}), and R_c. These parameters have a significant impact on the performance of the algorithm. Inappropriate values for these parameters may result in non-optimal solutions. Thus, it is important to find the most appropriate parameter setup. Table 5.1 provides a summary of these parameters for the FStocE algorithm. Five different combinations of these parameters were tried, as depicted in the table. A variety of experiments were conducted to evaluate the performance of the two algorithms. The first set of experiments focussed on the comparison of FStocE and TFStocE. For the second set, a comparative analysis of OWA and UAO operators using TFStocE was done. For each variant of FStocE, 30 runs were executed for each test case, and the average and standard deviations of overall goodness of best solutions were calculated. For TFStocE, tabu lists of different sizes as described in Section 5.3.1 were used. Since the aim of this thesis is to mutually compare different algorithms, the same initial solution was used for all simulations of all algorithms discussed in this thesis.

Table 5.1: Parameter settings for fuzzy StocE used in the experiments.

Parameter set	Values		
	p_0	p_{incr}	R
Set 1	0.5	0.2	50
Set 2	0.1	0.05	50
Set 3	0.01	0.005	20
Set 4	0.05	0.01	50
Set 5	0.1	0.05	100

5.3.1 Effect of Tabu List Size

Glover [100] in his research article raised issues about the appropriate magnitude of tabu list sizes. He states, "previous applications had found effective tabu list sizes to lie in the range from 5 to 12, clustered around 7, a finding that appeared to be independent of problem size and structure. The much larger tabu list sizes for the traveling salesman problem, and their dependency on problem size, show that the choice of a good tabu list size is more subtle that previous empirical outcomes had suggested". There are two main inferences from this statement. First, previous studies suggested that the size of tabu list was clustered around 7, irrespective of the nature and size of the problem, and Glover does not seem to agree with this. Second, as the problem size increases, the size of the appropriate tabu list size increases accordingly. To verify the above two inferences, the impact of a tabu list size of 7 on the quality of obtained solutions was analyzed. In addition, to see if better results could be obtained with tabu list sizes other than 7, tabu list sizes of 3, 5, 9, 11, and 13 were also tried. Experimentation with different tabu list sizes also provided an insight into Glover's observation as whether the best tabu list size increased with increasing the problem size. Tables 5.2 and 5.3 summarize the average overall goodness for the best solutions for each test case with different tabu list sizes for OWA and UAO respectively.

The results in Table 5.2 show that the size of tabu list that is related to best overall goodness varies for each case when OWA is used. A graphic illustration of the variation in average overall goodness with respect to different tabu list sizes for OWA is given in Figure 5.4. The second last column of Table 5.2 provides the percentage difference in the average overall goodness of the given tabu list size

when compared with that of size 7. Since size 7 was used as the reference, no percentage improvement is shown for this size, and NA (i.e not applicable) appears in the corresponding row in the second last column. For example, for case *n15*, average overall goodness obtained with size 5 was 6.44% better than that of size 7. For most of the cases, a mid-size tabu list seems to be most appropriate. For example, for cases *n40* and *n50*, sizes 9 and 7 respectively produced the best average overall goodness. For *n15*, size 5 resulted in the best overall goodness, while for *n33* and *n25*, size 11 produced the best results. However, to statistically validate the results, a two-sided t-test was also performed to test the hypothesis whether the two averages (i.e. the average overall goodness obtained with tabu list size of XYZ and that of 7) were significantly different from each other. The t-test results were obtained at 5% significance level. Percentage improvements which are statistically significant are shown in italics. Two important observations come out of the t-test results. First, in general there is no concrete evidence that the results produced by the tabu list size of 7 were statistically more significant than the results produced by other tabu list sizes, for example, for cases *n15*, *n33* (expect for tabu list size of 9) and *n40*. However, there is one case *n50*, where the size of 7 was able to obtain statistically more significant results than the sizes 3, 5, and 11. Second, there is the case of *n25* where almost all sizes (except 13) were able to achieve statistically more significant results than the ones obtained with size 7.

The above discussion suggests that it is not necessary that only size 7 would always produce the best results, since other sizes produced results that were statistically equivalent to the results of size 7. In addition, there were instances where tabu list sizes other than 7 produced statistically better results. Therefore, the re-

sults confirm Glover's disagreement with having the tabu list size of 7 as the best choice. Second, there was no concrete evidence that as the test case size increased, the tabu list size that produced the best results also increased. This negates the second inference from Glover's statement.

As for UAO, a trend similar to that of OWA was observed, where the tabu list varied with each test case. As shown in Table 5.3, a tabu list size of 9 produced the best results for *n15*, while for *n25* and *n33*, the best tabu list size was 7. For n40, size 3 produced the best results, whereas for *n50*, size 13 produced the best overall goodness. A graphic illustration of variation in average overall goodness with respect to different tabu list sizes for UAO is presented in Figure 5.5. The t-test results suggest that for all test cases, a tabu list size of 7 was unable to achieve statistically better results when compared to other sizes. Thus, Glover's first observation holds valid for UAO as well. However, the best tabu list size did not increase with the increase in problem size, which negates Glover's observation.

The conclusion from the above discussion is that it is not true that a tabu list size of 7 would always produce the best results for any problem. Moreover, the size of the best tabu list size is not proportional to the size of test case. The size of the tabu list depends on the structure of the problem, and not on the size of the test case.

5.3.2 Comparison of FStocE and TFStocE

Tables 5.4 and 5.5 respectively summarize the average results obtained by FStocE and TFStocE with the OWA and UAO operators. For each test case, different values of p_0, p_{incr}, and R_c as given in Table 5.1 were tried. The set of parameter values

Table 5.2: Effect of tabu list size on the quality of overall goodness for TFStocE using OWA. Run time is in seconds. Statistically significant improvement is in italics. NA = Not Applicable (since size 7 was used as the reference for comparison).

Test Case	Tabu list size	Avg. Overall goodness	% Improvement	Run time
n15	3	0.102 ±0.050	-9.24	5.7
	5	0.120 ± 0.053	6.44	4.6
	7	0.112 ± 0.049	NA	5.5
	9	0.108 ± 0.044	-4.20	6.0
	11	0.114 ± 0.054	1.12	5.6
	13	0.108 ± 0.054	-3.91	6.1
n25	3	0.152 ± 0.024	*15.73*	26.1
	5	0.143 ± 0.008	*8.55*	26.2
	7	0.132 ± 0.020	NA	24.2
	9	0.155 ± 0.021	*17.38*	24.7
	11	0.160 ± 0.035	*21.27*	30.1
	13	0.141 ± 0.009	7.13	30.1
n33	3	0.096 ± 0.041	-1.05	36.6
	5	0.089 ± 0.042	-8.47	31.9
	7	0.097 ± 0.040	NA	40.1
	9	0.075 ± 0.032	*-22.50*	28.4
	11	0.099 ± 0.045	1.73	36.2
	13	0.083 ± 0.026	-14.93	37.9
n40	3	0.119 ± 0.054	0.28	119.9
	5	0.111 ± 0.055	-6.43	116.0
	7	0.119 ± 0.051	NA	125.0
	9	0.131 ± 0.053	10.30	145.1
	11	0.111 ± 0.057	-6.82	107.2
	13	0.121 ± 0.053	1.88	162.5
n50	3	0.153 ± 0.048	*-14.12*	1257.7
	5	0.153 ± 0.052	*-14.36*	1253.7
	7	0.178 ± 0.044	NA	1341.1
	9	0.162 ± 0.052	-8.89	1296.7
	11	0.152 ± 0.053	*-14.81*	1291.5
	13	0.159 ± 0.047	-10.65	1286.9

Table 5.3: Effect of tabu list size on the quality of overall goodness for TFStocE using UAO. Run time is in seconds. Statistically significant improvement is in italics. NA = Not Applicable (since size 7 was used as the reference for comparison).

Test Case	Tabu list size	Avg. Overall goodness	% Improvement	Run time
n15	3	0.223 ± 0.015	-2.44	0.4
	5	0.226 ± 0.017	-1.22	0.6
	7	0.229 ± 0.019	NA	0.7
	9	0.245 ± 0.032	7.05	0.7
	11	0.228 ± 0.031	-0.28	0.6
	13	0.242 ± 0.033	5.82	1.2
n25	3	0.271 ± 0.008	-1.22	13.2
	5	0.273 ± 0.006	-0.75	14.7
	7	0.275 ± 0.008	NA	16.0
	9	0.272 ± 0.005	-1.12	15.6
	11	0.273 ± 0.009	-0.42	17.1
	13	0.276 ± 0.017	0.35	15.2
n33	3	0.222 ± 0.032	-0.92	10.5
	5	0.216 ± 0.011	-3.60	10.5
	7	0.224 ± 0.032	NA	11.3
	9	0.216 ± 0.011	-3.60	11.1
	11	0.220 ± 0.014	-1.75	12.7
	13	0.224 ± 0.032	0.00	12.4
n40	**3**	0.318 ± 0.032	5.31	143.1
	5	0.318 ± 0.062	5.34	136.2
	7	0.302 ± 0.062	NA	136.8
	9	0.301 ± 0.041	-0.21	136.5
	11	0.289 ± 0.044	-4.32	131.8
	13	0.304 ± 0.045	0.71	132.1
n50	3	0.244 ± 0.020	-1.34	48.3
	5	0.246 ± 0.023	-0.51	50.8
	7	0.247 ± 0.030	NA	48.5
	9	0.250 ± 0.031	1.32	48.8
	11	0.244 ± 0.018	-1.25	50.3
	13	0.256 ± 0.025	3.67	54.2

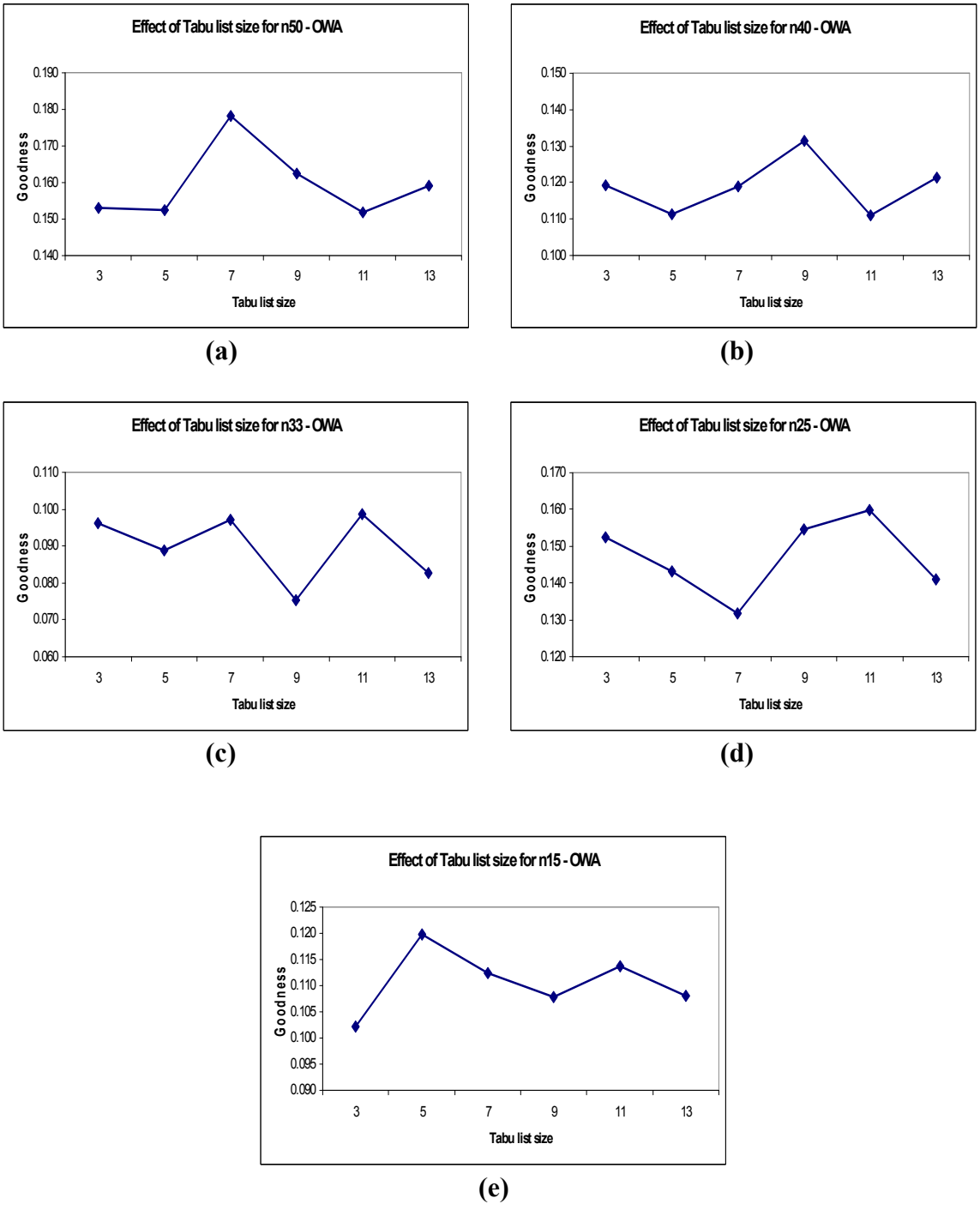


Figure 5.4: Plots of average overall goodness versus tabu list size for FStocE using the OWA operator for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

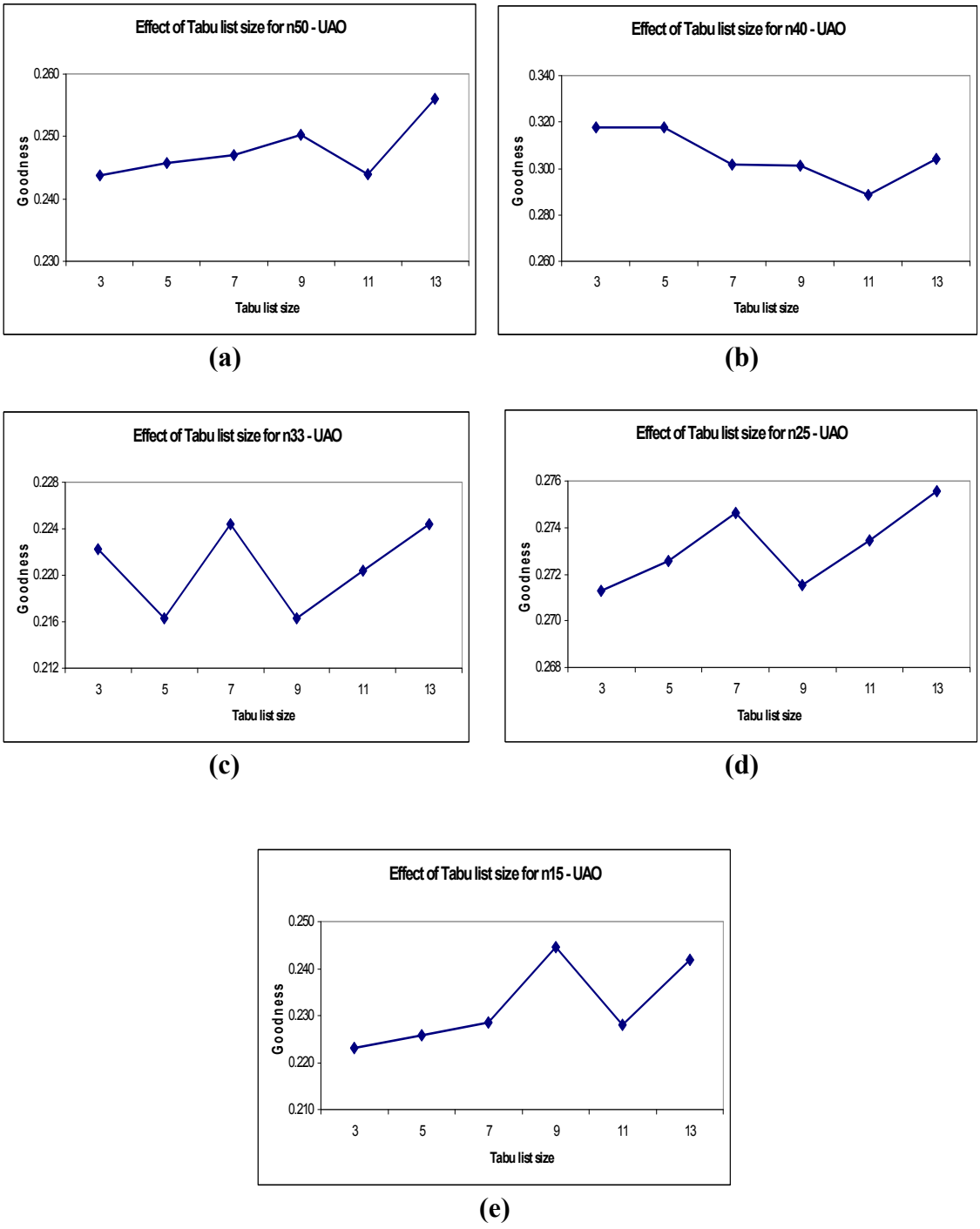


Figure 5.5: Plots average overall goodness versus tabu list size for FStocE using the UAO operator for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

which gave the best results for FStocE are reported in the tables. The same set of parameters was used for TFStocE and the results are also presented in the tables. The reason for using the same parameter setup for both FStocE and TFStocE was to observe the performance of the two algorithms under the same conditions.

For the OWA operator, the general trend for FStocE is that as the size of the test case increases, the values of p_0, p_{incr}, and R_c which resulted in the best overall goodness also increase as observed in Table 5.4. For example, for *n25*, best average overall goodness was obtained with $p_0 = 0.01$, $p_{incr} = 0.005$, and $R_c = 20$, and as the size of the test case increases, these parameter values also need to be increased. For example, for *n50*, $p_0 = 0.1$, $p_{incr} = 0.05$, and $R_c = 100$. The only exception to this trend was for test case *n15*. As far as the relative performance of FStocE and TFStocE (with respect to the quality of produced solutions) is concerned, it is very obvious from Table 5.4 that TFStocE demonstrated far better performance for all test cases. The percentage improvements ranged between 17% and 43%. The t-test (performed at 5% significance level) also suggested that all improvements achieved by TFStocE over FStocE were statistically significant.

With regard to the UAO operator, a behavior similar to that of OWA is observed for FStocE, where the values of p_0, p_{incr}, and R_c for the best overall goodness values need to be increased when the size of the test case increases. The only exception was for *n50*, where the best overall goodness was obtained for $p_0 = 0.01$, $p_{incr} = 0.005$, and $R_c = 20$. In comparison with FStocE, TFStocE generally showed improvements in the quality of solutions in the range of 4% to 12%. There were exceptions such as *n50* and *n33* where FStocE was able to achieve slightly better results than TFStocE. However, statistical testing (with t-test) of the results suggested that the improve-

ment achieved by TFStocE was significant only for *n40*. For other test cases, no significant differences were found. Thus, for UAO, the performance of TFStocE and FStocE was generally more or less the same.

Table 5.4: Comparison of FStocE and TFStocE for OWA. TL = Tabu List Size, Time = Run time (in seconds), and % imp = percentage improvement. Statistically significant percentage improvements are in italics.

Case	Values			FStocE		TFStocE			% imp
	p_0	p_{incr}	R_c	Avg. Overall Goodness	Time	TL	Avg. Overall Goodness	Time	
n15	0.05	0.01	50	0.069±0.036	2.8	5	0.120±0.053	4.6	*42.50*
n25	0.01	0.005	20	0.116±0.021	62.4	11	0.160±0.035	30.1	*27.50*
n33	0.01	0.005	20	0.079±0.022	54.4	11	0.099±0.045	36.2	*20.20*
n40	0.05	0.01	50	0.102±0.054	262.8	9	0.131±0.053	145.1	*22.14*
n50	0.1	0.05	100	0.147±0.055	1669.2	7	0.178±0.044	1341.1	*17.42*

Table 5.5: Comparison of FStocE and TFStocE for UAO. TL = Tabu List Size, Time = Run time (in seconds), and % imp = percentage improvement. Statistically significant percentage improvements are in italics.

Case	Values			FStocE		TFStocE			% imp
	p_0	p_{incr}	R_c	Avg. Overall Goodness	Time	TL	Avg. Overall Goodness	Time	
n15	0.01	0.005	20	0.233 ±0.021	0.8	9	0.245 ±0.032	0.7	4.90
n25	0.05	0.01	50	0.262 ±0.027	26.8	7	0.275 ±0.008	16.0	4.73
n33	0.05	0.01	50	0.227 ±0.037	16.3	7	0.224 ±0.032	11.3	-1.34
n40	0.1	0.05	100	0.280 ±0.048	319.6	3	0.318 ±0.032	143.1	*11.95*
n50	0.01	0.005	20	0.268 ±0.039	991.8	13	0.256 ±0.025	54.2	-4.69

It is observed from the above discussion that as compared to FStocE, TFStocE was able to achieve significantly better results for OWA, and slightly better results for UAO. This better performance of TFStocE can be attributed to the many constraints which limit the feasible search space. It may happen that, after some iterations, a number of moves are repeated, and thus the FStocE algorithm revisits

Table 5.6: Ratio of tabu moves for TFStocE using UAO.

Test case	Tabu list size for best solution	Total feasible moves	Tabu moves	% of Tabu moves
n15	5	1362.1	39.3	2.89
n25	11	2673.6	205.6	7.69
n33	11	1459.9	102.0	6.99
n40	9	2648.3	350.9	13.25
n50	7	10804.9	1028.7	9.52

the same part of the search space. However, TFStocE will list these repetitive moves as being tabu, causing the algorithm to diversify the search into another subarea. Recall that the *PERTURB* function randomly removes a link from the current solution, and adds a new feasible link to the solution. This new link is also saved in the tabu list. It may happen that this new link is removed in the following iterations and later reintroduced in the solution, but, since the link is in the tabu list, it will not be chosen again, thus allowing other links to be chosen. This increases exploration of the search space for TFStocE, resulting in better solutions than FStocE. The above reasoning is supported by the results in Table 5.6, which provides the ratio of tabu moves compared to the total number of feasible moves attempted by the TFStocE algorithm using the UAO operator. Note that tabu moves are a subset of all the feasible moves. As an example, consider *n25*, where on average, 2673.6 feasible moves were made, and 205.6 moves were tabu, resulting in 7.69% of tabu moves. This means that for these 7.69% tabu moves, other additional feasible moves were attempted, thus preventing the TFStocE from cycling back to the same moves. Had it been the FStocE algorithm, those 7.69% moves would have been repeated since there was no mechanism in FStocE to prevent repetitive acceptance of the same moves.

Table 5.7: Effect of different R_c values on overall goodness of solutions with $p_0 = 0.1$ and $p_{incr} = 0.05$ for OWA and UAO. Statistically significant difference is in italics.

Case	OWA (Goodness)			UAO (Goodness)		
	R_c= 50	R_c= 100	% Difference	R_c= 50	R_c= 100	% Difference
n15	0.06	0.07	7.48	0.24	0.26	8.54
n25	0.04	0.05	19.84	0.13	0.19	*29.81*
n33	0.07	0.04	*-96.64*	0.12	0.11	-11.58
n40	0.03	0.02	*-88.93*	0.27	0.28	4.64
n50	0.08	0.15	*48.65*	0.21	0.14	*-52.88*

5.4 Dynamic Value of R_c

The parameter R_c is used to decide how many extra iterations should be rewarded to the algorithm to continue the search. Thus in an iteration, if the goodness of the current solution is better than the goodness of the best solution found so far, then the algorithm takes the current solution Z as the best solution, $Zbest$, and decrements R_c by ρ, thereby rewarding itself by increasing the number of iterations and allowing the search to continue for R_c more iterations. The impact of this is that the algorithm is allowed to perform a more detailed investigation of the neighborhood, since more and more iterations are rewarded as long as the algorithm keeps finding a solution better than the current best solution. One point to note here is that the basic FStocE algorithm is always rewarded with the same number of extra iterations to perform the search, regardless of the level of improvement achieved by the algorithm during execution. This approach poses one important question of how to find the appropriate number of extra iterations that need to be awarded. If too few extra iterations are allowed, then the algorithm may not be able to explore the search space to a satisfactory level. If too many iterations are allowed, then the algorithm may waste time in exploring the search space without

producing any improvement. To understand this, consider the results for OWA and UAO in Table 5.7 which lists the effect of two different R_c values with other algorithm parameters kept constant. The results for OWA show that for *n50*, R_c = 50 was able to produce statistically better results with reference to R_c = 100 (validated through the t-test). Furthermore, the difference for *n15* and *n25* for both R_c values was statistically insignificant. This suggests that the FStocE algorithm wasted time by executing extra iterations unnecessarily with R_c = 100, since the same (or even better) results were achievable with R_c = 50 for the three test cases. However, for *n33* and *n40*, R_c = 100 produced statistically significant results with reference to R_c = 50. This suggests that a lower value of R_c = 50 was not sufficient to obtain the same level of results as those of R_c = 100. The same explanation can be given for results of UAO in Table 5.7, where R_c = 50 and R_c = 100 produced similar results (statistically), but R_c = 100 produced higher quality results for *n25* and *n50*.

One way of finding an appropriate value of R_c is to do several trial runs of the algorithm with different numbers of extra iterations. However, these trial runs will result in finding the best number of iterations for only that instance of the problem. As the problem changes, or even an instance of the problem changes, another set of trial runs will be required to find the best number of iterations. Therefore, the "trial run" approach cannot be used as a general rule for any set of problems. Furthermore, the approach also causes excessive execution time for all the trial runs. To overcome this problem, a "performance-based" rewarding scheme is proposed in this section. The objective of this performance-based scheme is to reward the algorithm with less iterations the better the improvement, and more

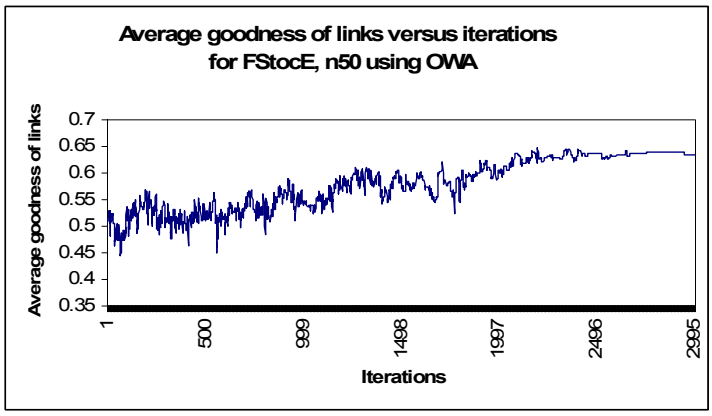
iterations the lower the improvement. The logic behind this approach is that a high goodness value suggests that the algorithm might be near convergence. This claim can be proven using average goodness of links as a measure. A high average goodness of links suggests that the majority of links in the solution are placed in their optimal positions, causing the algorithm to converge to the near-optimal position.

Figure 5.6 elaborates on the above phenomenon where Figure 5.6(a) shows the improvement in the average goodness of links versus iterations for a typical run of FStocE. Figure 5.6(b) shows the corresponding improvement in the goodness of solution with iterations. Notice that, as the average goodness of links improves, the goodness of solution also improves. Towards the end of the run, the average goodness of links ceases to improve any further, suggesting that most of the links have found their optimal positions. Accordingly, the goodness of solution does not improve any further, which is a sign that the algorithm has converged to a sub-optimal solution. Thus, awarding the algorithm with more iterations at the near-convergence situation might waste computational time in unnecessary traversing of the search space. On the other hand, if the goodness is low, the algorithm still needs more time to improve, and thus more iterations are required to give the algorithm sufficient time to traverse more of the search space, so as to possibly improve the quality of the solution.

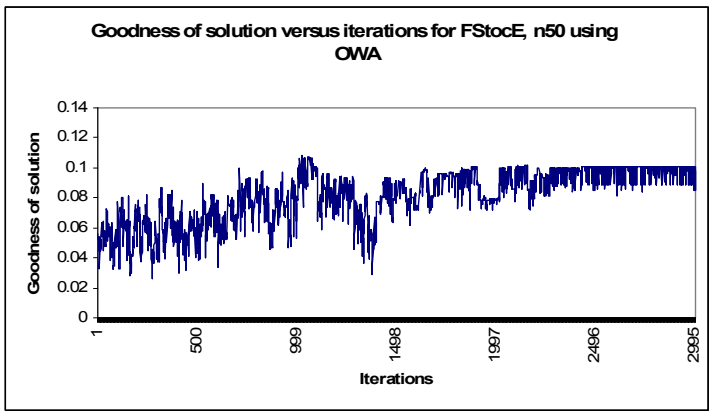
One way to achieve this is to associate the number of extra iterations with the improvement achieved by the algorithm as given by

$$\mathrm{R_{c_i}} = \frac{1}{GoodPre(Z)} \tag{5.1}$$

where R_{c_i} is the value of R_c at iteration i and $GoodPre(Z)$ is the goodness of



(a)



(b)

Figure 5.6: Plots of average overall goodness versus tabu list size for FStocE using the OWA operator for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

solution Z in the immediate previous iteration. Equation (5.1) allows the algorithm to dynamically assign the number of extra iterations. Also note that the required information (i.e. the goodness of the previous solution) to dynamically calculate R_c in Equation (5.1) is extracted from the problem instance itself.

The above strategy to dynamically calculate R_c was incorporated into TFStocE, and the resulting variant is called DTFStocE. Tables 5.8 and 5.9 respectively summarize the comparative results of FStocE and DTFStocE for OWA and UAO operators. According to Table 5.8, a degradation was observed in the quality of

overall goodness produced by DTFStocE as compared to FStocE in most of the cases. The degradation was mainly in the range of 14% to little over 21%. For *n50* a degradation of over 78% was observed. For *n25* DTFStocE was able to perform better than FStocE with an improvement of 14.51%. However, statistical analysis using t-test revealed that there was no significant difference in the performance of the two algorithms, except for *n50*. Therefore, for OWA, the dynamic R_c approach performed satisfactorily in general. As for the UAO, DTFStocE showed degradation in quality of results as compared to FStocE, with the degradation in the range of 4.5% to 20.5%. However, statistical analysis with t-test suggested that the degradation for *n15* and *n33* was not significant, but significant for *n25*, *n40*, and *n50*. Therefore, the performance of DTFStocE for UAO was not very appreciable.

A general observation from the above discussion is that DTFStocE showed an overall satisfactory performance compared to FStocE. However, the percentage difference between the results of DTFStocE and FStocE suggest that there is still scope to further investigate alternative methods. Such methods may use problem dependent or instance dependent information such as overall goodness of the solution (as in Equation (5.1)), goodness of each individual (link) in the solution, and/or number of nodes in the network.

5.5 Comparison of OWA and UAO Operators

As mentioned earlier in this chapter, the OWA and UAO operators were used to combine the four design objectives using Equations (3.11) and (4.1) respectively. In this section, a comparison of these two fuzzy operators is presented with respect

Table 5.8: Comparison of FStocE and DTFStocE for OWA. Time = Run time (in seconds), and % imp = percentage improvement. % improvement is for DTFStocE compared to FStocE. Statistically significant improvement is in italics.

Case	Values			FStocE		DTFStocE		% imp
	p_0	p_{incr}	R_c	Avg. Overall Goodness	Time	Avg. Overall Goodness	Time	
n15	0.05	0.01	50	0.069 ±0.036	2.8	0.059 ±0.014	0.4	-17.58
n25	0.01	0.005	20	0.116 ±0.021	62.4	0.136 ±0.034	12.9	14.51
n33	0.01	0.005	20	0.079 ±0.022	54.4	0.069 ±0.028	16.9	-14.78
n40	0.05	0.01	50	0.102 ±0.054	262.8	0.084 ±0.041	30.5	-21.64
n50	0.1	0.05	100	0.147 ±0.055	1669.2	0.082 ±0.038	120	*-78.19*

Table 5.9: Comparison of FStocE and DTFStocE for UAO. Time = Run time (in seconds), and % imp = percentage improvement. % improvement is for DTFStocE compared to FStocE. Statistically significant improvement is in italics.

Case	Values			FStocE		DTFStocE		% imp
	p_0	p_{incr}	R_c	Avg. Overall Goodness	Time	Avg. Overall Goodness	Time	
n15	0.01	0.005	20	0.233 ±0.021	0.8	0.223 ±0.015	0.1	-4.50
n25	0.05	0.01	50	0.262 ±0.027	26.8	0.237 ±0.020	0.7	*-10.53*
n33	0.05	0.01	50	0.227 ±0.037	16.3	0.212 ±0.000	1.2	-7.01
n40	0.1	0.05	100	0.280 ±0.048	319.6	0.237 ±0.032	3.9	*-18.37*
n50	0.01	0.005	20	0.268 ±0.039	991.8	0.222 ±0.033	6.2	*-20.48*

to their application to TFStocE. The tabu version of FStocE was chosen since it produced the best results with respect to the other two variants (FStocE and DTFStocE). These comparisons focussed on the effect of the four design objectives, namely cost, delay, hops, and reliability with respect to change in the number of nodes. In all experiments conducted in this study, $\beta = 0.5$ was chosen for the OWA operator in Equation (3.11) and $\nu = 0.5$ was used for the UAO operator in Equation (4.1). A simple linear regression analysis (see Appendix B for background) was performed with the number of nodes as the independent variable and each one

Table 5.10: Comparison of OWA and UAO for TFStocE.

Objective	Regression coefficients		Ratio = $\frac{OWA}{UAO}$	% Gain by UAO	Comment
	OWA	UAO			
Cost	0.947	0.775	1.22	18.16	UAO performs 18.16% better than OWA
Delay	0.672	0.549	1.22	18.30	UAO performs 18.30% better than OWA
Hops	0.753	0.507	1.49	32.67	UAO performs 32.67% better than OWA
Rel	-0.76	-0.56	1.36	26.32	UAO performs 26.32% better than OWA

of the design objectives as a dependent variable. Several regression models were developed to see how a design objective is affected by increasing number of nodes when OWA and then UAO is used. The effect is measured through regression coefficients. A confidence level of 95% was used. The analysis was based on 150 data values, since there were five test cases with five different number of nodes, where for each test case, 30 runs were made.

Table 5.10 presents a comparison of OWA and UAO for TFStocE considering all five test cases from *n15* to *n50*. Design objectives are listed in the first column. The regression coefficients for the OWA and UAO regression models are given in columns 2 and 3 respectively. The fourth column gives the ratio of regression coefficients of OWA versus UAO. This ratio signifies the rate of increase for a certain objective if the performance of OWA and UAO are compared for that objective. The percentage gain given in column 5 shows the improvement achieved by the UAO operator as compared to the OWA operator. A further comment elaborates on this finding in column 6. For example, the regression coefficients for *Cost* are given for OWA and UAO as 0.947 and 0.775 respectively. If these two values are

compared, the corresponding ratio is 1.22. The interpretation is that, as the number of nodes are increased, the rate at which *Cost* increases using OWA is 1.22 times faster than if UAO was used. This is equal to an 18.16% increase in performance for UAO compared to OWA. The same approach can be used for the *Delay* and *Hops* objectives in terms of the ratio and percentage gain. It is observed that, for these two objectives, UAO performs much better than OWA. Similarly, for *Reliability*, the regression coefficients have a negative sign. This negative sign implies an inverse relationship between number of nodes and reliability, i.e. the reliability decreases as the number of nodes is increased. The corresponding OWA/UAO ratio is 1.36, suggesting that, as the size of test case increases, the rate at which reliability deteriorates using OWA is 1.36 times faster than if UAO is employed. The percentage of 26.32% also suggests the same observation; the UAO would be 26.32% better than OWA in terms of controlling the decline in reliability. Thus, the analysis of the results in Table 5.10 suggest that UAO is undoubtedly performing better than OWA for the design objectives.

5.6 Conclusions

This chapter presented and investigated the fuzzy stochastic evolution algorithm (FStocE) for DLAN topology design. A variant of the proposed fuzzy stochastic evolution algorithm, 'TFStocE', was also proposed. This variant introduced tabu search characteristics to the FStocE algorithm. The effect of tabu list size was investigated, revealing that the size of the tabu list is related to the problem under investigation as well as the test instances of the problem. Moreover, empirical evalu-

ation and comparison of FStocE and TFStocE suggested that, in general, TFStocE produced better results than FStocE for both the OWA and UAO operators. An investigation of dynamic computation of R_c and comparison with FStocE showed that generally there was no significant difference in the results for the OWA operator, while for UAO lower quality solutions were obtained compared to the FStocE algorithm. As far as the effectiveness of the OWA and UAO operators are concerned, the investigation found that UAO performed much better than OWA in optimizing each of the four design objectives.

The focus of the next chapter is on another optimization algorithm, namely simulated evolution, which has been adapted to solve the multi-objective DLAN topology design problem.

Chapter 6

Fuzzy Simulated Evolution for DLAN Topology Design

A fuzzy multi-objective simulated evolution (FSimE) algorithm is proposed in this chapter. FSimE combines the four design objectives into one fuzzy function and optimizes this single fuzzy objective. This chapter first describes the steps of the proposed FSimE algorithm, and how fuzzy logic has been incorporated. This is followed by a modified version of FSimE, where tabu search characteristics are incorporated into the FSimE algorithm. Another modification of FSimE is proposed later in this chapter, with the purpose to reduce user dependency in setting the value of the bias factor of FSimE. The performance of FSimE and its variants are empirically assessed and mutually compared.

6.1 Fuzzy Simulated Evolution Algorithm

Chapter 2 discussed the general SimE algorithm, which consists of four steps: initialization, evaluation, selection, and allocation. Of these steps, evaluation and allocation are of special interest, since they involve assessment of the solution and play a key role in the overall performance of the algorithm. For example, in the *evaluation* step, the quality of each individual (a link in this case) of the current solution is evaluated based on a goodness measure. The need is to find an appropriate goodness measure. Similarly, in the *allocation* phase, the current solution is perturbed to generate a new solution. Again, an appropriate assessment function is required to compare the quality of the old and the new solutions. In both these steps, fuzzy logic plays an important role, as explained below for the FSimE algorithm.

6.1.1 Initialization

The initial spanning tree topology can be generated randomly, while taking into account all design constraints mentioned earlier. However, since the aim of this thesis is to mutually compare different algorithms, the initial solution is predefined and is used for all algorithms discussed in this thesis.

6.1.2 Fuzzy Evaluation

The **goodness** of each individual is computed as follows. For the purposes of this thesis, an individual represents a **link** interconnecting two network devices. For the *fuzzy evaluation scheme*, monetary cost, link reliability, and depth of a link are considered fuzzy variables. Depth of a link is measured as the distance from the

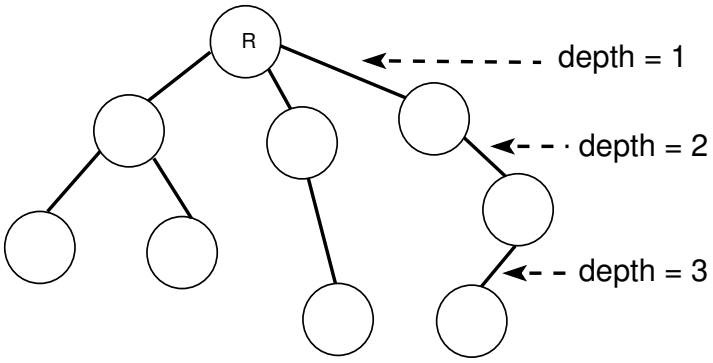


Figure 6.1: Depths of links with respect to the root node R

root in the spanning tree, as illustrated in Figure 6.1. Considering the above three variables, goodness of a link is then characterized by the following rule:

Rule 2: IF a link is *near optimum cost* AND *near optimum depth*
AND *near optimum reliability* **THEN** it has *high goodness*

Here, *near optimum cost*, *near optimum depth*, *near optimum reliability*, and *high goodness* are linguistic values for the fuzzy variables cost, depth, reliability, and goodness respectively. Using OWA-AND, Rule 2 translates to the following equation for the fuzzy goodness measure of link l_i:

$$g_{l_i} = \mu^e(l_i) = \beta^e \times min\{\mu_1^e(l_i), \mu_2^e(l_i), \mu_3^e(l_i)\} + (1 - \beta^e) \times \frac{1}{3} \sum_{i=1}^{3} \mu_i^e(l_i) \qquad (6.1)$$

The superscript e denotes **evaluation**. In Equation (6.1), $\mu^e(l_i)$ is the degree of membership to the fuzzy set of *high goodness links* and $\beta^e \in [0, 1]$ is a constant, which represents the degree to which the OWA operator resembles the pure "AND"; $\mu_1^e(l_i)$, $\mu_2^e(l_i)$, and $\mu_3^e(l_i)$ respectively represent memberships in the fuzzy sets *near optimum monetary cost*, *near optimum depth*, and *near optimum reliability* respectively.

The membership of a link with respect to *near optimum monetary cost* is determined as follows: from the cost matrix, which gives the costs of each possible link, the minimum and maximum costs among all the link costs are found. These minimum and maximum costs are taken as the lower and upper bounds, and are termed as "LCostMin" and "LCostMax", respectively. Then, the membership of a link with respect to cost, μ_1, is calculated with respect to these bounds as follows:

$$\mu_1 = \begin{cases} 1 & \text{if } LCost \leq LCostMin \\ \frac{LCostMax - LCost}{LCostMax - LCostMin} & \text{if } LCostMin < LCost \leq LCostMax \\ 0 & \text{if } LCost > LCostMax \end{cases} \tag{6.2}$$

where the term '$LCost$' represents the cost of the link. In the same manner, membership of a link with respect to *near optimum depth* can be found. The lower limit, called "LDepthMin", is taken to be a depth of 1 with respect to the root. The upper bound,"LDepthMax" is taken to be the maximum depth generated in the initial solution or a user-specified maximum limit.[1] The membership function with respect to *near optimum depth*, μ_2 is calculated using Equation (6.3) as follows:

$$\mu_2 = \begin{cases} 1 & \text{if } LDepth \leq LDepthMin \\ \frac{LDepthMax - LDepth}{LDepthMax - LDepthMin} & \text{if } LDepthMin < LDepth \leq LDepthMax \\ 0 & \text{if } LDepth > LDepthMax \end{cases} \tag{6.3}$$

[1]This user specified limit may be a design constraint, e.g., if each hop represents a router that uses the routing information protocol (RIP), then a reasonable limit would be 7, i.e. a branch of the tree should not have more than 7 routers.

where the term '*LDepth*' represents the depth of the link. Finally, the membership of a link with respect to *near optimum reliability* is determined as follows. From the reliability matrix, which gives the reliability of each possible link, the minimum and maximum reliabilities among all the link reliabilities are found. These minimum and maximum reliabilities are taken as the lower and upper bounds, "LRelMin" and "LRelMax", respectively. Then, the membership of a link for reliability, μ_3, is calculated with respect to these bounds as follows:

$$\mu_r(x) = \begin{cases} 1 & \text{if } LRel \geq LRMax \\ \frac{LRMax - LRel}{LRMax - LRMin} & \text{if } LRMin < LRel \leq LRMax \\ 0 & \text{if } LRel < LRMin \end{cases} \tag{6.4}$$

where $LRel$ represents the reliability of the link. For the purposes of this thesis, five values of link reliabilities were used, namely, 0.99, 0.95, 0.9, 0.85, and 0.8, as used by Altiparmak *et al.* [6] in a similar study. Thus, $LRelMin = 0.8$ and $LRelMax = 0.99$.

Selection

The selection process sample for each link, l_i, in the current tree topology, where i = 1,2,..., n-1, a random number $RANDOM \in [0, 1]$. If $RANDOM > g_{c_i} + B$, where B is the selection bias, then link l_i is selected for allocation and considered removed from the topology. The bias is used to control the size of the set of links selected for removal, as described earlier in Section 2.4.2.

6.1.3 Fuzzy Allocation

The allocation step of the algorithm removes the selected links from the topology one at a time. For each removed link, a new link is introduced in the topology, provided that the constraints are not violated. This procedure is repeated for all the links that are present in the set of selected links. The strategy of this operation is somewhat similar to the one used in the FStocE algorithm. However, in FSimE, prior to the allocation step, the selected links are sorted according to their goodness values, with the link with the worst goodness at the head of the list.

The *fuzzy allocation scheme* combines the four criteria to be optimized using fuzzy logic to characterize a good topology. This overall cost is computed using fuzzy Rule 1 discussed in Section 3.4, using either Equation (3.11) or Equation (4.1).

6.2 Tabu Simulated Evolution

This section proposes a new hybrid iterative search technique, tabu simulated evolution (TFSimE), which introduces features of tabu search in the allocation phase of the SimE algorithm. TFSimE takes an approach very similar to TFStocE described in Chapter 5. In TFSimE, a user-specified number of moves are made for each link in the selection set, and the best move is accepted, making the move (i.e. link) permanent. This newly accepted link is also saved in the *tabu list* to prevent cycling back to the same solution. As described in Section 2.4.5, tabu search requires that *attributes* and *aspiration criteria* need to be defined. Thus, the attribute is the link itself. The aspiration criterion overrides the tabu status of the link if the tabu link

produces a higher membership value (i.e. overall goodness) than the current one in the membership function "good topology". As with the case of TFStocE, the *tabu search based strategy* prevents the algorithm from repetitively performing the same move in consecutive iterations.

6.3 Experimental Results

The FSimE and TFSimE algorithms proposed in this chapter were tested on the five test cases as described in Chapter 3. For both algorithms, the bias B can have a significant impact on the performance of the algorithms (refer to Section 2.4.2). Thus, it is important to find the most appropriate value of the bias for each problem instance. Experiments with five different bias values were conducted, i.e. 0.0, 0.1, 0.2, 0.3, and 0.4. Values greater than 0.4 were not considered since a very high bias value would reduce the number links selected for removal, thereby increasing the risk of premature convergence, as discussed in Section 2.4.2. To assess the performance of the FSimE and TFSimE algorithms, four different sets of experiments were performed. The first set of experiments focussed on TFSimE, where different tabu list sizes as described in Section 6.3.1 were tried. In the second set, FSimE and TFSimE were compared. In the third set, a variant of TFSimE with a dynamically changing bias, termed as DTFSimE, is discussed and evaluated. Finally, the fourth set of experiments provided a comparative analysis of the OWA and UAO operators using DTFSimE. For each variant of SimE, 30 runs were executed for each test case, and the average and standard deviation of overall goodness of best solutions were calculated.

6.3.1 Effect of Tabu List Size

The effect of the tabu list size was investigated for the TFSimE algorithm with tabu list sizes of 3, 5, 7, 9, 11, and 13. The effect was evaluated for both the OWA and UAO operators. The purpose of this investigation was to see if the observations of Glover [100] (as discussed in Section 5.3.1) can be confirmed. Tables 6.1 and 6.2 summarize the average overall goodness for the best solutions for each test case with different tabu list sizes for OWA and UAO respectively.

Table 6.1 shows that the size of tabu list that is related to best overall goodness varies for each case when OWA is used. A graphic illustration of the variation in average overall goodness with respect to different tabu list sizes for OWA is given in Figure 6.2. Table 6.1 also provides the percentage difference in the average overall goodness of the given tabu list size when compared with size 7. For example, for case *n40*, the average overall goodness obtained with size 13 was 10.11% better than that obtained with size 7. It appears from the results in Table 6.1 that, in general, Glover's first observation that tabu list size of 7 does not always produce the best results seem to be proven true. However, this observation is confirmed through validation of results using the t-test. The t-test was performed to test the hypothesis whether the two averages (i.e. the average overall goodness obtained with tabu list size of XYZ and that of 7) were significantly different from each other. The t-test results were obtained at 5% significance level. Percentage improvements which are statistically significant are shown in italics. The t-test results signify two important issues. First, in general there is no hard evidence that the results produced by the tabu list size of 7 were statistically significantly different than the results produced by other sizes. This observation is obvious from the results of *n25*,

n33, and *n50*, where size 7 failed to produce statistically better results than any other size. Even for test cases *n15* and *n40*, a tabu list size of 7 did not result in significant improvement as a whole; there are only instances where size 7 produced better results than sizes 9 and 13 (for *n15*), and 3 and 9 (for *n40*). Second, there is the case of *n40* where size 13 achieved statistically significantly better results than the results obtained with size 7. However, the above discussion negates Glover's observation that size of the best tabu list (i.e., tabu list that produces the best results) increases as the test case size increases.

As for UAO, a trend similar to that of OWA was observed, where the tabu list for best results varied with each test case. As observed from Table 6.2, there was not a single test case where the tabu list size of 7 seemed to produce the best results. For example, size 3 provided the best results for *n50*, *n40*, and *n15*, while for *n33* the best size was 5. Only in case *n25* did a tabu list size of 11 provide the best overall goodness. A graphic illustration of variation in average overall goodness with respect to different tabu list sizes for UAO is presented in Figure 6.3. The t-test results suggest that for all test cases, in general, size 7 was unable to achieve statistically better results when compared to other tabu list sizes. There were minor exceptions such as *n15* (where size 7 showed improvement over sizes of 5 and 9), and *n50* (where size 3 had a statistically significant improvement over size 7). Moreover, the results also provide an evidence that the best tabu list size did not increase proportional to the size of the test case.

Based on the above discussion, it can be fairly claimed that the tabu list size that resulted in the best solution coincide with Glover's [100] observation that a size of 7 does not always produce the best results. However, the suggestion of Glover

that best tabu list size increases as the test case size increases is not true. The size of the tabu list depends on the nature of the problem, but is not proportional to the size of the test case.

6.3.2 Comparison of FSimE and TFSimE

This section compares the results of the FSimE and TFSimE algorithms. The results of 30 runs obtained by FSimE and TFSimE with the OWA and UAO operators are summarized in Tables 6.3 and 6.4. For each test case, different values of bias B were investigated as listed in Section 6.3. The results reported in the tables are for bias values responsible for producing the best overall goodness values for FSimE. The same bias was used for TFSimE mentioned in Section 6.3 and the best results are also mentioned in the tables.

Results in Table 6.3 show that the size of test cases was inversely proportional to the best bias value. A high bias value generated the best results for small test cases; for example *n15* has a best bias of 0.4. As the size of the test case was increased, the general trend is that the bias value decreased. For example, the bias for *n33* was 0.2, while that of *n40* and *n50* was 0. An exception to this trend is test case *n25*, where a bias of 0 produced the best results. As far as comparison of FSimE and TFSimE is concerned, it is obvious from Table 6.3 that, in general, TFSimE was able to achieve significant improvement in the overall goodness of the solution compared to FSimE. This achievement is more prominent in small and medium size test cases such as *n15*, *n25*, and *n33*. The improvements for these cases were in the range of 14% to 39%. For *n40*, the improvement was about 4.5%. As for *n50*, FSimE was able to perform better than TFSimE, as a deterioration of almost 12% was observed in the

Table 6.1: Effect of tabu list size on the quality of overall goodness for TFSimE using OWA. Run time is in seconds. Statistically significant improvement is in italics. NA = Not Applicable (since size 7 was used as the reference for comparison).

Test Case	Tabu list size	Avg. Overall goodness	% Improvement	Run time
n15	3	0.123 ± 0.020	-1.40	82.9
	5	0.122 ± 0.021	-2.20	82.6
	7	0.125 ± 0.019	NA	83.2
	9	0.116 ± 0.016	*-6.93*	83.9
	11	0.130 ± 0.039	4.51	83.6
	13	0.115 ± 0.016	*-7.53*	84.7
n25	3	0.226 ± 0.017	2.05	221.9
	5	0.226 ± 0.008	1.89	231.8
	7	0.222 ± 0.007	NA	244.0
	9	0.228 ± 0.021	2.84	256.5
	11	0.228 ± 0.020	2.71	268.0
	13	0.223 ± 0.006	0.72	280.6
n33	3	0.202 ± 0.006	0.12	174.1
	5	0.201 ± 0.009	0.04	192.3
	7	0.201 ± 0.006	NA	221.3
	9	0.200 ± 0.008	-0.67	230.9
	11	0.200 ± 0.008	-0.66	253.7
	13	0.201 ± 0.009	-0.34	266.6
n40	3	0.250 ± 0.086	*-9.30*	1568.5
	5	0.256 ± 0.150	-6.87	1590.4
	7	0.275 ± 0.140	NA	1591.8
	9	0.232 ± 0.028	*-15.72*	1747.2
	11	0.268 ± 0.066	-2.71	1805.3
	13	0.303 ± 0.176	*10.11*	1856.9
n50	3	0.219 ± 0.047	0.35	3734.6
	5	0.220 ± 0.036	0.44	3713.1
	7	0.219 ± 0.061	NA	3871.9
	9	0.231 ± 0.094	5.55	3981.1
	11	0.222 ± 0.048	1.62	4321.2
	13	0.220 ± 0.046	0.57	4482.5

Table 6.2: Effect of tabu list size on the quality of overall goodness for TFSimE using UAO. Run time is in seconds. Statistically significant improvement is in italics. NA = Not Applicable (since size 7 was used as the reference for comparison).

Test Case	Tabu list size	Avg. Overall goodness	% Improvement	Run time
n15	**3**	0.320 ± 0.052	2.53	99.5
	5	0.299 ± 0.024	*-4.03*	102.9
	7	0.312 ± 0.034	NA	100.0
	9	0.300 ± 0.014	*-3.60*	101.8
	11	0.313 ± 0.039	0.45	103.6
	13	0.315 ± 0.039	0.93	104.4
n25	3	0.273 ± 0.008	-0.19	94.0
	5	0.276 ± 0.008	1.06	103.0
	7	0.273 ± 0.007	NA	103.0
	9	0.276 ± 0.009	0.90	109.4
	11	0.277 ± 0.011	1.44	117.0
	13	0.270 ± 0.006	-1.02	114.0
n33	3	0.277 ± 0.012	-0.85	133.7
	5	0.282 ± 0.012	0.98	160.8
	7	0.279 ± 0.013	NA	165.2
	9	0.275 ± 0.020	-1.49	167.8
	11	0.274 ± 0.013	-1.59	183.8
	13	0.277 ± 0.011	-0.62	182.5
n40	**3**	0.302 ± 0.010	1.32	360.6
	5	0.295 ± 0.014	-0.93	479.3
	7	0.298 ± 0.010	NA	446.2
	9	0.299 ± 0.010	0.34	471.8
	11	0.302 ± 0.016	1.35	517.1
	13	0.301 ± 0.013	1.01	543.3
n50	**3**	0.314 ± 0.036	*4.17*	3580.1
	5	0.296 ± 0.025	-2.07	3464.5
	7	0.302 ± 0.032	NA	3566.4
	9	0.303 ± 0.036	0.51	3673.9
	11	0.292 ± 0.023	-3.18	3994.1
	13	0.309 ± 0.042	2.31	4346.1

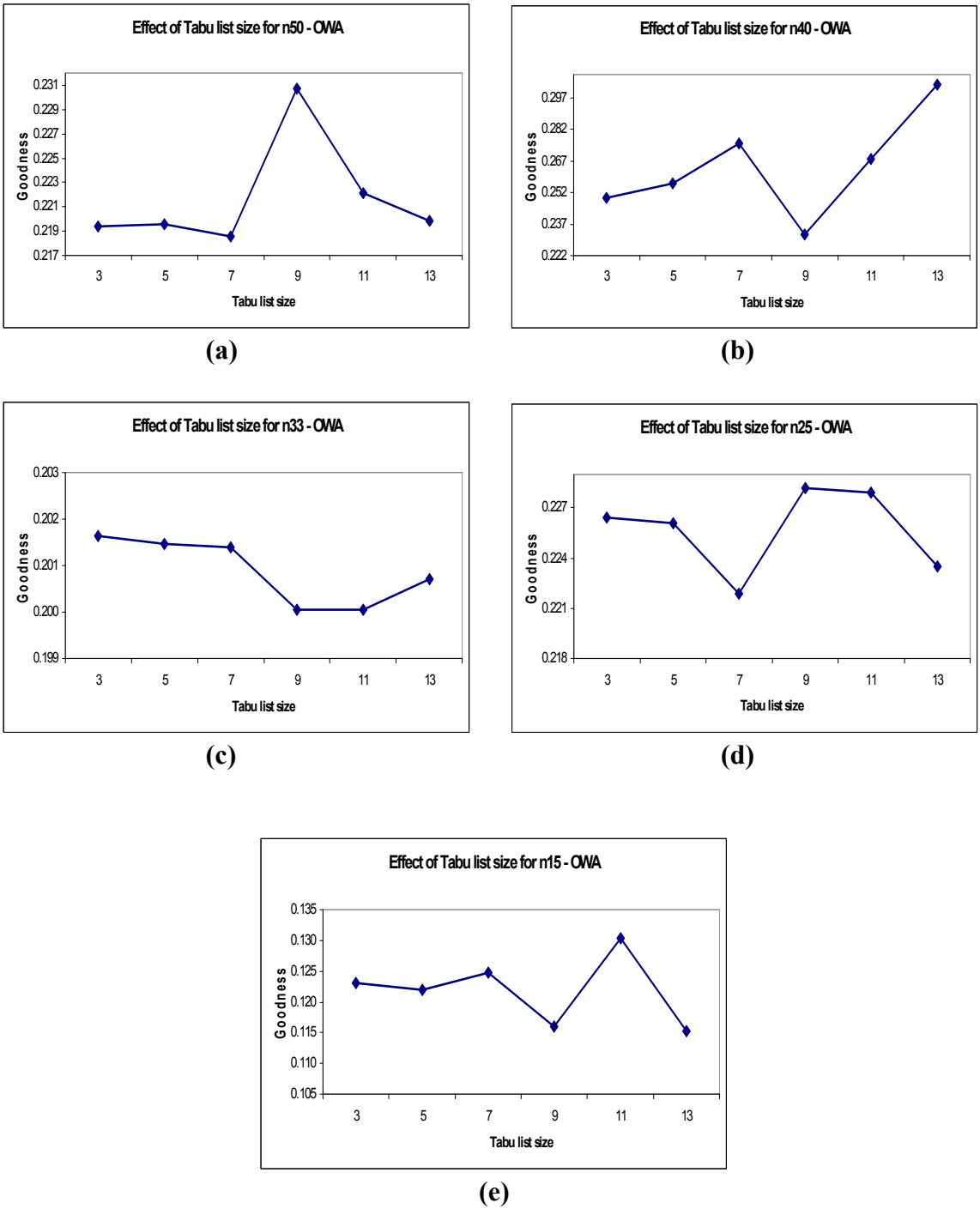


Figure 6.2: Plots of average overall goodness versus tabu list size for FSimE using the OWA operator for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

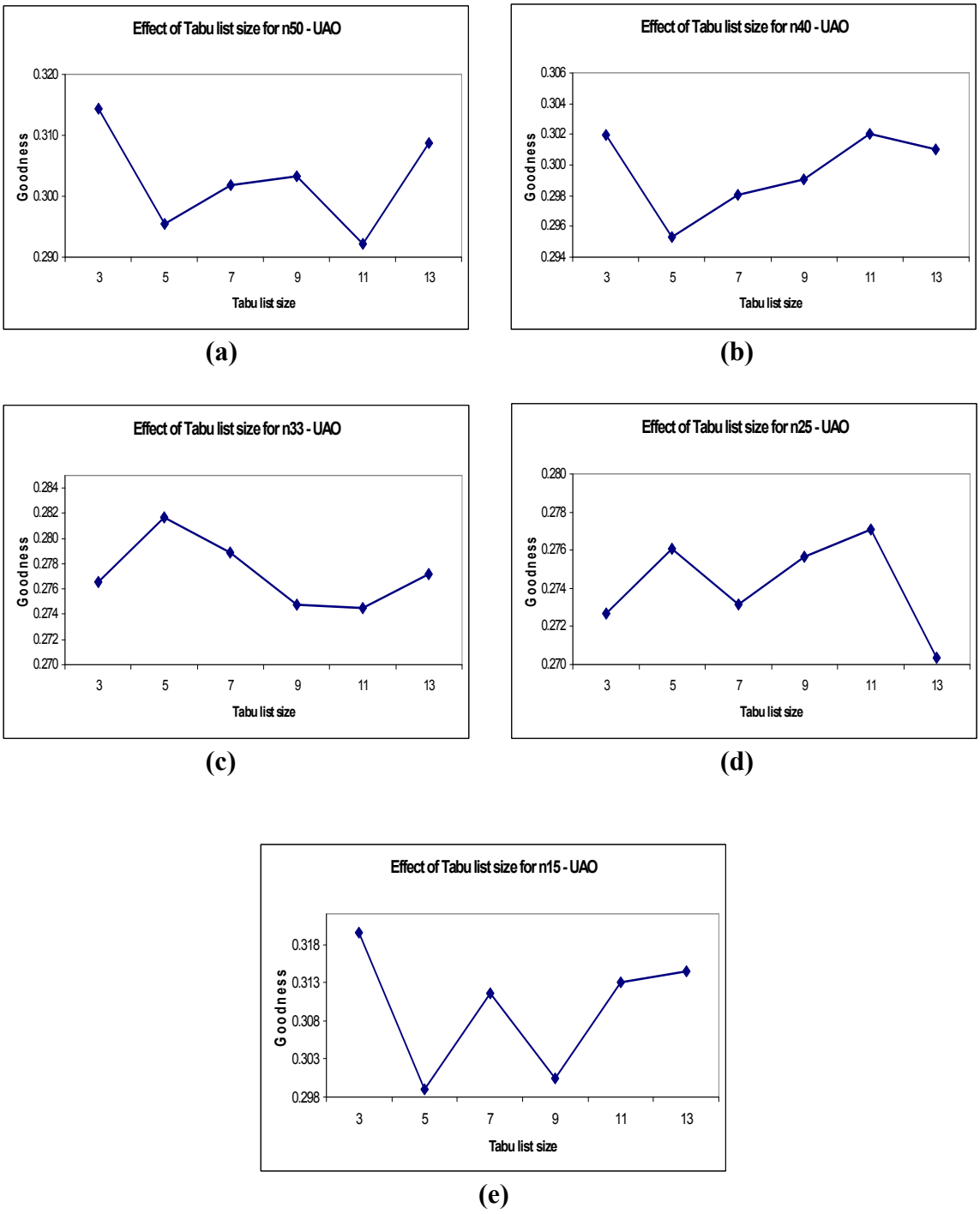


Figure 6.3: Plots average overall goodness versus tabu list size for FSimE using the UAO operator for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

quality of solution produced by TFSimE. A statistical validation of results using a t-test at 5% significance level suggested that TFSimE produced significantly better results for *n15*, *n25*, and *n33*, while FSimE showed better performance for *n50*. For *n40*, the improvement achieved by TFSimE was not statistically significant.

For UAO, the results in Table 6.4 show that the general trend was somewhat similar to the results in Table 6.4 as far as the relationship of bias with test case size is concerned. For FSimE, the best results were obtained when the bias was inversely proportional to the test case size, with the exception of *n15*. With respect to the performance of FSimE and TFSimE, the results were somewhat unclear. A statistical validation using t-test suggested that TFSimE was able to achieve better results than FSimE for *n25*, *n33*, and *n50* with percentage improvement in the range of 2% to almost 35%, while FSimE showed better performance than TFSimE for *n15* with improvement equal to 31.35%. For *n40*, the improvement achieved by TFSimE compared to FSimE was not statistically significant.

From the discussion above, it is observed that TFSimE was able to achieve better results for the majority of cases when OWA was used, and had more or less equivalent performance when UAO was used. This better performance of TFSimE is due to reasons very similar to what was observed for TFStocE in Chapter 5: due to the limited search space, several moves are repeated again and again. Thus, FSimE keeps searching in the same parts of the search space repetitively. For TFSimE, larger parts of the search space are covered because previous moves remain tabu for some time. This causes the algorithm to diversify the search into another subarea. Recall that, during the allocation phase, a new valid link is selected for each removed link. This new link is then also saved in the tabu list. However, the new link may

become "bad" (in terms of the evaluation function) in the following iterations, in which event it is removed. However, the same link may become good again after one or more iterations, but, since it is in the tabu list, it will not be selected again, thus giving room for other links to be considered. This allows TFSimE to cover larger parts of the search space, causing the algorithm to find better solutions than FSimE. The above reasoning is supported by evidence presented earlier in Table 5.6 and the related discussion at the end of Section 5.3.2. The evidence and discussion in Section 5.3.2 are also applicable to the TFSimE algorithm.

Table 6.3: Comparison of FSimE and TFSimE for OWA. TL = Tabu List Size, Time = Run time (in seconds), and % imp = percentage improvement. Statistically significant percentage improvements are in italics.

Case	Bias	FSimE		TFSimE			% imp
		Avg. Overall Goodness	Time	TL	Avg. Overall Goodness	Time	
n15	0.4	0.114 ±0.053	91.4	11	0.130 ±0.039	83.6	*14.70*
n25	0.0	0.175 ±0.010	276.0	9	0.228 ±0.021	256.5	*30.32*
n33	0.2	0.145 ±0.019	492.8	7	0.201 ±0.006	221.3	*38.70*
n40	0	0.290 ±0.082	6067.4	13	0.303 ±0.176	1856.9	4.49
n50	0	0.262 ±0.105	7017.0	9	0.231 ±0.094	3981.1	*-11.94*

Table 6.4: Comparison of FSimE and TFSimE for UAO. TL = Tabu List Size, Time = Run time (in seconds), and % imp = percentage improvement. Statistically significant percentage improvements are in italics.

Case	Bias	FSimE		TFSimE			% imp
		Avg. Overall Goodness	Time	TL	Avg. Overall Goodness	Time	
n15	0.1	0.465 ±0.022	125.6	3	0.320 ±0.052	99.5	*-31.35*
n25	0.3	0.271 ±0.014	236.1	11	0.277 ±0.011	117.0	*2.08*
n33	0.3	0.264 ±0.039	523.5	5	0.282 ±0.012	160.8	*6.85*
n40	0.2	0.298 ±0.045	2272.3	3	0.302 ±0.010	360.6	1.42
n50	0.0	0.235 ±0.037	7025.2	3	0.314 ±0.036	3580.1	*33.94*

6.4 Dynamic Bias

The results in Tables 6.3 and 6.4 point to the fact that a proper bias value is highly dependent on the test case under consideration. Since it is computationally expensive to find the best bias value by a process of trial-and-error, it will be more efficient if the value of the bias is dynamically adjusted. A dynamic bias holds the following advantages:

1. The bias value is not arbitrarily selected, and no trial runs are required to find the best bias value. The bias value automatically adjusts according to the problem state, thus saving the time and effort spent by the user in finding the best bias value by trial-and-error.

2. For bad quality solutions, the average overall goodness of solution and of the links are low, forcing the algorithm to perform a significant number of moves, and thus resulting in large perturbations. To prevent this, the algorithm would dynamically adjust the bias to a high value. This ensures that the size of the selection set is not excessively large, since only links with a very high goodness value will be selected for removal. A selection set of small size will prevent the algorithm from making moves with large step sizes.

3. For good quality solutions, the average link goodness and overall goodness is high, and there is no need to inflate the overall goodness of the solution. Therefore, a low bias value is used. A low bias value will allow the algorithm to have a selection set of a sufficient number of links. This in turn will allow the algorithm to perform a sufficient number of moves, thus enabling the algorithm to escape local minima, and protecting the algorithm from early convergence.

Considering the above points, it is proposed that at each iteration, t, the bias be calculated as:

$$\mathrm{B(t)} = 1 - \left(\frac{1}{L} \sum_{i=1}^{L} g_{l_i}(t) \right)^{\mu} \tag{6.5}$$

where $B(t)$ is the bias at iteration t, $g_{l_i}(t)$ is the goodness of all links in the solution at iteration t, L is the total number of links in the solution at iteration t, and μ is the overall goodness of the solution at time t. From Equation (6.5), the value of the bias is a function of average goodness of links present in the current solution raised to power of overall goodness (solution membership), μ, of the solution. The above strategy to dynamically adjust the bias value was incorporated into the TFSimE algorithm. The resulting algorithm is referred to as the DTFSimE algorithm.

Tables 6.5 and 6.6 respectively summarize the results for the FSimE and DTFSimE algorithms. It is very clear from the tables that DTFSimE demonstrated a much superior performance compared to FSimE. For OWA, Table 6.5 shows a significant improvement in the overall goodness for DTFSimE compared to FSimE. Improvements are in the range of 17% to 82%. Statistical testing with a t-test also suggests that DTFSimE produced significantly better results than TFSimE for all test cases. Likewise, the results for UAO as given in Table 6.6 show that DTFSimE produced statistically better overall goodness values (validated by the t-test) for *n25*, *n33*, and *n50*, with improvements in the range of 10% and 20%. For *n15* and *n40*, FSimE had slightly better results than DTFSimE, but these results were not statistically significant.

Table 6.5: Comparison of FSimE and DTFSimE for OWA. Time = Run time (in seconds). % improvement is for DTFSimE compared to FSimE. Statistically significant improvement is in italics.

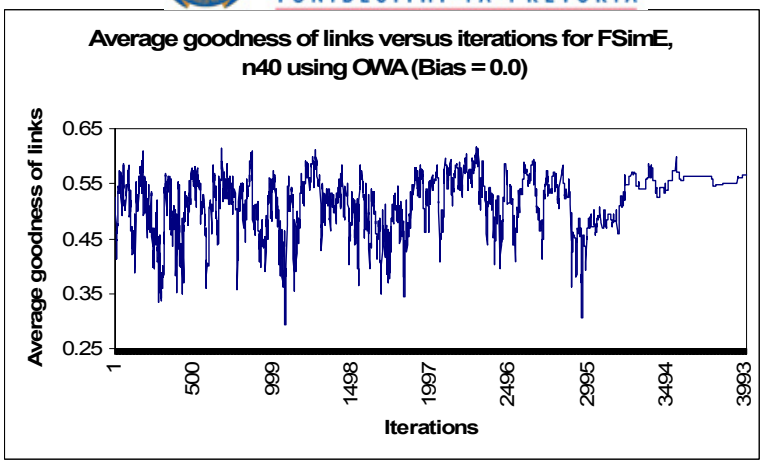
Case	Bias	FSimE		DTFSimE		% improvement
		Avg. Overall Goodness	Time	Avg. Overall Goodness	Time	
n15	0.4	0.114 ±0.053	91.4	0.206 ±0.052	134.2	*81.19*
n25	0.0	0.175 ±0.010	276.0	0.240 ±0.008	354.2	*36.87*
n33	0.2	0.145 ±0.019	492.8	0.229 ±0.061	1080.7	*57.43*
n40	0	0.290 ±0.082	6067.4	0.340 ±0.122	2861.0	*17.26*
n50	0	0.262 ±0.105	7017.0	0.350 ±0.143	7042.8	*33.71*

Table 6.6: Comparison of FSimE and DTFSimE for UAO. Time = Run time (in seconds). % improvement is for DTFSimE compared to FSimE. Statistically significant improvement is in italics.

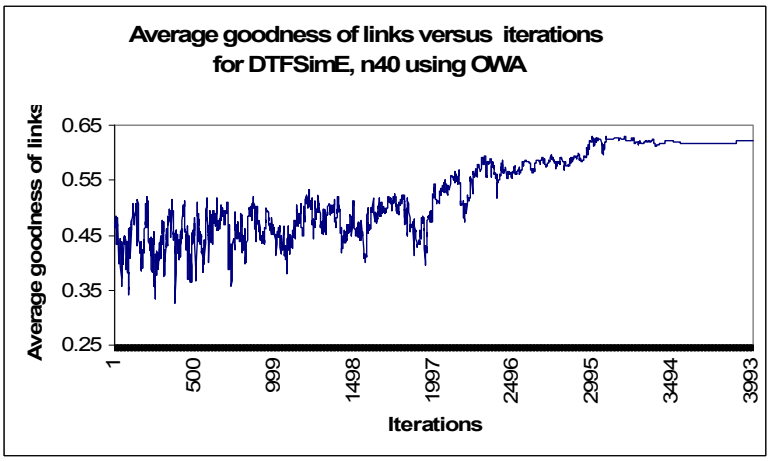
Case	Bias	FSimE		DTFSimE		% improvement
		Avg. Overall Goodness	Time	Avg. Overall Goodness	Time	
n15	0.1	0.465 ±0.022	125.6	0.446 ±0.061	116.9	-4.14
n25	0.3	0.271 ±0.014	236.1	0.301 ±0.006	312.9	*10.72*
n33	0.3	0.264 ±0.039	523.5	0.303 ±0.004	775.1	*14.99*
n40	0.2	0.298 ±0.045	2272.3	0.297 ±0.122	2526.3	-0.26
n50	0.0	0.235 ±0.037	7025.2	0.281 ±0.000	5013.3	*19.84*

The superior performance of DTFSimE is due to two factors:

1. **The dynamic bias**: When a static bias is used, the FSimE algorithm does not have the flexibility to adjust itself according to the problem state. The bias value remains fixed throughout the execution of the algorithm, without considering the average goodness of links. Note that, in the early stages of the algorithm, the average goodness of links is expected to be low, while it tends to be high in later stages. The static bias does not fully account for this behavior, whereas the dynamic bias adapts itself accordingly. To elaborate



(a)



(b)

Figure 6.4: Plots of average goodness of links versus iterations for n40 using OWA obtained with (a) FSimE (with bias = 0.0) (b) DTFSimE

on this, consider the plots in Figures 6.4(a) and (b) which respectively show average goodness of links for typical runs of fixed bias FSimE and dynamic bias DTFSimE for *n40* using OWA. A comparison of the two plots show that the average goodness of links in FSimE varies significantly throughout the execution of the algorithm as is apparent from Figure 6.4(a). In other words, in FSimE, a number of the links are not able to find their optimum positions, and, even if they do, these links are removed from their optimum positions

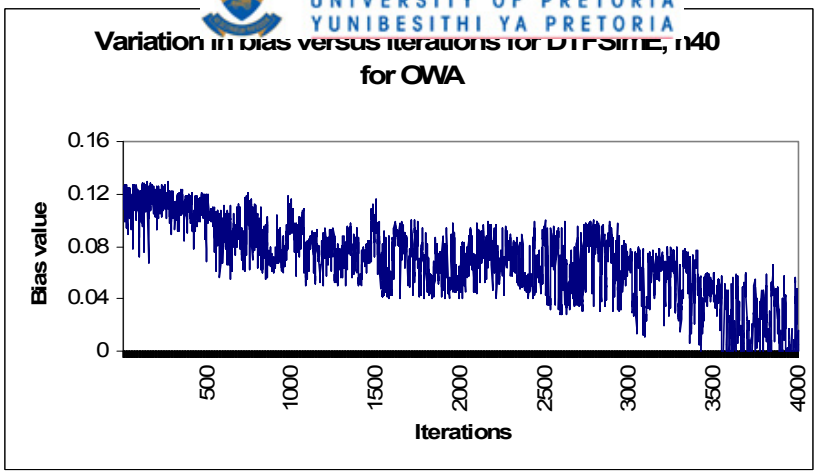


Figure 6.5: Plots of variation in bias versus iterations for n40 using OWA obtained with DTFSimE

in the following iterations. The impact of this behavior is that the FSimE algorithm cannot converge to a sub-optimal solution within the given time frame. On the other hand, the average goodness of links in DTFSimE increases smoothly towards higher values as depicted in Figure 6.4(b). Towards the end of the execution, the variation in average goodness of links in DTFSimE is almost negligible, i.e. most of the links have found their optimal positions. This suggests that the DTFSimE algorithm has converged to a sub-optimal solution. The gradual increase in the average goodness of links for DTFSimE can be associated with the constant change in the bias value. Figure 6.5 plots the variation in the bias with respect to the iterations. It is clear from Figure 6.5 that the bias is initially at a higher value, and with the passage of time the bias decreases. This is logical, since at the beginning of the search a higher bias value would prevent huge number of selection of links (for removal), while towards the end of the search most of the links are already in their optimal positions, and therefore the bias adjusts itself to lower values to allow sufficient perturbations.

2. **The tabu search characteristics**: Tabu search characteristics further enhance the search capability of the DTFSimE algorithm due to the reasons described in the last paragraph of Section 6.3.2.

6.5 Comparison of OWA and UAO Operators

The OWA and UAO operators were used to combine the four design objectives according to Equations (3.11) and (4.1). This section presents a comparison of these two fuzzy operators with respect to their application to DTFSimE. The tabu version of fuzzy SimE was chosen, since the results produced by DTFSimE were relatively better than the other variants, namely FSimE and TFSimE. The comparative analysis focussed on the effect of the four design objectives, namely cost, delay, hops, and reliability with respect to change in the number of nodes. For all the experiments conducted in this study, $\beta = 0.5$ was chosen for the OWA operator in Equation (3.11) and $\nu = 0.5$ was used for the UAO operator in Equation (4.1). A simple linear regression analysis was performed with the number of nodes as the independent variable and each one of the design objectives as a dependent variable. A number of regression models were developed to see how a design objective is affected by increasing the number of nodes when OWA and then UAO are used, and the effect was measured by regression coefficients. The analysis was performed with a confidence level of 95%. The analysis was done using a total of 150 data values, with 30 runs for each test case (*n15* to *n50*).

Results of the comparison are given in Table 6.7, where design objectives are indicated in the first column, regression coefficients for the OWA regression models

Table 6.7: Comparison of OWA and UAO for DTFSimE.

Objective	Regression coefficients		Ratio = $\frac{OWA}{UAO}$	% Gain by UAO	Comment
	OWA	UAO			
Cost	0.947	0.963	0.98	-1.69	UAO performs 1.69 % worse than OWA
Delay	0.336	0.376	0.89	- 11.90	UAO performs 11.9% worse than OWA
Hops	0.628	0.402	1.56	35.99	UAO performs 35.99 % better than OWA
Reliability	-0.86	-0.75	1.15	12.79	UAO performs 12.79 % better than OWA

are given in the second column, and regression coefficients for the UAO regression models are given in the third column. The fourth column provides the ratio of regression coefficients of OWA versus UAO, and the fifth column provides the percentage gain. The ratio in column 4 signifies the rate of increase for a certain objective if the performance of OWA and UAO are compared for that objective. The percentage gain given in column 5 shows the improvement achieved by the UAO operator as compared to the OWA operator. A comment is given in column 6 highlighting this improvement. According to Table 6.7, the regression coefficients for *Cost* are given for OWA and UAO as 0.947 and 0.963 respectively [2]. The corresponding ratio of OWA and UAO is 0.98. The interpretation of this ratio is that, as the number of nodes are increased, the rate at which cost increases using OWA is 0.98 times more than the rate if UAO was used. In other words, the rate of increase of cost with respect to the increase in the number of nodes for OWA is slightly less than that of UAO. Similarly, the percentage gain shows that the performance of UAO is 1.69%

[2] A higher value of the regression coefficient would mean that the rate of increase of cost with respect to increasing number of nodes is high; therefore, a low regression coefficient is desired

worse than that of OWA in terms of controlling the rise in cost when measured against the increment in number of nodes. For the *Delay* objective, UAO had a 11.9% worse performance than OWA. For the *Hops* objective, the OWA/UAO ratio was 1.56, implying a significant performance improvement of 36% by UAO as compared to OWA. For the *Reliability* objective, the regression coefficients have a negative sign, which implies that the number of nodes and reliability are inversely proportional. The OWA/UAO ratio for reliability is 1.15, suggesting that, as the size of the test case increases, the rate at which reliability deteriorates using OWA is 1.15 times faster than if UAO is used. The percentage difference of 12.79% confirms this observation; the UAO is 12.79% better than OWA in terms of controlling the decline in reliability. The analysis of results in Table 6.7 suggests that UAO showed better performance for the *Hops* and *Reliability* objectives, a performance almost equal to that of OWA for *Cost*, and a worse performance for *Delay*.

6.6 Conclusions

This chapter proposed and investigated a fuzzy multi-objective algorithm based on simulated evolution algorithm, namely FSimE. A hybrid variant of FSimE, known as TFSimE, was also proposed. This variant introduced features of tabu search in the *allocation* phase of the FSimE algorithm. The effect of tabu list size was studied, which suggested that the best size is problem dependent, and not fixed at 7 as recommended by Glover [100]. The performance of TFSimE and FSimE was compared. It was shown that the TFSimE generally produced better results than FSimE. This improvement was observed for both the OWA and UAO operators.

A method for dynamically adjusting the value of the bias was proposed. Results showed that a dynamic bias improved the performance of TFSimE for all test cases with respect to OWA and a majority of the test cases for UAO. An empirical analysis showed that the performance of UAO was:

- better than that of OWA for the *number of hops* and *reliability* design objectives,
- similar to OWA for the *cost* objective, and
- worse for the *delay* objective.

The next chapter discusses the fuzzy simulated annealing algorithm which has been tailored for the multi-objective DLAN topology design problem.

Chapter 7

Fuzzy Simulated Annealing for DLAN Topology Design

A fuzzy simulated annealing algorithm (FSA) is proposed in this chapter for DLAN network topology design. Two variants of FSA are also proposed, as described later in the chapter. These variants incorporate characteristics of the tabu search and simulated evolution algorithms into the FSA algorithm. Another modification of FSA is proposed later in this chapter, with the purpose of reducing the user's dependency in setting the value of the Markov chain factor of FSA. The performance of FSA and its variants are empirically evaluated and mutually compared.

7.1 Fuzzy Simulated Annealing Algorithm

The studies in Section 2.4.4 focussed on the use of SA for network design problems with a single objective. Therefore, more research was considered necessary to evaluate the performance of SA in a multi-objective environment. Another aspect of

SA which deserves more attention is its hybridization with other iterative heuristics. Considering the above two issues, this section proposes a fuzzy simulated annealing algorithm for the DLAN topology design problem.

In the fuzzy SA algorithm for the DLAN network topology design, the neighborhood state of a current solution is generated by randomly selecting a link l_i from the topology and replacing l_i with another link l_j not in the topology. The FSA algorithm executes different steps similar to the basic SA algorithm discussed in Section 2.4.4. These steps as adapted for FSA are discussed below in detail.

7.1.1 Initialization

The initial topology (a spanning tree) is generated randomly, while keeping in mind the constraints mentioned earlier. The FSA control parameters are also initialized, i.e. the initial temperature T_0, the cooling rate α_{SA}, the constant β_{SA}, maximum time for the annealing process $MaxTime$ (in terms of number of iterations), and M which represents the time until the next parameter update.

The control parameters have an impact on the convergence of the FSA algorithm. As mentioned in Chapter 2, a suitable value of T_0 should be chosen. A number of approaches have been reported in the literature to find a suitable value of T_0 [33, 126, 145, 167, 202]. In this thesis, the approach presented by Kirkpatrick *et al.* [145] has been adopted. According to this method, the value of T_0 is chosen such that the initial acceptance ratio, $\chi(T_0)$, is close to unity, where

$$\chi(T_0) = \frac{\text{Number of moves accepted at } T_0}{\text{Total number of moves attempted at } T_0} \tag{7.1}$$

In Equation (7.1), a ratio close to unity will allow a high majority of initial moves to be accepted, thus allowing the algorithm to escape local minima. If the ratio is low, then the FSA algorithm will mainly accept good moves and will behave as a greedy algorithm, thus reducing the chances of escaping local minima, and resulting in premature convergence.

As for the other parameters of FSA, a discussion has already been given in Section 2.4.4 with regard to the importance of these parameters as well as the selection of appropriate values for them.

7.1.2 Metropolis Algorithm

The metropolis procedure uses a sub-procedure *neighbor* to perform a move. A move involves removing a randomly selected link between two nodes from the current solution (i.e. topology) and introducing another link between two nodes. A move is represented by a local neighbor $NewZ$ of any given solution Z. Removal of a link and introduction of a new link is exactly the same as described in Section 5.1. This new link is accepted if all constraints are satisfied and the new *overall goodness* fulfils the metropolis criterion. The overall goodness represents the fuzzy value of a solution given by Equation (3.11) or (4.1) and is calculated by function OG in the pseudo-code of the simulated annealing algorithm given in Figure 2.10. If the overall goodness of a new solution, NewZ (where NewZ is the restructured tree), is higher than the overall cost of the current solution Z, then NewZ is definitely accepted. However, if the overall goodness of NewZ is less than the overall goodness of the current solution Z, then NewZ is accepted probabilistically based on the *metropolis criterion* given by $P(random < e^{-\Delta h/T})$. If any of the constraints are violated, or if

the new solution does not pass the metropolis criterion, then the new link is rejected and the original link is restored.

7.1.3 Evaluation of a solution

During the *Metropolis* stage of the algorithm, a move is made which might result in a new solution (if the constraints are satisfied and the new solution is definitely or probabilistically accepted). The overall goodness of the resulting solution is computed by combining the four criteria to be optimized using fuzzy logic, as described in Section 3.4. To form the membership functions of individual objectives, the minimum and maximum values are required for each objective, which have been discussed earlier in Section 3.1.

7.1.4 Stopping Criterion

The algorithm is stopped when a maximum number of iterations is reached. For the purposes of this thesis, the maximum number of iterations was set to 20,000.

7.2 Hybrid Simulated Annealing Algorithms

Apart from the FSA algorithm for DLAN topology design that was presented in Section 7.1, two hybrid variants of the FSA algorithm are proposed in this section. Section 7.2.1 presents a hybrid algorithm incorporating tabu search into the FSA algorithm, while a hybrid algorithm incorporating tabu search and simulated evolution characteristics into the FSA algorithm is presented in Section 7.2.2.

7.2.1 Tabu Fuzzy Simulated Annealing

This thesis proposes a new hybrid iterative search technique, tabu simulated annealing (TFSA), which combines features of TS within the metropolis procedure of FSA. Recall that, in the metropolis procedure, a move is made in which a link is randomly removed from the current solution (i.e. the current topology), and a new feasible link is introduced in the solution. This newly accepted link is saved in a *tabu list*. Thus, the *attribute* is the link itself. The *aspiration criterion* is that, if the link that had been made tabu produces a higher membership value than the current link in the membership function "good topology", then the tabu status of the link is overridden, making the link permanent. This strategy prevents the algorithm from repetitively removing the same link and replacing it with a link of equal or worse goodness.

7.2.2 Evolutionary Tabu Fuzzy Simulated Annealing

The hybrid tabu search FSA algorithm, TFSA, as described in Section 7.2.1 is modified to incorporate the evaluation phase of the SimE algorithm. The resulting algorithm is referred to as TEFSA. In TEFSA, the evaluation phase considers a solution as a set of movable elements, where a movable element is a link. Each link, l_i, has an associated *goodness* (fitness) measure, g_i, in the interval [0,1], defined as

$$g_i = \frac{O_i}{C_i} \tag{7.2}$$

where O_i is the estimate of the optimal cost of the link l_i and C_i is the actual cost of l_i in its current location. The fuzzy evaluation measure proposed in Rule 2 in

Section 6.1.2 is used to evaluate the goodness of a link according to Equation (6.1).

The main objective of developing TEFSA is to obtain a better solution than FSA and TFSA, with the same runtime for all algorithms. Recall that the metropolis procedure of TFSA randomly removes a link, and a new link is introduced to maintain the topology. There is a non-zero probability that this removed link was already placed in its optimum (or near-optimum) position. If this process continues blindly for other links with near optimum position, then it will take a significant amount of time for the algorithm to converge. Rather than having a 'blind move', an 'intelligent move' would be more appropriate, where a link is removed based on its quality. Thus, a link having low quality will have a higher chance of getting removed from its current position, while a link with a high quality (i.e. in a near-optimum position) will have a lower probability of being removed. On the other hand, with the 'blind move' approach, each link in the topology has an equal probability of being removed from its current position, irrespective of its quality. The 'intelligent move' approach has been incorporated in TEFSA, with all links being evaluated using Equation (6.1). After evaluation of all links, a set of links is randomly chosen. The size of this set can vary from one link to all links evaluated. A large-sized set will allow a higher number of low-quality links, but at the expense of more runtime. On the other hand, a small-sized set will have less runtime, but many of the low-quality links may not be a part of the set. In any case, the worst link from this set is chosen and removed from the topology. The new link is then inserted as described earlier, subject to constraints and the tabu criteria.

7.3 Results and Discussion

The FSA, TFSA, and TEFSA algorithms proposed in this thesis were experimentally tested using the five test cases. A variety of experiments were conducted to evaluate the performance of the algorithms. The first set focussed on comparisons between the three variants of SA. In the second set, a comparative analysis of OWA versus UAO was done. Details of these experiments are discussed below.

The control parameters of FSA have an impact on the performance of the algorithm, as discussed in Section 2.4.4. Inappropriate values for these parameters may result in non-optimal solutions. After trials with several values for the initial temperature, a value of T_0=1000 was found to be the most appropriate since the value satisfied the ratio in Equation (7.1) to be near unity. To find a good value for α_{SA}, the following values were considered: 0.6, 0.75, 0.85, 0.95, and 0.99. Two values of the Markov chain with M =10 and M = 30 were tried, with the annealing constant set at β = 1.1. This value of β was taken since a gradual change in the value of M was desired. As shown in Tables 7.1 and 7.2, M = 30 and M = 10 produced results of equal quality, as validated by the t-test. In some cases, M = 10 produced slightly better results (marked in italics). Therefore, a value of M = 10 was used for all experimentation. For each variant of SA, 30 runs were executed for each test case, and the average and standard deviation of overall goodness of best solutions were calculated. For TFSA, different tabu list sizes as described in Section 7.3.1 were tried. The best tabu list size obtained for TFSA was also used for TEFSA. For fair comparison of the performance, the same initial solution was used for all runs of the three variants for each test case. The three algorithms were

run for the same amount of absolute time.

Table 7.1: Summary of best overall goodness with Markov chain size M = 10 and M = 30 using the OWA operator for FSA. % improvement shows improvement achieved by M = 10 with reference to M = 30. Statistically significant improvement is in italics.

Case	α_{SA}	$M = 10$ Avg. Goodness	$M = 30$ Avg. Goodness	% improvement
n15	0.95	0.206 ±0.035	0.203 ±0.034	1.33
n25	0.75	0.414 ±0.115	0.346 ±0.101	*19.92*
n33	0.85	0.623 ±0.369	0.543 ±0.380	*14.70*
n40	0.75	0.244 ±0.051	0.253 ±0.040	-3.65
n50	0.75	0.238 ±0.066	0.237 ±0.044	0.42

Table 7.2: Summary of best overall goodness with Markov chain size M = 10 and M = 30 using the UAO operator for FSA. % improvement shows improvement achieved by M = 10 with reference to M = 30. Statistically significant improvement is in italics.

Case	α_{SA}	$M = 10$ Avg. Goodness	$M = 30$ Avg. Goodness	% improvement
n15	0.99	0.335 ±0.003	0.335 ±0.003	-0.03
n25	0.75	0.345 ±0.034	0.335 ±0.034	2.86
n33	0.75	0.351 ±0.092	0.394 ±0.087	*-12.43*
n40	0.85	0.382 ±0.068	0.368 ±0.061	3.69
n50	0.85	0.350 ±0.053	0.343 ±0.036	1.97

7.3.1 Effect of Tabu List size

The effect of the tabu list size was investigated for the TFSA algorithm with tabu list sizes of 3, 5, 7, 9, 11, and 13. The effect was evaluated for both the OWA and UAO operators. As for the previous chapters, the purpose of this investigation is to see if the statement made by Glover [100] (mentioned in Section 5.3.1) can be confirmed. Tables 7.3 and 7.4 summarize the average overall goodness for the

best solutions for each test case with different tabu list sizes for OWA and UAO respectively.

It is observed in Table 7.3 that the size of tabu list that is related to best overall goodness varies for each case when OWA is used. A graphic illustration of the variation in average overall goodness with respect to different sizes for OWA is given in Figure 7.1. Table 7.3 also provides the percentage difference in the average overall goodness of the given size when compared with size 7. For example, for case *n33*, the average overall goodness obtained with size 9 was 19.35% better than that obtained with size 7. It appears from the results in Table 7.3 that, in general, Glover's first observation that tabu list size of 7 does not always produce the best results seem to be proven true. This observation is confirmed through validation of results using the t-test. The t-test was performed (at 5% significance level) to test the hypothesis whether the two averages (i.e. the average overall goodness obtained with a tabu list size of XYZ and that of 7) were significantly different from each other. Percentage improvements which are statistically significant are shown in italics. The t-test results signify two important issues. First, in general there is no hard evidence that the results produced by a tabu list size of 7 were statistically more significant than the results produced by other sizes. This observation is obvious from the results of *n15*, *n25*, *n33*, and *n40*, where size 7 failed to produce statistically better results than any other size. Even for test case *n50*, size 7 did not result in significant improvement as a whole; there is only one instance where size 7 produced better results than size 3. Second, there are instances where sizes other than 7 achieved statistically more significant results than results obtained with size 7. These instances include a tabu list size of 9 for *n25*, and the sizes 3, 9, and 13 for *n33*. Furthermore, Glover's

observation that size of the best tabu list increases as the test case size increases was not found to be true.

As for UAO, a trend similar to that of OWA was observed, where the size for best results varied with each test case. As observed from Table 7.4, there was only one instance of *n15* where size 7 produced statistically better results than size 3, with an improvement of 0.93%. For other test cases, size 7 failed to produce statistically significant results compared to other sizes. In some instances, tabu list sizes other than 7 produced statistically better results. Such instances include *n25*, where tabu list sizes of 3 and 9 respectively produced 4.71% and 6.7% better overall goodness than size 7. Another instance is *n40* where size 9 produced 11.40% better results than size 7. A graphic illustration of the variation in average overall goodness with respect to different tabu list sizes for UAO is given in Figure 7.2.

The conclusion from the above discussion and results is that it is not true that a tabu list size of 7 will always produce the best results for any problem; the size of the tabu list is problem-dependent. Moreover, the best tabu list size also does not necessarily increase with increasing the size of test case.

7.3.2 Comparison of FSA, TFSA, and TEFSA

Table 7.5 summarizes the results of 30 runs obtained by TEFSA, TFSA, and FSA for the OWA operator. The corresponding run time is given in Table 7.6. For each test case, different values of α_{SA} were tried, and the values which gave the best results are reported in Table 7.5. The results show that the value of α_{SA} is not proportional to the size of the test case. For example, the best value of α_{SA} for *n15* is 0.95, which is a small test case. A value of $\alpha_{SA} = 0.85$ gave best results for

Table 7.3: Effect of tabu list size on the quality of overall goodness for TFSA using OWA. Run time is in seconds. Statistically significant improvement is in italics. NA = Not Applicable (since size 7 was used as the reference for comparison).

Test Case	Tabu list size	Avg. Overall goodness	Run time	% Improvement
n15	3	0.257 ± 0.041	53.9	-2.67
	5	0.257 ± 0.041	74.1	-2.37
	7	0.264 ± 0.042	75.0	NA
	9	0.265 ± 0.048	77.2	0.43
	11	0.265 ± 0.047	61.1	0.69
	13	0.250 ± 0.045	62.7	-5.31
n25	3	0.502 ± 0.114	240.8	5.09
	5	0.464 ± 0.134	242.1	-2.88
	7	0.478 ± 0.129	241.8	NA
	9	0.532 ± 0.106	246.1	*11.36*
	11	0.444 ± 0.141	247.1	-7.00
	13	0.485 ± 0.114	265.0	1.44
n33	3	0.526 ± 0.352	478.3	*11.38*
	5	0.490 ± 0.358	475.7	3.87
	7	0.472 ± 0.359	492.2	NA
	9	0.563 ± 0.362	509.2	*19.35*
	11	0.501 ± 0.354	525.3	6.11
	13	0.523 ± 0.350	567.1	*10.78*
n40	3	0.321 ± 0.078	1457.4	-0.42
	5	0.334 ± 0.099	1462.6	3.72
	7	0.322 ± 0.102	1491.4	NA
	9	0.316 ± 0.076	1509.0	-1.81
	11	0.309 ± 0.085	1542.4	-3.98
	13	0.310 ± 0.089	1544.6	-3.90
n50	3	0.254 ± 0.088	3208.1	*-15.38*
	5	0.279 ± 0.076	3310.8	-6.85
	7	0.300 ± 0.068	3308.9	NA
	9	0.316 ± 0.080	3289.6	5.52
	11	0.275 ± 0.082	3342.0	-8.34
	13	0.296 ± 0.092	3361.9	-1.41

Table 7.4: Effect of tabu list size on the quality of overall goodness for TFSA using UAO. Statistically significant improvement is in italics. NA = Not Applicable (since size 7 was used as the reference for comparison).

Test Case	Tabu list size	Avg. Overall goodness	Run time	% Improvement
n15	3	0.340 ± 0.008	53.3	0.15
	5	0.336 ± 0.004	73.7	*-0.93*
	7	0.339 ± 0.008	76.3	NA
	9	0.338 ± 0.006	77.2	-0.46
	11	0.339 ± 0.006	61.3	-0.14
	13	0.338 ± 0.005	65.4	-0.49
n25	3	0.368 ± 0.032	594.0	*4.71*
	5	0.360 ± 0.040	227.2	2.58
	7	0.351 ± 0.043	229.6	NA
	9	0.375 ± 0.048	239.1	*6.70*
	11	0.352 ± 0.039	234.1	0.23
	13	0.352 ± 0.032	234.8	0.25
n33	3	0.321 ± 0.068	772.3	-5.78
	5	0.328 ± 0.080	754.2	-3.86
	7	0.341 ± 0.094	759.5	NA
	9	0.339 ± 0.084	760.0	-0.59
	11	0.316 ± 0.074	848.2	-7.31
	13	0.337 ± 0.099	850.1	-1.05
n40	3	0.450 ± 0.057	944.8	*6.88*
	5	0.421 ± 0.070	944.9	0.04
	7	0.421 ± 0.070	1004.0	NA
	9	0.469 ± 0.036	1032.6	*11.40*
	11	0.431 ± 0.065	1061.9	2.44
	13	0.440 ± 0.063	1088.9	4.54
n50	3	0.356 ± 0.048	2050.9	-3.83
	5	0.369 ± 0.062	2104.9	-0.30
	7	0.370 ± 0.061	2162.1	NA
	9	0.393 ± 0.070	2214.6	6.13
	11	0.379 ± 0.065	2270.0	2.30
	13	0.370 ± 0.062	2322.5	-0.11

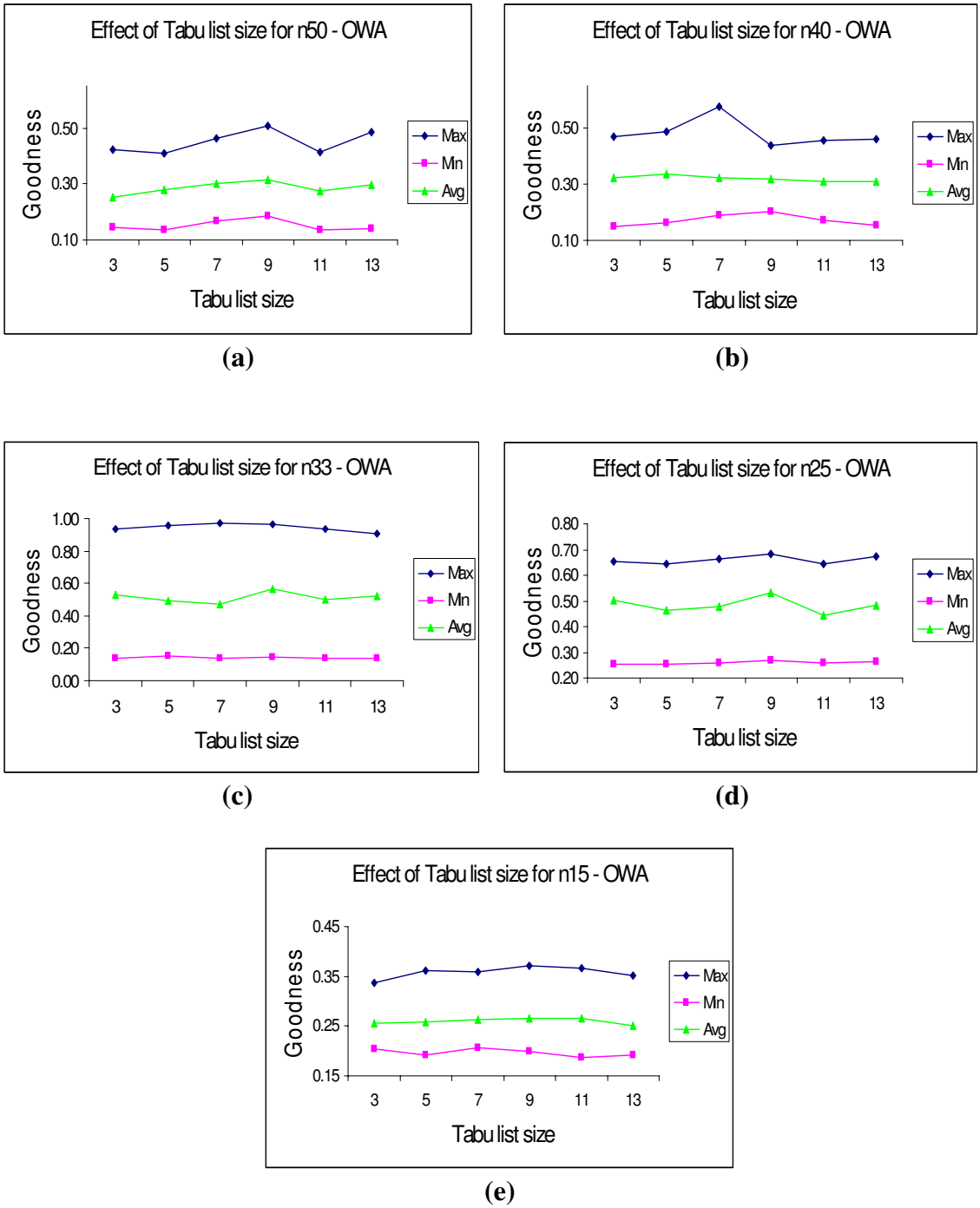
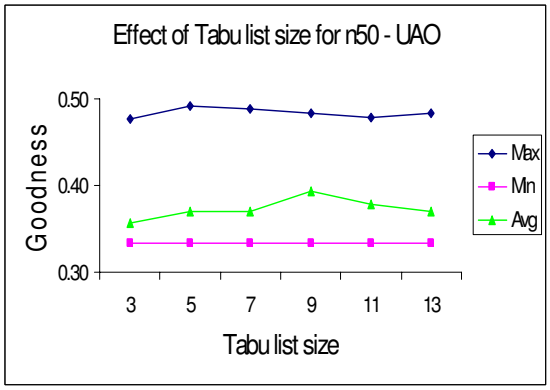
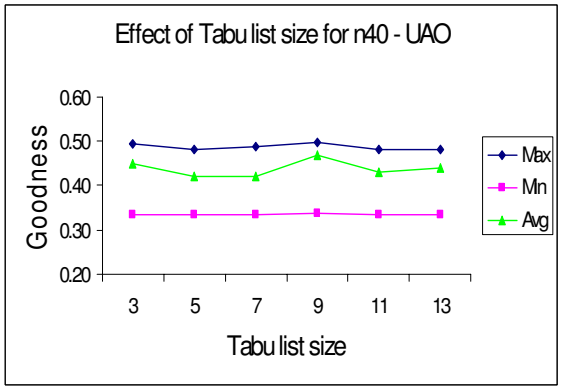


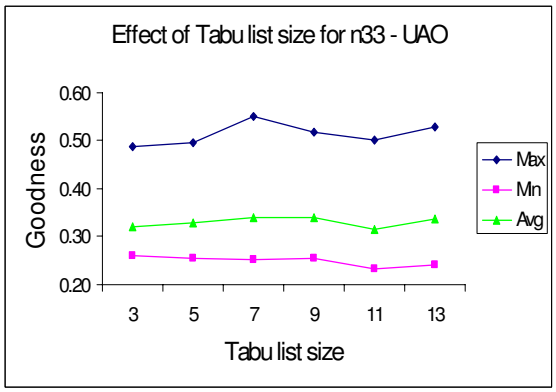
Figure 7.1: Plots of maximum, minimum, and average values of membership function "Good topology" versus tabu list size using the OWA operator for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15



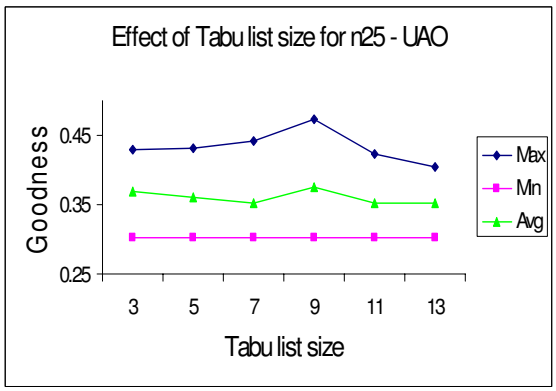
(a)



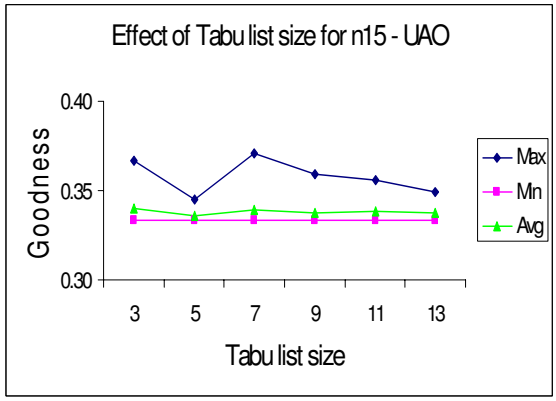
(b)



(c)



(d)



(e)

Figure 7.2: Plots of maximum, minimum, and average values of membership function "Good topology" versus tabu list size using the UAO operator for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

Table 7.5: Summary of overall goodness and percentage improvement with OWA for FSA, TFSA, and TEFSA. TL = Tabu list size, imp = percentage improvement. Statistically significant improvement is in italics.

Case	α_{SA}	FSA Goodness	TFSA		TEFSA Goodness	% imp TEFSA vs FSA	% imp TEFSA vs TFSA	%imp TFSA vs FSA
			TL	Goodness				
n15	0.95	0.206±0.04	9	0.265±0.05	0.389±0.06	*89.14*	*47.13*	*28.56*
n25	0.75	0.391±0.12	9	0.53±0.106	0.500±0.13	*27.81*	-6.11	*36.12*
n33	0.85	0.646±0.37	9	0.563±0.36	0.429±0.19	*-33.56*	-23.79	-12.82
n40	0.75	0.231±0.05	7	0.322±0.10	0.489±0.08	*111.56*	*51.68*	*39.48*
n50	0.75	0.198±0.07	9	0.316±0.08	0.329±0.12	*66.39*	4.10	*59.83*

Table 7.6: Average run time (in seconds) of algorithms in Table 7.5.

Test Case	Run time		
	FSA	TFSA	TEFSA
n15	89.0	89.2	89.5
n25	314.4	315.1	314.8
n33	765.1	765.2	764.7
n40	1498.8	1498.4	1499.5
n50	4295.8	4296.6	4295.4

n33 which is a medium size case. However, the best value of α_{SA} was 0.75 for *n25*, *n40*, and *n50*. Notice that *n25* is a small size test case, while *n40* and *n50* are large size test cases. As for the UAO operator, Table 7.7 reports similar trends as that of OWA with respect to α_{SA}. There is no clear trend which would suggest that the test case size is directly or inversely proportional to α_{SA}. Therefore, it can be concluded that the best value of α_{SA} is problem-dependent.

As far as the comparison of TEFSA, TFSA, and FSA with respect to OWA is concerned, it is observed from Table 7.5 that TEFSA obtained the best results among the three variants for most test cases. As per the table, TEFSA had statistically better overall goodness than FSA for test cases *n50*, *n40*, *n25*, and *n15*, as validated

Table 7.7: Summary of overall goodness and percentage improvement with UAO for FSA, TFSA, and TEFSA. TL = Tabu list size, imp = percentage improvement. Statistically significant improvement is in italics.

Case	α_{SA}	FSA Goodness	TFSA		TEFSA Goodness	% imp TEFSA vs FSA	% imp TEFSA vs TFSA	%imp TFSA vs FSA
			TL	Goodness				
n15	0.99	0.335±0.0	7	0.339±0.01	0.365±0.02	*9.16*	*7.68*	*1.37*
n25	0.75	0.345±0.03	9	0.375±0.05	0.412±0.06	*19.30*	*9.82*	*8.63*
n33	0.75	0.339±0.09	7	0.341±0.09	0.411±0.07	*21.14*	*20.55*	0.49
n40	0.85	0.374±0.07	9	0.469±0.04	0.470±0.08	*25.72*	0.30	*25.34*
n50	0.85	0.350±0.05	9	0.388±0.07	0.374±0.05	*6.99*	-3.48	*10.85*

Table 7.8: Average run time (in seconds) of algorithms in Table 7.7.

Test Case	Run time		
	FSA	TFSA	TEFSA
n15	88.0	88.3	88.5
n25	321.3	322.1	322.4
n33	757.6	758.5	757.0
n40	1564.5	1565.6	1564.4
n50	3485.3	3484.6	3485.6

by the t-test (performed at 5% significance level). However, for *n33*, FSA obtained statistically better results than TEFSA. As for comparison of TEFSA and TFSA, results in Table 7.5 suggest that TEFSA produced results that were statistically significantly better for *n15* and *n40*. For *n25*, *n33*, and *n50*, TEFSA and TFSA had results of equal quality, as validated by the t-test. All in all, it can be fairly claimed that TEFSA performed better than the other two variants when OWA was used.

The above observations are further supported by Figure 7.3. In this figure, the average performance of the three algorithms is compared by plotting the frequency of solutions in different membership (i.e. overall goodness) range. The plots for *n50*, *n40* and *n15* show that TEFSA has more solutions in the higher membership

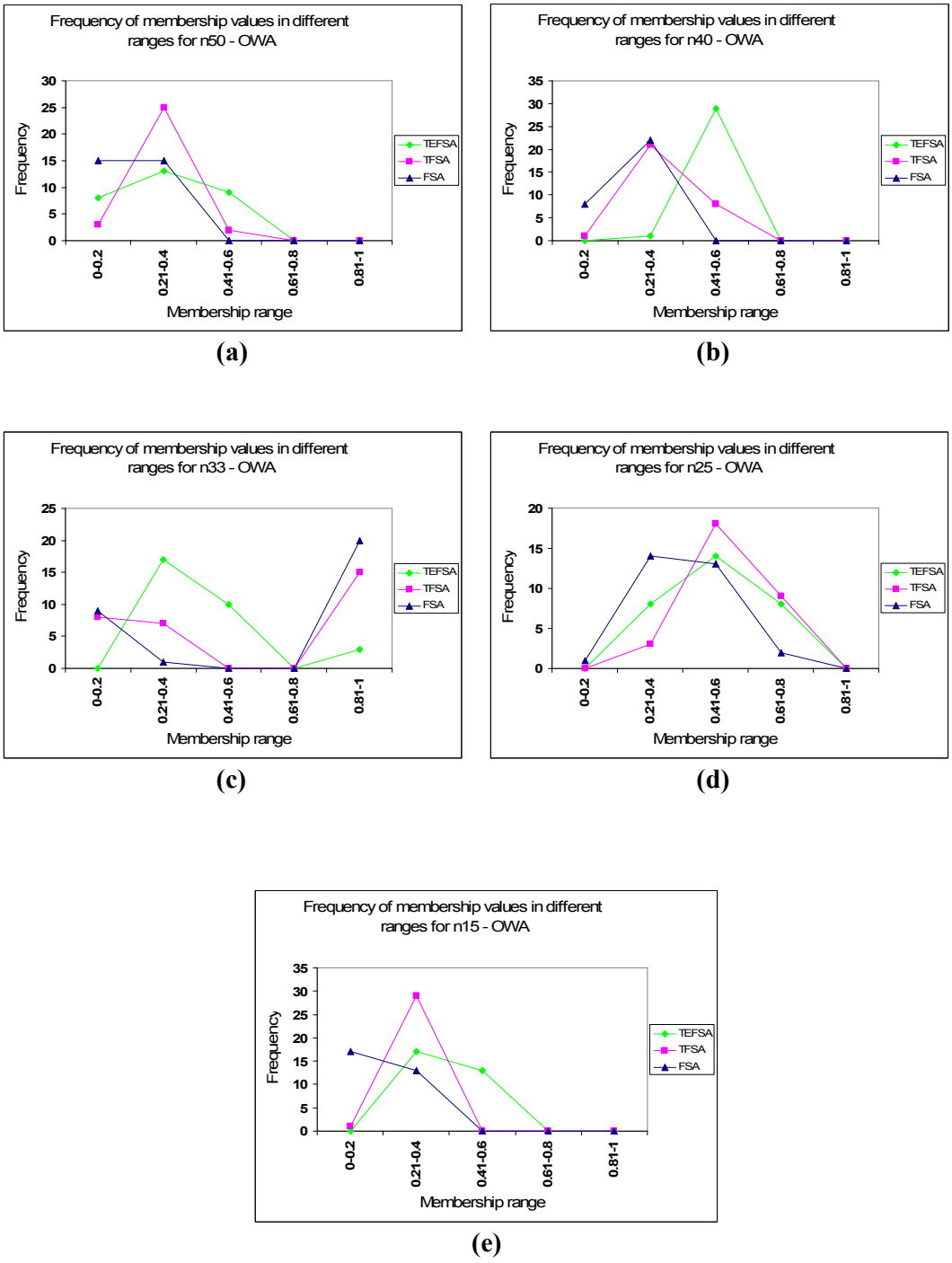


Figure 7.3: Frequency of solution in different membership ranges for function "Good topology" using the OWA operator for FSA, TFSA, and TEFSA for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

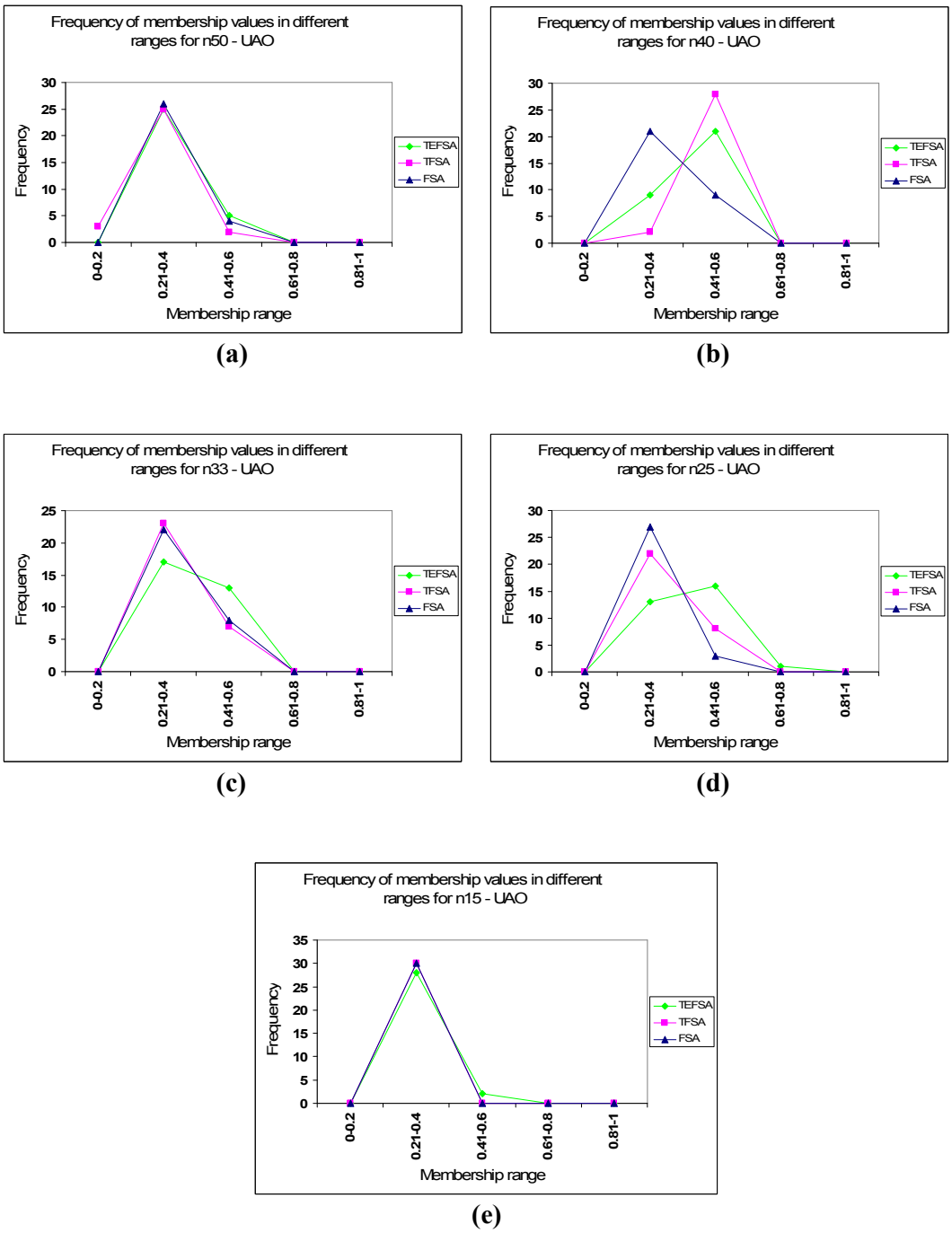


Figure 7.4: Frequency of solution in different membership ranges for function "Good topology" using the UAO operator for FSA, TFSA, and TEFSA for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

range than TFSA and FSA. For example, in Figure 7.3(a), TEFSA has around 10 solutions in the goodness range $0.41 - 0.6$, whereas TFSA and FSA have less than 2 solutions in the same range. A similar pattern is apparent in Figure 7.3(b) where TEFSA has almost all solutions in the range $0.41 - 0.6$. In contrast, TFSA has less than 10 solutions in the same range, while FSA has none. For *n15* (Figure 7.3(e)), the pattern is somewhat similar to Figure 7.3(a). For *n25*, TFSA shows the best performance, which is apparent from plots in Figure 7.3(d), where it is observed that TFSA had more solutions than TEFSA in the ranges $0.41 - 0.6$ and $0.61 - 0.8$. In this plot, FSA has the worst performance since most of its solutions fall in ranges $0.21 - 0.4$ and $0.41 - 0.6$. Finally, for *n33*, the plots show that FSA has more solutions than TEFSA and TFSA in the range $0.81 - 1$, with TEFSA showing the least number of solutions than FSA and TFSA.

With respect to the UAO operator, a mutual comparison of performance of TEFSA, TFSA, and FSA is given in Table 7.7. The table clearly shows that TEFSA was the best for all test cases except *n50*. For this test case, TEFSA and TFSA had almost the same level of performance. This particular observation is also supported by the graphs in Figure 7.4, where it is observed that, for *n50*, TFSA had more solutions in the higher overall range $(0.41 - 0.6)$ than TEFSA and FSA. For other test cases, TEFSA has more solutions than FSA and TFSA in higher goodness ranges, which is clearly visible in test cases *n33*, *n25* and *n40*. The results and observations presented above suggest that TEFSA performed better than the other two variants for UAO.

It is evident from the above discussion that, in general, TEFSA was able to produce results of higher quality than TFSA and FSA, for both OWA and UAO.

This superior performance of TEFSA can be attributed to the fact that TEFSA performs a more efficient exploration of the search space than TFSA and FSA (refer to Section 7.2.2). Recall that, in TEFSA, links with bad goodness have a higher probability of being removed from the topology, while links which are already in "good" positions have a lower probability of being removed. This characteristic of TEFSA makes it different from TFSA and FSA, where all links, whether good or bad, have equal probability of being removed. This explanation is also supported by the results in Table 7.9, where the average goodness of links (AGL) for FSA, TFSA, and TEFSA is given for test cases *n33* and *n40*. The AGL is calculated as the average of 30 runs, and, for each run, the value of AGL associated with the best solution in that run is taken. As observed from Table 7.9, there is clear evidence that the AGL for TEFSA was higher than that for TFSA and FSA. In other words, TEFSA was able to retain links of higher goodness, while TFSA and FSA were not so efficient in retaining links of higher goodness.

As far as a comparison of TFSA and TSA is concerned, it is observed from Tables 7.5 and 7.7 that TFSA produced statistically better results than TSA for all cases, with the exception of *n33*, as depicted in the last columns of Tables 7.5 and 7.7. A possible explanation for the better performance of TFSA in comparison to FSA is that there might be certain moves which are recursively repeated. For example, a link is removed from the topology and placed back in its position. It is quite possible that, in the next iteration, the same link is chosen, removed, and placed back in its location. If this process continues for a certain amount of time, then the algorithm will not improve the solution and execution time will be wasted. However, TFSA prevents this situation from occurring, thus allowing the algorithm to explore more

of the search space and obtain a better quality solution than FSA. This explanation is also supported by the subsequent example where two typical runs for FSA and TFSA for *n40* (while using OWA) were compared in terms of number of repeated moves. The results suggest that out of the 20000 moves performed for each run, FSA had 6842 moves repeated, while TFSA had 4103 moves repeated. That is, FSA repeated 2739 additional moves compared to TFSA. In other words, TFSA had 2739 new moves compared to FSA. These additional new moves allowed TFSA to explore a bigger part of the search space than did FSA, thus resulting in higher quality solutions.

Table 7.9: Average goodness of links for FSA, TFSA, and TEFSA using the OWA operator. AGL represents the average goodness of links. Statistically significant percentage difference is given in italics.

Case	AGL			% Difference	% Difference
	FSA	TFSA	TEFSA	TEFSA vs FSA	TEFSA vs TFSA
n33	0.491	0.597	0.733	*17.71*	*18.61*
n40	0.401	0.484	0.646	*17.12*	*25.09*

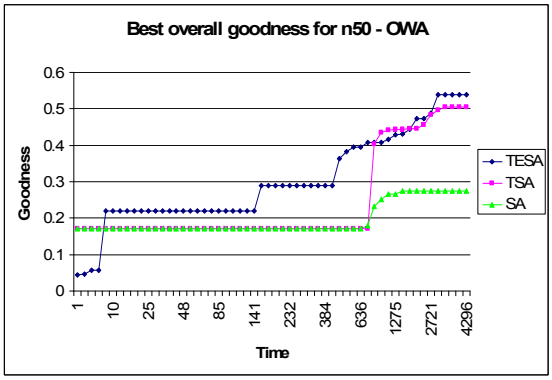
Figure 7.5 illustrates a typical performance pattern over time for TEFSA, TFSA, and FSA, with respect to the best value of membership function "Good topology" (i.e. best overall goodness) using the OWA operator. It is observed that, for all test cases, the overall goodness of TESA generally stays higher than that of FSA and TFSA. For example, for *n50*, the overall goodness of FSA, TFSA, and TEFSA is the same at time 0 (since all three algorithms start with the same initial solution). However, as time progresses, the overall goodness of TEFSA increases at a faster rate than FSA and TFSA. This pattern is also prominent in cases *n25* and *n15*. However, for *n40* and *n33*, there are some time instances where TESA attains higher overall

goodness than TEFSA. As for UAO, Figure 7.6 shows patterns similar to that of OWA. In all test cases, TEFSA either maintains a higher overall goodness or the same level of goodness, when compared with TESA and TSA.

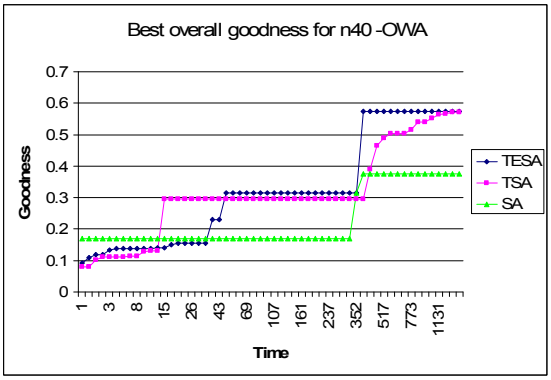
With respect to the performance patterns of TESA and TSA, Figures 7.5 and 7.6 show that TESA generally maintains a higher goodness level than TSA for both OWA and UAO. However, there are a few instances, such as *n50* in Figure 7.5(a) and *n15* in Figure 7.5(e), where both TESA and TSA maintain the same level of overall goodness for a major portion of the runtime.

7.4 Dynamic Markov chain size

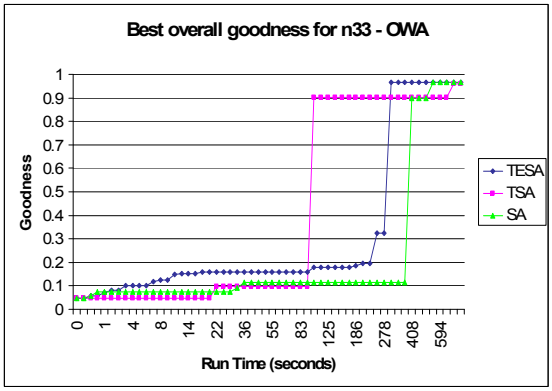
The effect of Markov chain, M, on the convergence of SA algorithm was discussed in detail in Section 2.4.4. A too high value of M increases the algorithm runtime unnecessarily, while a too low value may produce results of low quality. Therefore, it is necessary to find an appropriate value of M. It will be more efficient if the value of the Markov chain is dynamically adjusted. The motivation behind a dynamic value of the Markov chain is that it is computationally expensive to find the best value of M by a process of trial-and-error. A dynamic value of M can be achieved by enhancing the capabilities of the SA algorithm such that the algorithm itself adjusts the value of M, rather than using a pre-defined static value of M. Considering the above points, it is proposed that at each iteration, t, the value of the Markov chain be calculated as:



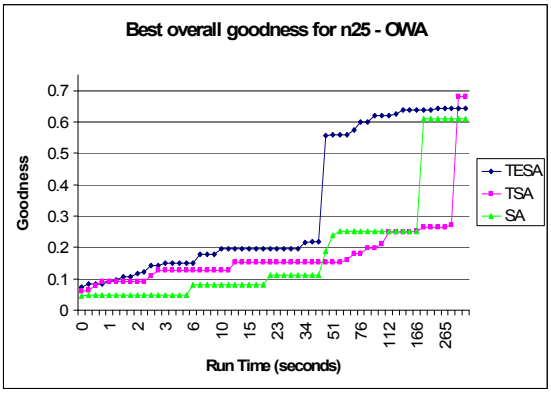
(a)



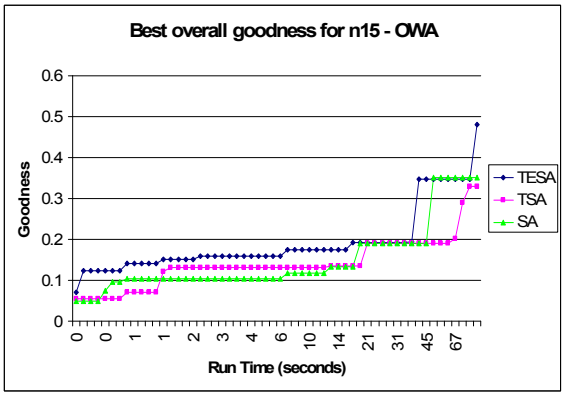
(b)



(c)



(d)



(e)

Figure 7.5: Plots of best value of membership function "Good topology" versus execution time using the OWA operator for FSA, TFSA, and TEFSA for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

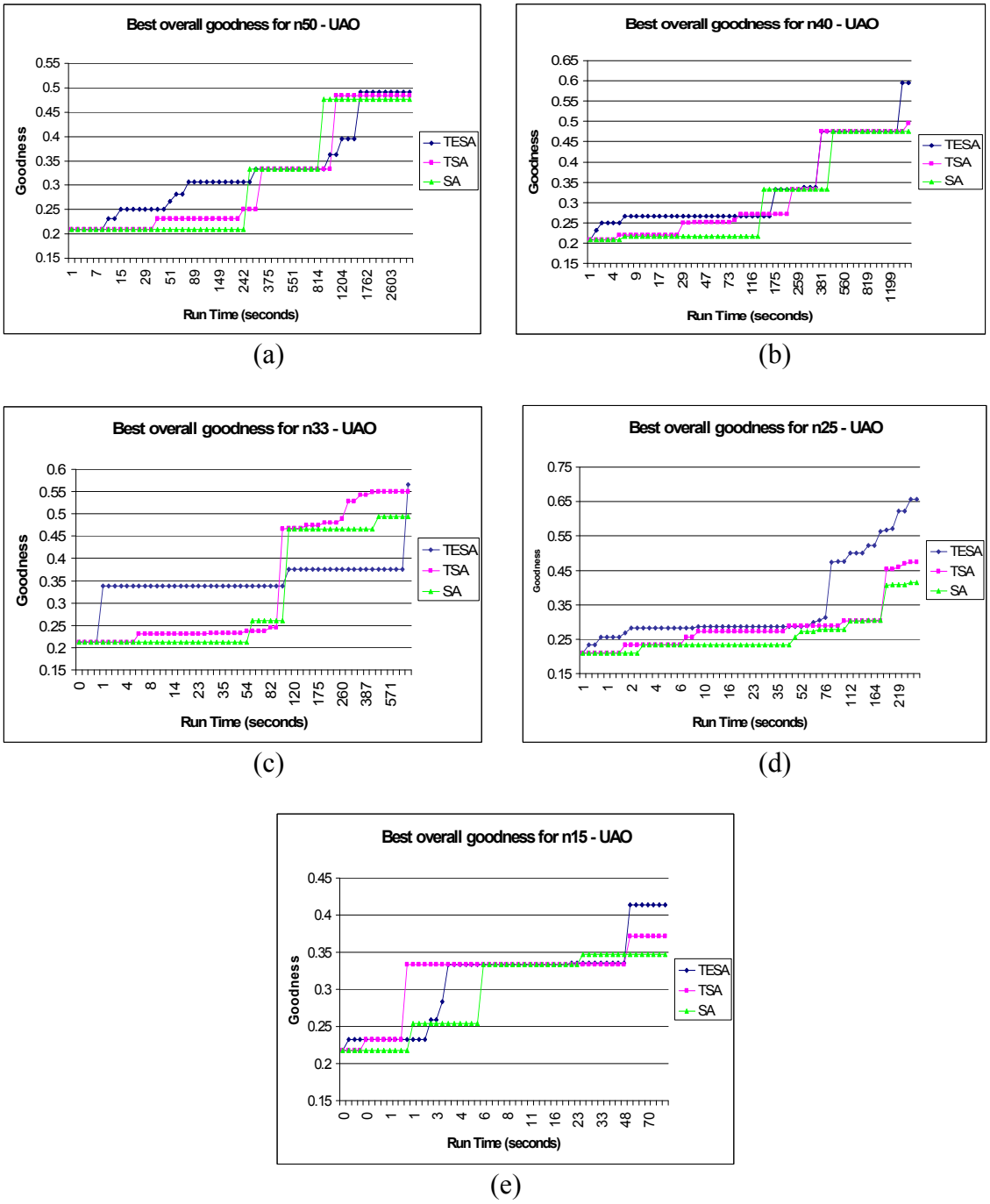


Figure 7.6: Plots of best value of membership function "Good topology" versus execution time using the UAO operator for FSA, TFSA, and TEFSA for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

$$M(t) = \begin{cases} M(t-1) + (\frac{1}{L}\sum_{i=1}^{L} g_{l_i}(t))/\mu & \text{if } \mu \neq 0 \\ M(t-1) + 1 & \text{if } \mu = 0 \end{cases} \tag{7.3}$$

where $M(t)$ is the bias at iteration t, $g_{l_i}(t)$ is the goodness of all links in the solution at iteration t, L is the total number of links in the solution at iteration t, and μ is the overall goodness of the solution at time t. From Equation (7.3), the value of the Markov chain length is updated by a factor which is a function of the average goodness of links present in the current solution divided by the overall goodness (solution membership) μ of the solution, provided that the goodness of solution is not zero. However, the goodness of solution can be zero in an iteration, resulting in $M = \infty$. To avoid this, M is incremented by 1 in the case where the goodness of solution is zero. The above strategy to dynamically adjust the value of M was incorporated into the TEFSA algorithm. The resulting algorithm is referred to as the DTEFSA algorithm.

Tables 7.10 and 7.11 respectively present the results obtained for dynamic M for both the OWA and UAO operators. It is obvious from these tables that the computation of M based on Equation (7.3) did not produce better results than FSA. For the OWA operator, a degradation was observed in the quality of overall goodness produced by DTEFSA in most of the cases, as shown in Table 7.10. This degradation was mainly in the range of 14% to almost 72%. There was an exception to the above trend, where an improvement of 2.13% was observed for DTEFSA with test case *n15*. As for the UAO, a trend similar to that of OWA is observed, where DTEFSA produced results of lesser quality for all cases. The degradation was in the range of 4% to 19.5%. Statistical testing with the t-test also confirmed that the

degradation in results reported in Tables 7.10 and 7.11 was statistically significant.

From the above discussion, a general observation is that DTEFSA was not able to produce better results than FSA. However, the extent of degradation of performance by DTEFSA for UAO was far less than that of OWA. This information gives motivation for improving the proposed dynamic M measure such that the gap observed in the quality of solution is reduced or completely eliminated. A modified approach for DTEFSA can be proposed where different combinations of parameters in Equation (7.3) can be tried.

Table 7.10: Comparison of FSA and DTEFSA for OWA. Time = Run time (in seconds). % imp shows percentage improvement achieved by DTEFSA compared to FSA. Statistically significant results are in italics.

Case	α_{SA}	FSA		DTEFSA			% imp
		Avg. Overall Goodness	Time	TL	Avg. Overall Goodness	Time	
n15	0.95	0.206 ±0.035	89.0	9	0.210 ±0.057	93.9	2.13
n25	0.75	0.391 ±0.145	314.4	9	0.257 ±0.008	300.1	*-34.31*
n33	0.85	0.646±0.363	765.1	9	0.184 ±0.012	791.5	*-71.56*
n40	0.75	0.231 ±0.067	1498.8	7	0.199 ±0.007	1911.6	*-14.24*
n50	0.75	0.198 ±0.046	4295.8	9	0.173 ±0.019	4161.5	*-12.80*

7.4.1 Comparison of OWA and UAO

The relative performance of the OWA and UAO operators was also compared with respect to the four individual objectives, by taking the averages of 30 runs for the three algorithms. With respect to the *Cost* objective, OWA performed slightly better than UAO for FSA, since OWA generated statistically better results for *n25* and *n50*, while UAO generated better results for *n33*, as shown in Table 7.12. With regard to TFSA, again OWA was better than UAO, since OWA generated better

Table 7.11: Comparison of FSA and DTEFSA for UAO. Time = Run time (in seconds). % imp shows percentage improvement achieved by DTEFSA compared to FSA. Statistically significant results are in italics.

Case	α_{SA}	FSA		DTEFSA			% imp
		Avg. Overall Goodness	Time	TL	Avg. Overall Goodness	Time	
n15	0.99	0.335 ±0.003	88.0	7	0.321 ±0.028	98.9	*-3.93*
n25	0.75	0.345 ±0.034	321.3	9	0.303 ±0.007	225.8	*-12.08*
n33	0.75	0.339 ±0.088	757.6	7	0.313 ±0.017	739.1	*-7.65*
n40	0.85	0.377 ±0.066	1564.5	9	0.304 ±0.006	1950.9	*-19.49*
n50	0.85	0.350 ±0.053	3485.3	9	0.285 ±0.013	2496.9	*-18.65*

results for test cases *n50*, *n40* and *n25* as shown in Table 7.13. For TEFSA, UAO performed better than OWA, since UAO produced solutions with lesser costs than OWA for *n50* and *n25*, while for *n15*, *n33*, and *n40*, UAO and OWA produced results of equal quality, as statistically tested and reported in Table 7.14.

For the *Delay* objective, OWA performed relatively better than UAO for all three algorithms. For FSA, OWA produced lesser delays for two cases (*n15* and *n40*) than UAO as shown in Table 7.15. Similarly, for TFSA, OWA showed more improvement for *n50*, *n40*, and *n15*, and equal improvement for *n25* and *n33* as shown in Table 7.16. As for TEFSA, OWA had statistically better results for *n50*, *n40*, and *n25*, and equal performance for *n15*, as shown in Table 7.17.

As far as *number of hops* is concerned, OWA generally performed better than UAO. For example, OWA performed better than UAO for all test cases except *n50*, *n40*, and *n15* for FSA as shown in Table 7.18. For TFSA (Table 7.19), OWA had more improvement for *n50*, *n40*, and *n15*. For TEFSA, OWA again performed better than UAO for test cases *n50*, *n33*, and *n25*, as shown in Table 7.20.

Finally, for the *Reliability* objective, UAO and OWA performed equally well.

Table 7.12: Comparison of OWA and UAO for monetary cost of best solutions of 30 runs for FSA. % imp = percentage improvement achieved by UAO compared to OWA. Statistically significant results are in italics.

Case	OWA	UAO	% imp	Remarks
n15	11422.0 ±1266.5	10850.3 ±1475.9	5.0	UAO and OWA statistically equal
n25	12346.0 ±2549.7	15453.3 ±1831.8	*-25.2*	OWA statistically better
n33	27645.6 ±2730.4	22355.5 ±6419.9	*19.1*	UAO statistically better
n40	34932.9 ±4002.5	35566.0 ±2526.4	-1.8	UAO and OWA statistically equal
n50	45508.6 ±3772.7	50443.8 ±3405.3	*-10.8*	OWA statistically better

Table 7.13: Comparison of OWA and UAO for monetary cost of best solutions of 30 runs for TFSA. % imp = percentage improvement achieved by UAO compared to OWA. Statistically significant results are in italics.

Case	OWA	UAO	% imp	Remarks
n15	9647.7 ±910.3	9494.3 ±1108.6	1.6	UAO and OWA statistically equal
n25	10788.1 ±1448.1	13288.5 ±1747.4	*-23.2*	OWA statistically better
n33	26969.2 ±2641.5	26396.6 ±2622.1	2.1	UAO and OWA statistically equal
n40	31599.9 ±4728.8	35608.5 ±2250.0	*-12.7*	OWA statistically better
n50	45241.5 ±3835.8	49910.9 ±3329.4	*-10.3*	OWA statistically better

Table 7.14: Comparison of OWA and UAO for monetary cost of best solutions of 30 runs for TEFSA. % imp = percentage improvement achieved by UAO compared to OWA. Statistically significant results are in italics.

Case	OWA	UAO	% imp	Remarks
n15	7968.7 ±563.3	7716.7 ±701.6	3.2	UAO and OWA statistically equal
n25	9792.2 ±1255.2	8580.2 ±1736.7	*12.4*	UAO statistically better
n33	17423.3 ±1514.4	18067.1 ±2595.9	-3.7	UAO and OWA statistically equal
n40	21851.2 ±1888.8	21701.9 ±1401.0	0.7	UAO and OWA statistically equal
n50	31642.0 ±6766.2	28239.2 ±3657.3	*10.8*	UAO statistically better

Table 7.15: Comparison of OWA and UAO for delay of best solutions of 30 runs for FSA. % imp = percentage improvement achieved by UAO compared to OWA. Statistically significant results are in italics.

Case	OWA	UAO	% imp	Remarks
n15	0.0031 ±0.0002	0.0036 ±0.0004	*-15.8*	OWA statistically better
n25	0.0036 ±0.0004	0.0037 ±0.0007	-2.8	UAO and OWA statistically equal
n33	0.0076 ±0.0038	0.0083 ±0.0145	-9.7	UAO and OWA statistically equal
n40	0.0061 ±0.0008	0.0095 ±0.0109	*-55.4*	OWA statistically better
n50	0.0075 ±0.0073	0.0083 ±0.0038	-10.0	UAO and OWA statistically equal

Table 7.16: Comparison of OWA and UAO for delay of best solutions of 30 runs for TFSA. % imp = percentage improvement achieved by UAO compared to OWA. Statistically significant results are in italics.

Case	OWA	UAO	% imp	Remarks
n15	0.0030 ±0.0002	0.0032 ±0.0003	*-5.24*	OWA statistically better
n25	0.0034 ±0.0002	0.0034 ±0.0004	0.36	UAO and OWA statistically equal
n33	0.0066 ±0.0059	0.0064 ±0.0063	4.19	UAO and OWA statistically equal
n40	0.0055 ±0.0006	0.0071 ±0.0024	*-29.85*	OWA statistically better
n50	0.0056 ±0.0023	0.0093 ±0.0134	*-65.37*	OWA statistically better

Table 7.17: Comparison of OWA and UAO for delay of best solutions of 30 runs for TEFSA. % imp = percentage improvement achieved by UAO compared to OWA. Statistically significant results are in italics.

Case	OWA	UAO	% imp	Remarks
n15	0.0030 ±0.0002	0.0030 ±0.0002	0.0	UAO and OWA statistically equal
n25	0.0034 ±0.0002	0.0039 ±0.0008	*-15.1*	OWA statistically better
n33	0.0073 ±0.0127	0.0058 ±0.0014	*20.6*	UAO statistically better
n40	0.0056 ±0.0005	0.0066 ±0.0016	*-17.4*	OWA statistically better
n50	0.0055 ±0.0032	0.0066 ±0.0039	*-19.7*	OWA statistically better

Table 7.18: Comparison of OWA and UAO for number of hops of best solutions of 30 runs for FSA. % imp = percentage improvement achieved by UAO compared to OWA. Statistically significant results are in italics.

Case	OWA	UAO	% imp	Remarks
n15	5.9 ±0.7	7.2 ±1.3	*-20.8*	OWA statistically better
n25	6.3 ±1.4	6.5 ±1.9	-4.3	UAO and OWA statistically equal
n33	11.2 ±2.9	9.7 ±3.2	*13.9*	UAO statistically better
n40	9.5 ±1.2	11.7 ±1.5	*-23.2*	OWA statistically better
n50	9.9 ±1.5	12.0 ±1.9	*-20.5*	OWA statistically better

Table 7.19: Comparison of OWA and UAO for number of hops of best solutions of 30 runs for TFSA. % imp = percentage improvement achieved by UAO compared to OWA. Statistically significant results are in italics.

Case	OWA	UAO	% imp	Remarks
n15	5.6 ±0.8	6.3 ±1.0	*-13.10*	OWA statistically better
n25	5.3 ±0.9	5.1 ±1.3	3.77	UAO and OWA statistically equal
n33	9.7 ±2.5	8.6 ±2.9	11.38	UAO and OWA statistically equal
n40	8.5 ±1.2	11.0 ±1.4	*-29.80*	OWA statistically better
n50	8.8 ±1.3	10.8 ±1.9	*-22.64*	OWA statistically better

Table 7.20: Comparison of OWA and UAO for number of hops of best solutions of 30 runs for TEFSA. % imp = percentage improvement achieved by UAO compared to OWA. Statistically significant results are in italics.

Case	OWA	UAO	% imp	Remarks
n15	5.2 ±0.7	5.5 ±0.6	-5.1	UAO and OWA statistically equal
n25	5.5 ±1.2	6.8 ±2.5	*-23.5*	OWA statistically better
n33	7.8 ±1.0	8.7 ±1.9	*-10.6*	OWA statistically better
n40	9.1 ±1.0	9.2 ±1.2	-0.7	UAO and OWA statistically equal
n50	7.9 ±1.1	8.8 ±1.2	*-11.8*	OWA statistically better

Table 7.21: Comparison of OWA and UAO for reliability of best solutions of 30 runs for FSA. % imp = percentage improvement achieved by UAO compared to OWA. Statistically significant results are in italics.

Case	OWA	UAO	% imp	Remarks
n15	0.273 ±0.000	0.273 ±0.000	0.0	UAO and OWA statistically equal
n25	0.091 ±0.019	0.089 ±0.018	2.9	UAO and OWA statistically equal
n33	0.064 ±0.022	0.059 ±0.011	8.5	UAO and OWA statistically equal
n40	0.036 ±0.005	0.037 ±0.000	-3.6	UAO and OWA statistically equal
n50	0.023 ±0.004	0.024 ±0.003	-4.4	UAO and OWA statistically equal

Table 7.22: Comparison of OWA and UAO for reliability of best solutions of 30 runs for TFSA. % imp = percentage improvement achieved by UAO compared to OWA. Statistically significant results are in italics.

Case	OWA	UAO	% imp	Remarks
n15	0.273 ±0.000	0.272 ±0.000	0.04	UAO and OWA statistically equal
n25	0.092 ±0.038	0.087 ±0.015	5.86	UAO and OWA statistically equal
n33	0.061 ±0.003	0.063 ±0.017	-3.32	UAO and OWA statistically equal
n40	0.037 ±0.000	0.037 ±0.000	0.02	UAO and OWA statistically equal
n50	0.025 ±0.000	0.025 ±0.000	0.14	UAO and OWA statistically equal

Table 7.23: Comparison of OWA and UAO for reliability of best solutions of 30 runs for TEFSA. % imp = percentage improvement achieved by UAO compared to OWA. Statistically significant results are in italics.

Case	OWA	UAO	% imp	Remarks
n15	0.273 ±0.001	0.272 ±0.000	0.1	UAO and OWA statistically equal
n25	0.095 ±0.031	0.088 ±0.014	*7.9*	UAO statistically better
n33	0.060 ±0.000	0.060 ±0.000	0.2	UAO and OWA statistically equal
n40	0.037 ±0.000	0.037 ±0.000	0.0	UAO and OWA statistically equal
n50	0.020 ±0.007	0.024 ±0.004	*-19.0*	OWA statistically better

For example, both UAO and OWA produced statistically equal results for FSA as observed in Table 7.21. The same situation was observed in Table 7.22 for TFSA where OWA and UAO showed statistically equal improvement for all test cases. Finally, for TEFSA, a trend similar to FSA and TFSA was observed (refer to Table 7.23), where UAO and OWA had equal improvement for four test cases. The only exception was *n50*, where OWA showed statistically better improvement than UAO.

7.5 Conclusion

This chapter presented a novel approach for topology design of DLANs based on a fuzzy simulated annealing (FSA) algorithm. Two variants of the FSA algorithm were proposed, namely tabu FSA (TFSA) and evolutionary tabu FSA (TEFSA). TFSA combines FSA with tabu search, while TEFSA integrated features of tabu search and simulated evolution with FSA. The performance of the three algorithms was evaluated using both the OWA and UAO operators. Results suggest that TEFSA generally performed better than the other two variants. Moreover, the performance of UAO and OWA were found to be comparable. Another issue discussed in the chapter was the proposal of dynamic assignment of the Markov chain, M. Although the proposed computation of dynamic M produced lower quality results than FSA, degradation in quality was not significant.

The next chapter proposes and discusses a swarm intelligence technique, namely ant colony optimization, in the context of its application to the multi-objective DLAN topology design problem.

Chapter 8

Fuzzy Ant Colony Optimization Algorithm for DLAN Topology Design

This chapter presents a fuzzy multi-objective ant colony optimization algorithm for the DLAN topology design problem. The chapter first discusses the main features of the algorithm, which include the discussion on the fuzzy heuristic value, and the fuzzy multi-objective function to assess the quality of solution generated by an ant. This is followed by empirical results to evaluate the performance of the fuzzy ACO algorithm. This evaluation is based on a number of ACO parameters, which include pheromone deposit and evaporation rate and the number of ants. The performance of the OWA and UAO operators is also evaluated through their usage with the ant colony algorithm.

8.1 Fuzzy Ant Colony Optimization Algorithm

As discussed earlier in Section 2.4.6, Ant Colony Optimization (ACO) is a relatively new optimization algorithm. The ACO maintains a population of ants, where each ant is responsible for building a feasible network topology. The ant starts with the root node and incrementally builds a topology. The guiding factors in the process of decision and selection of a particular path are the heuristic value, pheromone trail, and evaporation. A complete tour by an ant results in a complete feasible network topology. Below, each step of the proposed fuzzy ACO is presented.

8.1.1 Initialization (Generation of Ants)

Since ACO is a population-based algorithm, initialization consists of generating a set of candidate solutions. These initial solutions are generated randomly. That is, each ant produces a solution without any feedback or previous information about the paths. However, while an ant is generating a solution, the constraints are checked at each step to ensure feasible initial solutions. Algorithm parameters (such as pheromone deposit and evaporation rates, as well as values of α_{ant} and β_{ant}) are also initialized in this phase.

8.1.2 Ants Activity

Once the initial set of solutions is generated, the next step is to find the probability of selecting link l between nodes i and j for the k^{th} ant, and to apply pheromone deposit and pheromone evaporation rules. The approach adopted in this work is to update pheromone values based on the solution generated by the "elitist" ant

only (discussed in Section 2.4.6). More specifically, the iteration-based elitist ant approach [10, 117, 170] has been used in this thesis. In every iteration following initialization, each ant constructs a solution using the link selection probability function given in the following equation:

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha_{ant}}[\eta_{ij}]^{\beta_{ant}}}{\sum_{l \in N_i}[\tau_{ij}(t)]^{\alpha_{ant}}[\eta_{ij}]^{\beta_{ant}}} \tag{8.1}$$

where N_i is the set of neighbors of node i, $p_{ij}^{k}(t)$, is the probability of selecting a connection l between nodes i and j for the k^{th} ant, τ_{ij} is the pheromone on connection l, and η_{ij} is the heuristic value of connection l. α_{ant} and β_{ant} represent the influence of pheromone content and heuristic respectively.

Once each ant has finished its tour and found a valid solution, the ant which generates the highest value for Equation (3.11) (or Equation (4.1)) is taken as the elitist ant. This ant is allowed to update the pheromone value. For each link that occurs in the solution of the elitist ant, pheromone is updated by adding $\Delta\tau_{ij}$ to the current pheromone concentration of that link. Pheromone evaporation takes place on all links, irrespective of whether the links are part of the elitist ant or not. The updated values of pheromone are then passed to following ants, and the above cycle is repeated until a predefined stopping condition is met.

The prime advantage of the above elitist ant approach is that the approach forces the ants to search more rigorously near the best solution that has been found so far. However, the approach can have a disadvantage of exploiting too much, and exploring less.

8.1.3 Fuzzy Heuristic Value

Another novelty of the proposed fuzzy ACO is the introduction of a fuzzy heuristic value in Equation (8.1) above. The heuristic value η_{ij} basically provides an insight into the structure of the individual elements of the solution. In the context of DLAN topology design, each element is a link between any two nodes i and j. The heuristic value of a link is calculated exactly as given in Equation (6.1). Details of calculating the upper and lower bounds for each objective in Equation (6.1) have already been discussed in Chapter 6.

8.2 Results and Discussion

The fuzzy ACO was applied to the five test cases. To give equal importance to pheromone level on a link and the heuristic value of the link, both α_{ant} and β_{ant} were taken as 0.5. The main focus of the experiments was to study the effect of the colony size (i.e. number of ants), and the effect of pheromone deposit and evaporation, and also to evaluate the performance of the ACO using the OWA and UAO operators. Table 8.1 shows the combinations of pheromone deposit and evaporation rates used in the experiments. The colony was taken as 10, 15, 20, 25, and 30 ants. Thirty independent runs were made for each parameter setup, and the average of best solutions found in each run was reported, with the standard deviation. Each test instance was run for 100 iterations.

Table 8.1: Parameter settings for fuzzy ACO used in experiments. DEP = difference between pheromone deposit and evaporation rates.

Parameter set	Pheromone rates		DEP
	Deposit	Evaporation	
Set 1	0.2	0.0	0.2
Set 2	0.4	0.1	0.3
Set 3	0.6	0.2	0.4
Set 4	0.8	0.3	0.5

8.2.1 Effect of Pheromone Deposit and Evaporation

An objective of the work was to study the effect of pheromone deposit and evaporation. In the original ACO algorithm, the trail is updated proportional to the quality of solution as given in Equation (2.24). This does not provide enough insight into the effect of high and low pheromone deposit and evaporation rates. Therefore, pheromone update and evaporation were statically assigned to links. The parameters in Table 8.1 were used to study the effect of pheromone deposit and evaporation. These parameters depict different rates of pheromone deposit and evaporation, as given in columns 2 and 3 respectively. Column 4 of the table depicts the difference between the pheromone deposit and evaporation rate, denoted by DEP. Note that in Table 8.1, the gap between pheromone deposit and evaporation rates (i.e. DEP) increases with each set. For example, in Set 1, the difference between deposit and evaporation was 0.2. DEP gradually increased in each set and reached a maximum of 0.5 in Set 4. Experiments were done using 20 ants.

Table 8.2 depicts the best and worst overall goodness achieved using OWA. The last column depicts the percentage improvement obtained when the two situations were compared. As an example, consider the case of *n25*, where the overall goodness

of 0.421 was obtained with DEP = 0.4 (i.e. deposit = 0.6 and evaporation = 0.2), while for DEP = 0.2 (representing deposit = 0.2, and evaporation = 0.0), the overall goodness was 0.277. This resulted in an improvement of 34.17% which was achieved when the gap between deposit and evaporation was high. As is obvious from the table, moderate to high improvements were achieved for almost all cases. An exception was case *n15*, where the best results were obtained when the deposit rate was low and no evaporation was allowed (DEP = 0.2), giving an improvement of 17.01% over the lowest goodness levels obtained with DEP = 0.5. A general trend in this table suggests that the worst results were obtained when the difference between the deposit and evaporation was the lowest (i.e. deposit = 0.2, evaporation = 0.0). Nevertheless, all percentage improvements in Table 8.2 were statistically significant, as validated by the t-test.

Table 8.2: Results for best and worst average overall goodness and their respective pheromone deposit and evaporation rate setup using OWA. Time = Run time (in seconds), % imp = percentage improvement. Statistically significant improvement is in italics.

Case	DEP	Max. Goodness	Time	DEP	Min. Goodness	Time	% imp
n15	0.2	0.294 ±0.033	39.3	0.5	0.244 ±0.024	25.4	*17.01*
n25	0.4	0.421 ±0.022	222.9	0.2	0.277 ±0.021	230.7	*34.17*
n33	0.3	0.342 ±0.030	452.5	0.2	0.254 ±0.023	458.2	*25.79*
n40	0.5	0.331 ±0.021	1228.6	0.2	0.279 ±0.014	2044.7	*15.62*
n50	0.5	0.258 ±0.022	4050.3	0.2	0.217 ±0.014	12572.9	*15.86*

The results presented in Table 8.3 for UAO followed the same pattern as that of OWA. In this table, it is observed that most of these improvements were of moderate level, generally greater than 15%. Again, an exception was in the case of *n15*, where a slight improvement of 6.21% was observed. It is also obvious from the table that

Table 8.3: Results for best and worst average overall goodness and their respective pheromone deposit and evaporation rate setup using UAO. Time = Run time (in seconds), % imp = percentage improvement. Statistically significant improvement is in italics.

Case	DEP	Max. Goodness	Time	DEP	Min. Goodness	Time	% imp
n15	0.5	0.333 ±0.000	25.4	0.2	0.312 ±0.009	25.3	*6.21*
n25	0.3	0.360 ±0.009	246.0	0.2	0.289 ±0.004	245.4	*19.81*
n33	0.5	0.347 ±0.004	501.9	0.2	0.268 ±0.007	516.3	*22.75*
n40	0.4	0.346 ±0.006	1143.0	0.2	0.290 ±0.003	1741.2	*16.30*
n50	0.4	0.334 ±0.002	4239.4	0.2	0.264 ±0.006	12461.4	*20.93*

for all test cases, the worst results were obtained when DEP = 0.2. However, all percentage improvements were found to be statistically significant, as validated by the t-test.

The results and observations presented in Tables 8.2 and 8.3 can be attributed to the following explanation. When the pheromone deposit rate is low, and no evaporation takes place (a situation associated with DEP = 0.2), then the pheromone contents on links present in the elitist ant are not much different from the pheromone contents on links not present in the elitist ant. Consequently, there will be a little difference between the probabilities, $p_{ij}^k(t)$, which makes selection of links more random. In general, this situation would persist as time elapses. There might be very few links which would have accumulated a noticeable amount of pheromone, and therefore would be selected again and again. On the other hand, if the pheromone deposit rate is high, and pheromone evaporation also takes place on links (a situation associated with DEP = 0.3, 0.4, and 0.5), then the amount of pheromone deposited on links in the elitist ant will be substantially higher than the pheromone on links not present in the elitist ant. Thus, there will be a significant difference between

the probabilities, $p_{ij}^k(t)$. Therefore, links with high pheromone content would become strong candidates for selection, resulting in the search to be more directional. Therefore, for the same amount of time, FACO will converge faster towards a sub-optimal solution for situations where DEP is high as compared to instances where DEP is low. The above claim is supported by the plots in Figure 8.1. This figure shows a typical behavior of FACO with DEP = 0.2 and DEP = 0.5 for test case *n50* using the UAO operator. It is observed in this figure that with DEP = 0.5, the improvement in the overall goodness is much higher than that obtained with DEP = 0.2 for the same number of iterations.

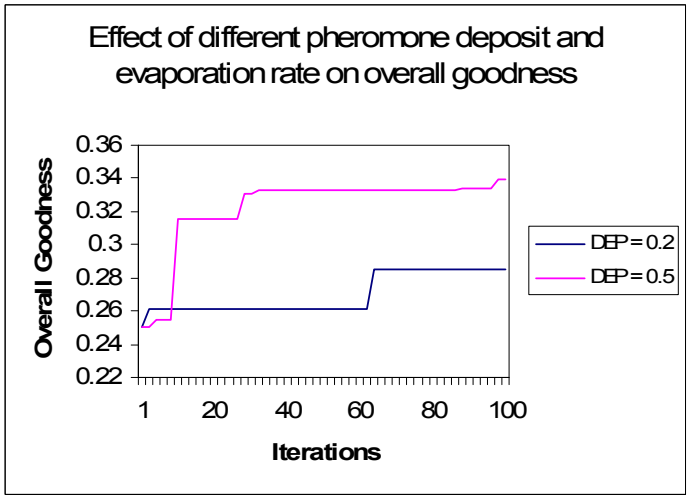


Figure 8.1: Plot for overall goodness for test case *n50* using FACO with DEP = 0.2 and DEP = 0.5

8.2.2 Effect of Number of Ants

Tables 8.4 to 8.8 show the effect of the number of ants on the overall goodness of the solution using OWA. The tables consist of nine columns. The first column shows the number of ants, ranging from 10 to 30. Columns 2 to 9 display the results

and run time obtained using the four parameter setups according to Table 8.1. It is observed in these tables that, collectively, an increasing number of ants had a positive impact on the quality of the final solution. The trend was observed for all four parameter sets. For example, in column 8 of Table 8.4, it is observed for case *n50* with DEP = 0.5 that the overall goodness of solution was 0.239 for 10 ants, which increased with each step as the number of ants increased. An overall goodness value of 0.270 was achieved with 30 ants at the end. This improvement is given as a percentage in Table 8.14 with a value of 11.21% when OWA was used. In a similar manner, percentage improvements for other cases are also given the table. A validation through t-test showed that, in general, all improvements were statistically significant. The percentage improvements are also illustrated in Figure 8.2(a). Another interesting observation in Tables 8.4 to 8.8 is that the best results (marked in boldface) were obtained with the number of ants being 30 in most cases. An exception is case *n40*, where the best overall goodness was achieved with 25 ants.

Tables 8.9 to 8.13 depict the behavior of the ACO algorithm with UAO when the number of ants was increased from 10 to 30. For UAO, the pattern was similar to that of OWA, where it is observed that, when the number of ants were increased from 10 to 30 in progression, the overall goodness of solution also increased proportionally. The percentage improvements are given in Table 8.14 and depicted in Figure 8.2(b). The results in Table 8.14 show that the improvements were statistically significant, as validated by the t-test. As with the case of OWA, the best results were obtained when 30 ants were used, with the exception of *n33* where the best solution was obtained with the number of ants equal to 25.

The above trends appear to be logical. Note that each ant is independent in

generating its solution, although the elitist ant influences the pheromone concentrations on the links of the solution represented by the ant. Increasing the number of ants initially enhances diversity in solutions, thus increasing the probability of finding a new elite ant of better quality. This in turn allows the elitist ant to pass information of links having higher quality to ants in the next iteration.

Another interesting observation in Tables 8.4 to 8.8 is about the quality of solutions obtained with different pheromone deposit and evaporation rates. A careful analysis of these tables shows that the level of overall goodness achieved with DEP values of 0.3, 0.4, and 0.5 was almost in the same range, while with DEP = 0.2 the quality of overall goodness was considerably less. As an example, consider Table 8.5, where the overall goodness for DEP = 0.3 varied between 0.310 and 0.333, for DEP = 0.4 between 0.312 and 0.331, and for DEP = 0.5 between 0.314 and 0.331. These ranges are very close to each other. However, when DEP = 0.2 was considered, the goodness ranged from 0.268 to 0.283. This range is relatively much less than the others mentioned above. The trend was more or less true for all test cases, with the exception of *n15*.

The same trend was also very visible for UAO for all cases. One possible reason for this behavior is that for DEP = 0.2 (with no evaporation), there is no feedback from 'bad' links. In other words, although the good links are rewarded by pheromone deposit, bad links are not penalized since no evaporation takes place on these links. Therefore, their quality does not deteriorate, in contrast to other cases with DEP from 0.3 to 0.5, where bad links are penalized. This penalty has a positive effect on the faster convergence of the algorithm, since good quality links will be passed to later generations again and again, while bad quality links will be annihilated from

Table 8.4: Results for n50 with OWA for different population size, pheromone deposit rate, and evaporation rate. OG = average overall goodness with standard deviation.

Number of Ants	DEP = 0.2		DEP = 0.3		DEP = 0.4		DEP = 0.5	
	OG	Time	OG	Time	OG	Time	OG	Time
10	0.208 ±0.015	6399.1	0.244 ±0.028	2666.7	0.243 ±0.022	2166.7	0.239 ±0.028	1995.4
15	0.217 ±0.015	8246.5	0.247 ±0.028	3925.3	0.257 ±0.041	3248.2	0.238 ±0.022	3011.6
20	0.217 ±0.014	12572.9	0.251 ±0.028	5299.8	0.250 ±0.017	4250.3	0.258 ±0.022	4050.3
25	0.224 ±0.014	15465.0	0.252 ±0.019	6660.7	0.254 ±0.021	5498.5	0.267 ±0.030	4970.7
30	0.219 ±0.015	18884.8	0.262 ±0.021	7838.6	0.253 ±0.022	6632.7	**0.270** ±0.032	5923.3

Table 8.5: Results for n40 with OWA for different population size, pheromone deposit rate, and evaporation rate. Goodness = average overall goodness with standard deviation.

Number of Ants	DEP = 0.2		DEP = 0.3		DEP = 0.4		DEP = 0.5	
	OG	Time	OG	Time	OG	Time	OG	Time
10	0.268 ±0.012	1213.1	0.310 ±0.019	849.1	0.312 ±0.020	817.8	0.326 ±0.025	811.0
15	0.271 ±0.012	1681.9	0.324 ±0.026	1066.0	0.323 ±0.022	1034.1	0.314 ±0.019	1019.2
20	0.279 ±0.014	2044.7	0.322 ±0.020	1286.6	0.323 ±0.020	1246.1	0.331 ±0.021	1228.6
25	0.283 ±0.021	2450.1	**0.333** ±0.028	1521.2	0.331 ±0.021	1464.9	0.330 ±0.019	1437.2
30	0.283 ±0.016	2906.9	0.323 ±0.022	1129.3	0.315 ±0.020	1068.7	0.324 ±0.020	1042.6

Table 8.6: Results for n33 with OWA for different population size, pheromone deposit rate, and evaporation rate. Goodness = average overall goodness with standard deviation.

Number of Ants	DEP = 0.2		DEP = 0.3		DEP = 0.4		DEP = 0.5	
	OG	Time	OG	Time	OG	Time	OG	Time
10	0.249 ±0.024	430.4	0.333 ±0.027	426.7	0.328 ±0.028	426.0	0.317 ±0.023	426.0
15	0.236 ±0.013	445.7	0.331 ±0.020	439.3	0.340 ±0.028	438.7	0.335 ±0.023	438.3
20	0.254 ±0.023	458.2	0.342 ±0.030	452.5	0.333 ±0.020	451.0	0.339 ±0.022	451.2
25	0.256 ±0.021	474.8	0.343 ±0.024	466.3	0.341 ±0.026	464.3	0.341 ±0.020	464.9
30	0.267 ±0.022	489.5	0.348 ±0.024	478.5	0.343 ±0.022	478.7	**0.362** ±0.022	553.3

Table 8.7: Results for n25 with OWA for different population size, pheromone deposit rate, and evaporation rate. Goodness = average overall goodness with standard deviation.

Number of Ants	DEP = 0.2		DEP = 0.3		DEP = 0.4		DEP = 0.5	
	OG	Time	OG	Time	OG	Time	OG	Time
10	0.272 ±0.018	214.9	0.389 ±0.015	215.0	0.403 ±0.027	213.3	0.404 ±0.024	211.6
15	0.281 ±0.018	221.8	0.410 ±0.028	221.0	0.417 ±0.030	217.1	0.421 ±0.022	217.1
20	0.277 ±0.021	230.7	0.414 ±0.022	225.5	0.421 ±0.022	222.9	0.415 ±0.020	222.9
25	0.293 ±0.021	237.0	0.419 ±0.024	229.2	0.423 ±0.030	228.3	0.425 ±0.030	228.4
30	0.294 ±0.016	238.8	0.426 ±0.023	233.8	**0.430** ±0.023	234.0	0.423 ±0.023	234.1

Table 8.8: Results for n15 with OWA for different population size, pheromone deposit rate, and evaporation rate. Goodness = average overall goodness with standard deviation.

Number of Ants	DEP = 0.2		DEP = 0.3		DEP = 0.4		DEP = 0.5	
	OG	Time	OG	Time	OG	Time	OG	Time
10	0.276 ±0.029	12.9	0.251 ±0.026	13.0	0.244 ±0.044	13.0	0.237 ±0.029	13.2
15	0.290 ±0.018	19.0	0.257 ±0.032	19.0	0.250 ±0.033	19.1	0.254 ±0.041	19.2
20	0.294 ±0.033	39.3	0.249 ±0.023	40.3	0.253 ±0.039	27.5	0.244 ±0.024	25.4
25	0.302 ±0.027	47.9	0.279 ±0.034	47.5	0.256 ±0.027	31.7	0.248 ±0.024	44.6
30	**0.313** ±0.032	50.3	0.277 ±0.044	31.0	0.259 ±0.027	31.2	0.256 ±0.033	31.2

Table 8.9: Results for n50 with UAO for different population size, pheromone deposit rate, and evaporation rate. OG = average overall goodness with standard deviation.

Number of Ants	DEP = 0.2		DEP = 0.3		DEP = 0.4		DEP = 0.5	
	OG	Time	OG	Time	OG	Time	OG	Time
10	0.258 ±0.007	6073.4	0.331 ±0.004	2729.9	0.333 ±0.004	2237.0	0.333 ±0.004	2008.9
15	0.262 ±0.008	9452.9	0.333 ±0.002	4029.3	0.333 ±0.002	3418.6	0.334 ±0.004	2980.7
20	0.264 ±0.006	12461.4	0.334 ±0.002	5498.3	0.334 ±0.002	4239.4	0.334 ±0.004	3980.6
25	0.266 ±0.005	15940.8	0.334 ±0.003	6681.8	0.335 ±0.003	5478.4	0.336 ±0.004	5110.3
30	0.265 ±0.009	16546.5	0.335 ±0.003	8132.1	**0.336** ±0.004	6478.8	0.334 ±0.002	5891.5

Table 8.10: Results for n40 with UAO for different population size, pheromone deposit rate, and evaporation rate. OG = average overall goodness with standard deviation.

Number of Ants	DEP = 0.2		DEP = 0.3		DEP = 0.4		DEP = 0.5	
	OG	Time	OG	Time	OG	Time	OG	Time
10	0.288 ±0.003	1032.3	0.344 ±0.008	741.9	0.344 ±0.007	727.8	0.345 ±0.008	716.0
15	0.289 ±0.003	1416.5	0.346 ±0.006	947.6	0.345 ±0.005	931.3	0.343 ±0.005	919.6
20	0.290 ±0.003	1741.2	0.346 ±0.006	1178.5	0.346 ±0.006	1143.0	0.345 ±0.005	1142.5
25	0.290 ±0.003	2178.4	0.348 ±0.008	1403.9	0.349 ±0.005	1358.5	0.350 ±0.007	1341.3
30	0.290 ±0.002	2602.5	0.349 ±0.006	1614.2	**0.352** ±0.006	1561.9	0.348 ±0.007	1534.3

Table 8.11: Results for n33 with UAO for different population size, pheromone deposit rate, and evaporation rate. Goodness = average overall goodness with standard deviation.

Number of Ants	DEP = 0.2		DEP = 0.3		DEP = 0.4		DEP = 0.5	
	OG	Time	OG	Time	OG	Time	OG	Time
10	0.265 ±0.007	459.8	0.343 ±0.005	451.8	0.345 ±0.007	451.4	0.345 ±0.005	451.2
15	0.264 ±0.005	487.5	0.345 ±0.004	477.8	0.346 ±0.006	477.1	0.345 ±0.005	476.6
20	0.268 ±0.007	516.3	0.347 ±0.005	502.9	0.347 ±0.005	502.5	0.347 ±0.004	501.9
25	0.271 ±0.006	551.5	0.348 ±0.005	528.4	0.348 ±0.006	528.1	**0.349** ±0.006	528.1
30	0.270 ±0.006	177.5	0.348 ±0.005	153.9	0.348 ±0.004	152.7	0.348 ±0.004	152.7

Table 8.12: Results for n25 with UAO for different population size, pheromone deposit rate, and evaporation rate. Goodness = average overall goodness with standard deviation.

Number of Ants	DEP = 0.2		DEP = 0.3		DEP = 0.4		DEP = 0.5	
	OG	Time	OG	Time	OG	Time	OG	Time
10	0.284 ±0.004	227.4	0.355 ±0.007	227.0	0.357 ±0.010	224.4	0.358 ±0.008	223.4
15	0.288 ±0.007	233.9	0.355 ±0.009	238.7	0.358 ±0.013	237.8	0.357 ±0.007	237.9
20	0.289 ±0.004	245.4	0.360 ±0.009	246.0	0.358 ±0.008	246.1	0.359 ±0.008	246.1
25	0.288 ±0.006	266.8	0.359 ±0.009	257.1	0.361 ±0.008	257.2	0.361 ±0.006	257.2
30	0.289 ±0.004	278.7	**0.363** ±0.008	268.5	0.362 ±0.008	268.7	0.359 ±0.006	268.6

Table 8.13: Results for n15 with UAO for different population size, pheromone deposit rate, and evaporation rate. Goodness = average overall goodness with standard deviation.

Number of Ants	DEP = 0.2		DEP = 0.3		DEP = 0.4		DEP = 0.5	
	OG	Time	OG	Time	OG	Time	OG	Time
10	0.310 ±0.011	13.0	0.333 ±0.000	13.2	0.333 ±0.000	13.2	0.333 ±0.000	13.2
15	0.312 ±0.011	19.2	0.333 ±0.000	19.2	0.333 ±0.000	19.3	0.334 ±0.002	19.3
20	0.312 ±0.009	25.3	0.333 ±0.001	25.5	0.333 ±0.000	25.5	0.333 ±0.000	25.4
25	0.319 ±0.008	48.1	0.334 ±0.002	47.8	0.334 ±0.002	40.6	0.334 ±0.002	34.8
30	0.315 ±0.007	53.1	0.333 ±0.001	31.5	0.333 ±0.000	31.4	**0.334** ±0.002	31.3

Table 8.14: Improvement with respect to increase in number of ants for different DEP rates using OWA. Statistically significant improvements are in italics.

Case	% Improvement			
	DEP = 0.2	DEP = 0.3	DEP = 0.4	DEP = 0.5
n15	*12.03*	*9.46*	5.66	*7.66*
n25	*7.35*	*8.77*	*6.21*	*4.50*
n33	*6.91*	*4.34*	*4.54*	*12.43*
n40	*5.43*	*6.93*	*5.65*	1.30
n50	*4.94*	*7.12*	3.91	*11.21*

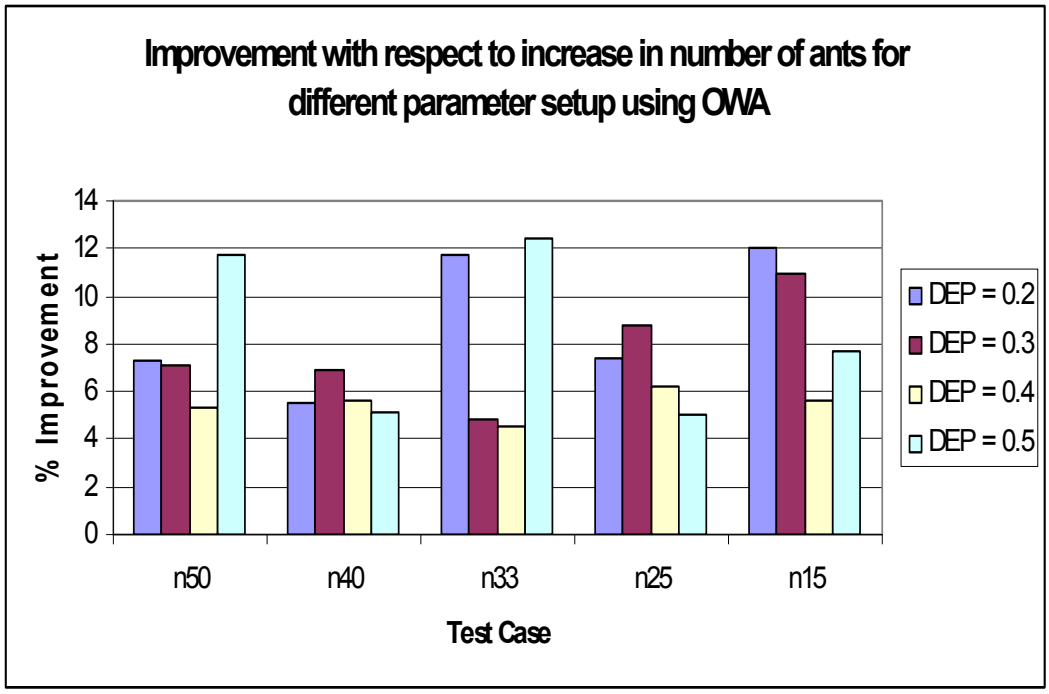
Table 8.15: Improvement with respect to increase in number of ants for different DEP rates using UAO. Statistically significant improvements are in italics.

Case	% Improvement			
	DEP = 0.2	DEP = 0.3	DEP = 0.4	DEP = 0.5
n15	*1.73*	0.07	0.07	0.20
n25	*1.82*	*2.22*	*1.58*	0.26
n33	*2.09*	*1.39*	0.83	*1.09*
n40	*0.50*	*1.53*	*2.29*	0.91
n50	*2.62*	*1.26*	*0.69*	0.31

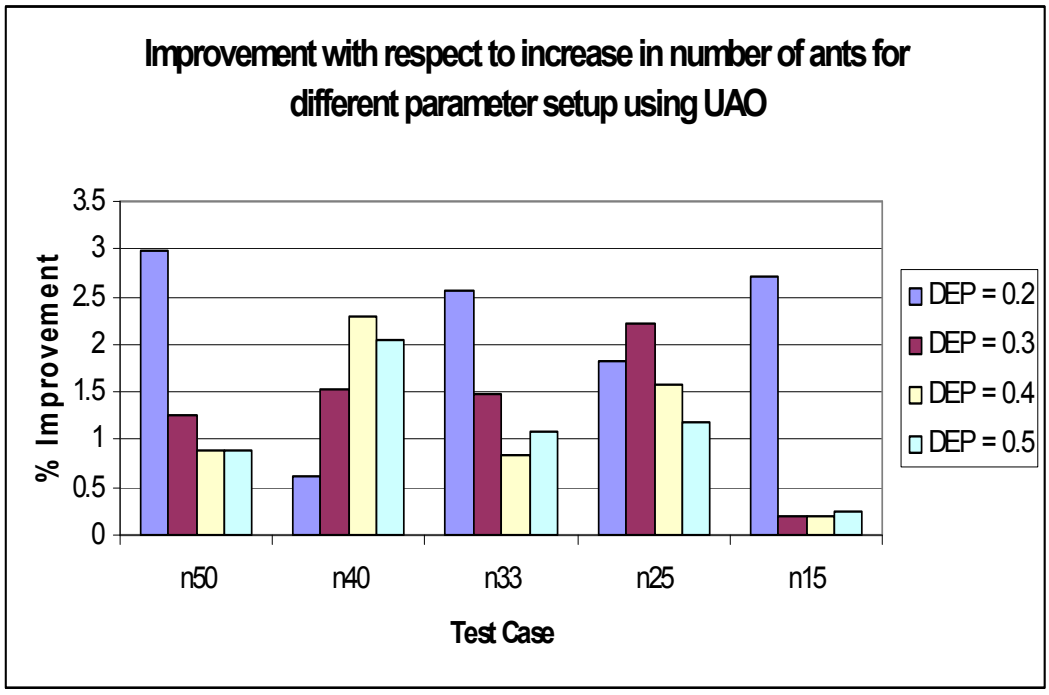
the solutions. For the case with no evaporation, the bad links continue generation by generation, thus preventing the algorithm from converging to a good quality solution for the same amount of time as that of the other deposit and evaporation setups.

8.2.3 Comparison of OWA and UAO

Table 8.16 compares the OWA and UAO operators. This comparison is done using linear regression analysis. The data for the analysis consisted of the best results with respect to different numbers of ants. That is, the data consisted of columns where the best results appear (in boldface) in tables 8.4 to 8.13. Each of these columns



(a)



(b)

Figure 8.2: Percentage improvement with increase in number of ants for different parameter setup using (a) OWA (b) UAO

contains five values (one for each different number of ants), where each value is an average of 30 runs. Thus, each regression coefficient in Table 8.16 was obtained using 150 values (5 values $\times$ 30 runs). The regression coefficients of OWA and UAO are provided in columns 2 and 3 of Table 8.16 for each test case. Column 4 enlists the ratio of UAO and OWA. Based on this ratio, the sensitivity of OWA and UAO with respect to an increasing number of ants was assessed. A comment related to this is provided in column 5.

It is observed in Table 8.16 that UAO was more sensitive with respect to an increasing number of ants. For cases *n50*, *n25*, and *n15*, both OWA and UAO increased the overall goodness almost at the same rate when the number of ants increased. However, UAO performed better than OWA for *n33* with a rate of 1.7. In other words, if the number of ants is increased, then UAO will improve the goodness 1.7 faster than OWA. For the case *n40*, UAO showed a rate of more than 6, which is significantly higher than the rate of OWA.

8.3 Conclusions

This chapter proposed and investigated a fuzzy multi-objective ant colony optimization algorithm. A fuzzy heuristic value was proposed. Furthermore, since pheromone deposit and evaporation, and the population of ants, are important parameters in ACO, empirical analysis was done to study the effect of these parameters. The analysis revealed that better results were obtained when the difference in pheromone deposit and evaporation rates was high. This was a general trend for both OWA and UAO. As for the number of ants, better results were obtained when the number of

Table 8.16: Comparison of OWA and UAO for ACO.

Case	Regression coefficients		Ratio = $\frac{UAO}{OWA}$	Comment
	OWA	UAO		
n50	0.343	0.314	0.92	UAO increases goodness at almost the rate as that of OWA with increase in number of ants
n40	0.072	0.456	6.33	UAO increases goodness 6.3 times faster than OWA with increase in number of ants
n33	0.242	0.418	1.73	UAO increases goodness 1.7 times faster than OWA with increase in number of ants
n25	0.305	0.327	1.07	UAO increases goodness at almost the rate as that of OWA with increase in number of ants
n15	0.158	0.163	1.03	UAO increases goodness at almost the rate as that of OWA with increase in number of ants

ants was high. Moreover, mutual comparison of OWA and UAO suggested that UAO produced better results in the context of its usage in the ant colony optimization algorithm.

The next chapter investigates another swarm intelligence technique, the particle swarm optimization algorithm, with its application to the DLAN topology design problem.

Chapter 9

Fuzzy Particle Swarm Optimization for DLAN Topology Design

This chapter presents another swarm intelligence algorithm, namely, the fuzzy particle swarm optimization algorithm. The original particle swarm algorithm was adapted to address the multi-objective aspects of the DLAN topology design using fuzzy logic. The chapter first discusses the main features of the fuzzy PSO algorithm. This is followed by empirical results to evaluate the performance of the fuzzy PSO algorithm with respect to different parameters of the algorithm.

9.1 Fuzzy Particle Swarm Optimization Algorithm

Particle swarm optimization (PSO) was discussed in Chapter 2. Although PSO has been applied to a number of problems, its effectiveness in the multi-objective

domain has to be explored further. The development of a multi-objective PSO for the DLAN topology design problem is one step towards the assessment of the performance of PSO in multi-objective optimization with application to a real-world design problem. Therefore, the focus of this chapter is not to compare the variants and alterations proposed for PSO by many researchers, but rather the development of a fuzzy logic based multi-objective PSO, and a preliminary analysis of the fuzzy PSO with respect to the OWA and UAO operators.

The fuzzy PSO (FPSO) maintains a population of particles. Each particle is responsible for generating a feasible network topology. In contrast to ACO, where each ant (except the elitist ant) dies after generating a solution, a particle in PSO progresses iteration by iteration, learning from its own history, and it also inherits characteristics from other particles generating high-quality solutions. This is done while simultaneously considering the design objectives and constraints. In FPSO, a particle incrementally improves an already existing solution. This improvement is done by replacing low-quality links with high-quality ones. The guidance in selection of links is provided by three parameters: the particle's current position, its own best position so far, and the best position in relation to the particle's neighborhood. Each step of the proposed FPSO is discussed next.

9.1.1 Particle Position and Velocity Representation

For the original PSO, particle position as well as velocity representation were in the real number domain, that is, all $x_{ij} \in \Re$, and all $v_{ij} \in \Re$. However the DLAN topology design problem has discrete-valued variables. Therefore, the representation of particle positions and velocities need to change, and a set representation need to

be used. This representation scheme is described below.

A position will be the set

$$\mathbf{X}_i(t) = \{l_1, l_2, ..., l_q, ..., l_L\}$$

where l_q is a link between any two nodes a and b in the network, and l_L is the number of links, i.e. $|\mathbf{X}_i(t)| = L$. The velocity of the particle i is represented as

$$\mathbf{V}_i(t) = \{l_q \Leftrightarrow l_q{}'\}$$

where link l_q is removed and replaced with link $l_q{}'$, and $|\mathbf{V}_i(t)|$ gives the total number of changes to particle i.

Example 1: Consider a simple network of 6 nodes as given in Figure 9.1. The topology in this figure represents a possible configuration at time t, and thus represents a solution (i.e. particle). According to the network configuration in Figure 9.1, the current solution is given as

$$\mathbf{X}_i(t)=\{(1,2),\ (1,3),\ (3,5),\ (4,5),\ (4,6)\}$$

That is, there are links between nodes (1,2), (1,3), (3,5), (4,5) and (4,6). This $\mathbf{X}_i(t)$ is also used in Examples 2 and 3 below.

Also assume that at time t, $\mathbf{V}_i(t) = \{(2,4) \Leftrightarrow (1,2), (3,4) \Leftrightarrow (3,5), (5,6) \Leftrightarrow (4,6)\}$ where the symbol "$\Leftrightarrow$" represents exchange of links. That is, the current solution $\mathbf{X}_i(t)$ was obtained when link (2,4) was removed and replaced with (1,2), then (3,4) was removed and replaced with (3,5), and then (5,6) was removed and replaced with (4,6). This $\mathbf{V}_i(t)$ is also used in Examples 2 and 3 below.

9.1.2 Velocity Update

The velocity of particle i is updated using

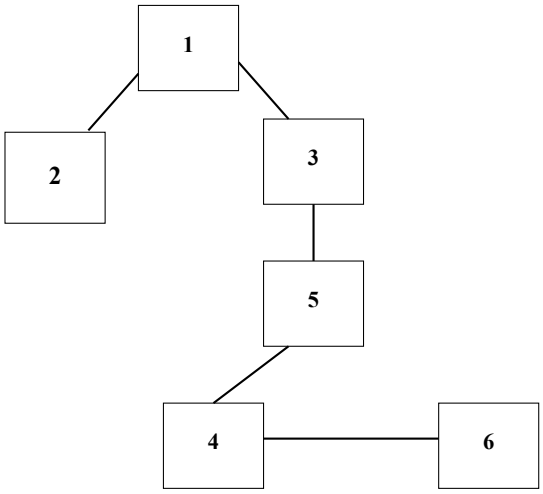


Figure 9.1: Network topology for PSO example

$$\mathbf{V}_i(t+1) = w \otimes \mathbf{V}_i(t) \oplus c_1 r_1(t) \otimes [\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)] \oplus c_2 r_2(t) \otimes [\mathbf{P}_g(t) \oslash \mathbf{X}_i(t)] \quad (9.1)$$

where $\mathbf{P}_i(t)$ represents the particle's own best position, and $\mathbf{P}_g(t)$ represents the global best position.

In Equation (9.1), the operator $\otimes$ is implemented as follows: the number of elements to be selected are determined as $\lfloor \mathrm{w} \times |\mathbf{V}_i(t)| \rfloor$. Then, the result will be the above number of elements randomly selected from $\mathbf{V}_i(t)$. The same is approach is applicable to other factors where the operator $\otimes$ is used.

The operator $\oslash$ is implemented as the 'exchange' operator. That is, the links in $\mathbf{X}_i(t)$ are replaced with the links in $\mathbf{P}_i(t)$.

The term $c_1 r_1(t) \otimes [\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)]$ is implemented by multiplying c_1 and $r_1(t)$ with the size of the set $\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)$ and taking the floor, i.e.

$$c_1 r_1(t) \otimes [\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)] = \lfloor c_1 r_1 \times |\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)| \rfloor \tag{9.2}$$

where $|\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)|$ represents the cardinality of the set. The result of Equation (9.2) indicates the number of elements that are randomly selected from the set $\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)$; $c_2 r_2(t) \otimes [\mathbf{P}_g(t) \oslash \mathbf{X}_i(t)]$ has the same meaning.

The operator $\oplus$ implements the set addition (union) operator, i.e. the elements in any two sets are combined in a new set using the set addition operator. Furthermore, V_{max} is used to limit the number of elements selected from a set.

Example 2: Continuing with Example 1, assume the following parameter values:

w $= 0.5$

$V_{max} = 2$

$c_1 = c_2 = 0.5$

$r_1 = 0.52$ (randomly generated)

$r_2 = 0.75$ (randomly generated)

Assume that the best goodness so far for particle i was generated by position,

$$\mathbf{P}_i(t) = \{(1,2),(1,4),(2,3),(2,5),(2,6)\}$$

Also assume that the best solution so far generated by the entire swarm was achieved by the following global best solution:

$$\mathbf{P}_g(t) = \{(1,2),(1,3),(1,4),(1,5),(1,6)\}$$

The inertia weight, w, determines the number of moves that will be randomly selected from $\mathbf{V}_i(t)$ (mentioned in Example 1 above). Since $w = 0.5$, and $|\mathbf{V}_i(t)| =$

3, the number of moves selected is $0.5 \times |\mathbf{V}_i(t)| = 1.5$. Since fractional moves are not possible, the value is truncated to 1.

Now, $0.5 \times \mathbf{V}_i(t) = \{(2,4) \Leftrightarrow (1,2)\}$. Note that $(3,4) \Leftrightarrow (3,5)$ OR $(5,6) \Leftrightarrow (4,6)$ is also possible; any one move of these three moves can be randomly chosen.

The difference between the particle's current position and its own best position, $\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)$ is calculated by replacing each link in $\mathbf{X}_i(t)$ with the link in the corresponding position in $\mathbf{P}_i(t)$ as

$\mathbf{P}_i(t) \oslash \mathbf{X}_i(t) = \{(1,2) \Leftrightarrow (1,2), (1,3) \Leftrightarrow (1,4), (3,5) \Leftrightarrow (2,3), (4,5) \Leftrightarrow (2,5),$
$(4,6) \Leftrightarrow (2,6)\}$

Therefore, $c_1 \times r_1 \otimes (\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)) = 0.5 \times 0.52 \times |\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)|$. Since cardinality of $\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)$ is 4 (i.e. there are four exchanges in the set, as $(1,2) \Leftrightarrow (1,2)$ is not considered an exchange), this implies that $0.5 \times 0.52 \otimes |\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)| = 1.04 = 1$. This means that any one of the four elements in $\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)$ can be randomly chosen. So, assume that $c_1 \times r_1 \otimes (\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)) = \{(4,6) \Leftrightarrow (2,6)\}$.

Similarly:

$\mathbf{P}_g(t) \oslash \mathbf{X}_i(t) = \{(1,2) \Leftrightarrow (1,2), (1,3) \Leftrightarrow (1,3), (3,5) \Leftrightarrow (1,4), (4,5) \Leftrightarrow (1,5),$
$(4,6) \Leftrightarrow (1,6)\}$

The cardinality of the above set is 3, since $(1,2) \Leftrightarrow (1,2)$ and $(1,3) \Leftrightarrow (1,3)$ are not considered exchanges. So, $0.5 \times 0.75 \otimes (\mathbf{P}_g(t) \oslash \mathbf{X}_i(t))$ =0.5 $\times$0.75 $\times$ 3 = 1.12 = 1 move. Assume $\{(4,5) \Leftrightarrow (1,5)\}$ is randomly chosen, although any combination consisting of a single move from $\mathbf{P}_g(t) \oslash \mathbf{X}_i(t)$ can be randomly chosen.

Putting the above calculations in Equation (9.1) gives $\mathbf{V}_i(t+1)$ containing three elements as

$$\mathbf{V}_i(t+1) = \{(2,4) \Leftrightarrow (1,2), (4,6) \Leftrightarrow (2,6), (4,5) \Leftrightarrow (1,5)\}$$

Since velocity clamping $V_{max} = 2$, only two moves (i.e. exchanges) from $\mathbf{V}_i(t+1)$ can be randomly chosen. Assume that $(2,4) \Leftrightarrow (1,2)$ and $(4,6) \Leftrightarrow (2,6)$ are chosen.

Hence,

$$\mathbf{V}_i(t+1) = \{(2,4) \Leftrightarrow (1,2), (4,6) \Leftrightarrow (2,6)\}$$

9.1.3 Particle Position Update

The position $\mathbf{X}_i(t)$ of a particle i is updated using

$$\mathbf{X}_i(t+1) = \mathbf{X}_i(t) \boxplus \mathbf{V}_i(t+1) \tag{9.3}$$

where $\boxplus$ is a special operator that updates the links in $\mathbf{X}_i(t)$ on the basis of link exchanges in $\mathbf{V}_i(t+1)$, to get the new position $\mathbf{X}_i(t+1)$.

Example 3: Continuing with Example 2,

$\mathbf{X}_i(t+1) = \mathbf{X}_i(t) \boxplus \mathbf{V}_i(t+1) = \{(1,2),(1,3),(3,5),(4,5),(4,6)\} \boxplus \{(2,4) \Leftrightarrow (1,2), (4,6) \Leftrightarrow (2,6)\} = \{(1,2), (1,3), (3,5), (4,5), (2,6)\}$

Notice that since the link (2,4) was not present in $\mathbf{X}_i(t)$, the exchange $(2,4) \Leftrightarrow (1,2)$ could not be performed. Therefore, in the new solution, the links (1,2), (1,3), (3,5), and (4,5) have been brought from the solution $\mathbf{X}_i(t)$, while the new link, i.e. (2,6), was introduced, replacing the link (4,6), as specified by the replacement in $\mathbf{V}_i(t+1)$.

9.1.4 Fitness Evaluation

The fitness (goodness) of a solution is evaluated using either Equation (3.11) or Equation (4.1), as discussed in Chapters 3 and 4.

9.1.5 Initialization

Since PSO is a population-based algorithm, the initialization process consists of generating a set of solutions. This process is exactly the same as discussed in Section 8.1.1 for fuzzy ACO. Algorithm parameters such as inertia weight, velocity clamping, and acceleration constants are also initialized. The goodness of each particle is then evaluated with respect to the reference solution (a predefined initial solution used in the earlier chapters of this thesis).

9.1.6 Particle Activity

It was mentioned in Chapter 2 that PSO has two basic models known as *lbest* PSO and *gbest* PSO. FPSO is based on the *gbest* model (although the *lbest* model could be used as well) to allow comparison with the fuzzy ACO. Recall that, in fuzzy ACO, the elitist ant has a significant effect in guiding the search process towards a certain direction. In other words, the fuzzy ACO is considering a global approach in guidance. Therefore, to match this global approach to the best possible extent, the *gbest* model has been adopted, where the global component is provided by the overall best particle.

Once the initial set of solutions is generated, the global best particle is chosen on the basis of the goodness value calculated in the initialization phase. Also, at this stage, each particle's current position is its best position. In the following iterations, each particle updates its position based on information provided by the particle's immediate previous position and by the alterations (moves) performed on the particle through the velocity update vector, as explained in Section 9.1.3. The velocity of a particle is updated on the basis of moves performed on the particle in its

immediate previous position, the particle's own best position so far, and the overall best position achieved by any particle in the swarm in any iteration, as described in Section 9.1.2. Moreover, to avoid premature convergence, the global best particle is updated regularly, i.e. as soon as a particle's overall goodness becomes higher than the overall goodness of the global best particle, that new particle is selected as the global best particle, and the search process continues. If no updating is done, then the algorithm will very quickly converge on a solution that might not even be a local minimum.

A 'move' in FPSO is very similar to what has been described for other algorithms in earlier chapters: removing a link and introducing a new one such that the tree is maintained (refer to Example 1 in Section 9.1.1). Then, the constraints are checked to evaluate the feasibility of the performed move. However, the notion of moves in FPSO depends on three factors, namely: moves performed in the immediate previous position of the particle, the structure of the particle's own best position, and the structure of the global best particle. For all these factors, the number of moves performed to get the new position of the particle is governed by parameters such as acceleration coefficients, inertia weight, and velocity clamping. Values of these parameters decide how many moves are required to get the new position of a particle.

9.2 Results and Discussion

The fuzzy PSO was applied to the five test cases. The performance of the algorithm was evaluated with respect to a number of parameters. These parameters are the

inertia weight w, velocity clamping V_{max}, swarm size, and acceleration constants. The parameter values used in the experiments are given in Table 9.1. In these experiments, each instance of the algorithm was run for 100 iterations. Thirty independent runs were executed for each parameter setup, and the average of best solutions found in each run was reported, with the standard deviation. Following default values were used for experiments, unless otherwise specified: number of particles = 20, $V_{max} = 5$, $w = 0.72$, and $c_1 = c_2 = 0.5$.

Table 9.1: Parameter settings for fuzzy PSO used in experiments.

Parameter	Values
Number of particles	5, 10, 15, 20, 25, 30
V_{max}	5 10% size of test case 20% size of test case
w	0.72 0.95 0.4
c_1, c_2	0.5 and 0.5 1.49 and 1.49 2.0 and 2.0

9.2.1 Effect of Swarm size

The effect of swarm size was investigated with different number of particles as given in Table 9.1. Other parameters were kept as follows: $V_{max} = 5$, $c_1 = c_2 = 0.5$, and inertia weight $w = 0.72$. Tables 9.2 to 9.6 reflect the effect of number of particles on the quality of solution. Column 1 lists the test case, column 2 gives the overall goodness obtained using the OWA operator, while column 3 provides the corresponding run time, and column 4 lists the percentage difference between the

corresponding number of particles and the best goodness (in boldface). Column 5 reports the overall goodness obtained using the UAO operator, column 6 provides the corresponding run time, and column 7 lists the percentage difference between the corresponding number of particles and the best goodness (in boldface). For example, in Table 9.2, the best overall goodness (in boldface) using OWA is obtained with 30 particles. The overall goodness with different numbers of particles is then compared with the best overall goodness, and the percentage difference is listed. It is evident from Tables 9.2 to 9.6 that the best overall goodness was obtained when the number of particles was relatively high. With OWA, the best results for *n50* and *n40* were obtained with 30 particles, while for *n25* and *n15*, 25 particles resulted in the best solutions. Only in test case *n33* did a moderate number of particles, 20 in this case, demonstrate better performance.

As for UAO, the effect of the large swarm size was even more prominent, where it is noticed that the best results were obtained in all cases with a swarm size of 30. A small deviation from this trend was the case *n33* where 25 particles produced the best overall goodness.

Table 9.2: Effect of swarm size on overall goodness for *n50* with OWA and UAO. Time = Run time (in seconds), % Diff = % Difference. Statistically significant difference is in italics.

Number of particles	Goodness OWA	Time	% Diff	Goodness UAO	Time	% Diff
10	0.251 ±0.074	2057.7	*14.26*	0.333 ±0.005	2151.9	*0.89*
15	0.268 ±0.041	3041.9	6.94	0.334 ±0.004	3510.4	*0.72*
20	0.263 ±0.039	4291.1	*9.02*	0.335 ±0.004	4505.4	0.23
25	0.269 ±0.042	5352.3	6.72	0.335 ±0.002	5704.9	0.43
30	**0.287** ±0.034	6328.1	NA	**0.336** ±0.003	7074.9	NA

Table 9.3: Effect of swarm size on overall goodness for *n40* with OWA and UAO. Time = Run time (in seconds), % Diff = % Difference. Statistically significant difference is in italics.

Number of particles	Goodness OWA	Time	% Diff	Goodness UAO	Time	% Diff
10	0.296 ±0.039	544.9	*14.12*	0.338 ±0.005	545.3	*3.90*
15	0.326 ±0.044	782.5	3.61	0.341 ±0.012	829.5	*2.92*
20	0.318 ±0.036	1132.9	*6.20*	0.342 ±0.009	1081.8	*2.80*
25	0.316 ±0.023	1390.7	*6.78*	0.346 ±0.010	1428.2	1.50
30	**0.337** ±0.026	1736.7	NA	**0.351** ±0.012	1659.2	NA

Table 9.4: Effect of swarm size on overall goodness for *n33* with OWA and UAO. Time = Run time (in seconds), % Diff = % Difference. Statistically significant difference is in italics.

Number of particles	Goodness OWA	Time	% Diff	Goodness UAO	Time	% Diff
10	0.300 ±0.041	215.8	4.17	0.332 ±0.006	210.5	*2.06*
15	0.306 ±0.030	324.7	2.29	0.337 ±0.006	329.3	0.56
20	**0.313** ±0.053	445.3	NA	0.337 ±0.005	455.9	0.58
25	0.312 ±0.031	597.9	0.17	**0.339** ±0.005	560.6	NA
30	0.311 ±0.040	667.3	0.63	0.338 ±0.006	662.5	0.16

Table 9.5: Effect of swarm size on overall goodness for *n25* with OWA and UAO. Time = Run time (in seconds), % Diff = % Difference. Statistically significant difference is in italics.

Number of particles	Goodness OWA	Time	% Diff	Goodness UAO	Time	% Diff
10	0.306 ±0.038	62.6	*11.39*	0.330 ±0.009	58.0	*2.69*
15	0.325 ±0.036	92.1	4.70	0.330 ±0.004	91.5	*2.73*
20	0.333 ±0.031	118.7	2.20	0.335 ±0.005	120.3	1.13
25	**0.340** ±0.030	153.0	NA	0.337 ±0.008	159.3	0.60
30	0.327 ±0.032	187.1	4.17	**0.339** ±0.008	187.0	NA

Table 9.6: Effect of swarm size on overall goodness for *n15* with OWA and UAO. Time = Run time (in seconds), % Diff = % Difference. Statistically significant difference is in italics.

Number of particles	Goodness OWA	Time	% Diff	Goodness UAO	Time	% Diff
10	0.186 ±0.020	13.7	*16.98*	0.332 ±0.002	14.3	0.42
15	0.204 ±0.049	22.1	6.85	0.332 ±0.002	21.2	0.42
20	0.218 ±0.031	29.8	-0.09	0.332 ±0.001	28.7	0.32
25	**0.218** ±0.029	36.5	NA	0.332 ±0.002	35.9	0.42
30	0.216 ±0.024	43.9	0.91	**0.333** ±0.003	43.2	NA

A t-test validation of percentage difference in Tables 9.2 to 9.6 was also performed, and the statistically significant differences are italicized. An important observation in Tables 9.2 to 9.6 is that the lowest level of overall goodness was obtained when the number of particles was lowest. More specifically, having 10 particles resulted in the worst solutions in all cases and with both fuzzy operators. The only exception to this trend was *n15* when UAO was applied. In this instance, all particles from 10 to 25 resulted in the same overall goodness value. The improvement between the highest and the lowest overall goodness using the OWA and UAO operators is given in Tables 9.7 and 9.8 respectively. Table 9.7 shows that the improvement was generally between 10% and 15%, with the exception of *n33* where an improvement of 4.17% was observed. Moreover, t-test validation showed that all improvements, except that for *n33*, were statistically significant. As for UAO, the improvement was generally less than 4%, as shown in Table 9.8, and all improvements (except for *n15*) were statistically significant, as validated by the t-test.

A graphical representation of the results in Tables 9.2 to 9.6 is given in Figure

9.2. This figure shows the effect on overall goodness when the number of particles are varied from 10 to 30. This figure further strengthens the observations, noted above, that in general, increasing the number of particles positively affects the quality of overall goodness of the solution. For example, in Figure 9.2(a), the overall goodness increased with an increase in the number of particles for case *n50*. The trend is more obvious for OWA than for UAO. Similarly, for all other cases, this general trend is observed for both the OWA and UAO operators. Only in the instance of *n25* with OWA (Figure 9.2(d)) did the overall goodness increase up to 25 particles and then dropped with 30 particles.

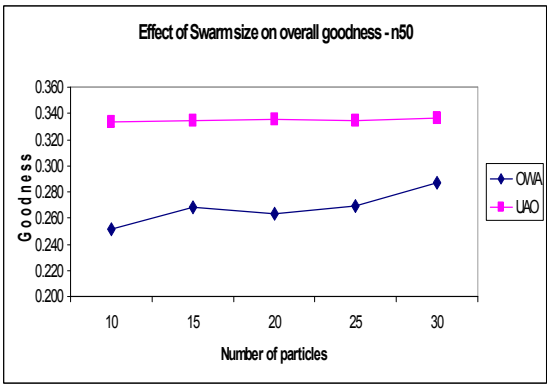
Table 9.7: Results for best and worst average overall goodness and their respective number of particles for OWA. Statistically significant improvement is in italics.

Case	Particles	Max. goodness	Particles	Min. goodness	% improvement
n15	25	0.218 ±0.029	10	0.186 ±0.020	*16.98*
n25	25	0.340 ±0.030	10	0.306 ±0.038	*11.39*
n33	20	0.313 ±0.053	10	0.300 ±0.041	4.17
n40	30	0.337 ±0.026	10	0.296 ±0.039	*14.12*
n50	30	0.287 ±0.034	10	0.251 ±0.074	*14.26*

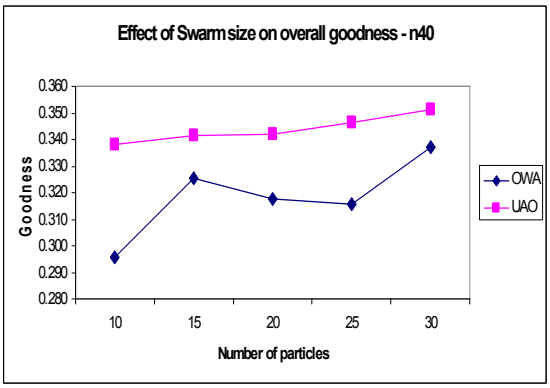
Table 9.8: Results for best and worst average overall goodness and their respective number of particles for UAO. Statistically significant improvement is in italics.

Case	Particles	Max. goodness	Particles	Min. goodness	% improvement
n15	30	0.333 ±0.003	10,15,20,25	0.332 ±0.002	0.42
n25	30	0.339 ±0.008	10	0.330 ±0.009	*2.69*
n33	25	0.339 ±0.005	10	0.332 ±0.006	*2.06*
n40	30	0.351 ±0.012	10	0.338 ±0.005	*3.90*
n50	30	0.336 ±0.003	10	0.333 ±0.005	*0.89*

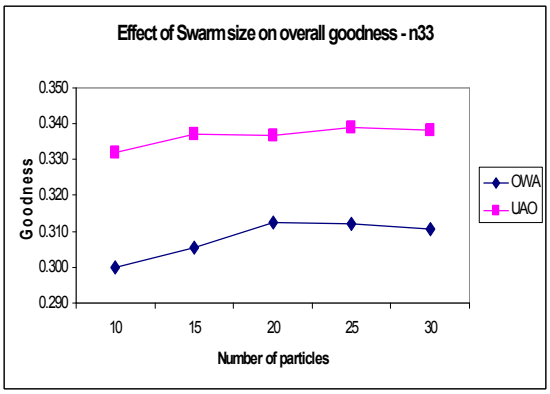
The above discussion and observations suggest that, in general, an increase in



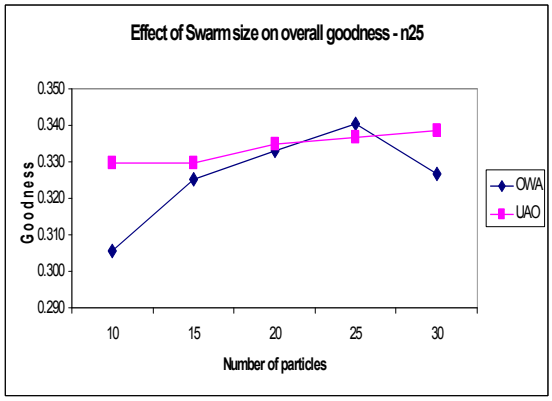
(a)



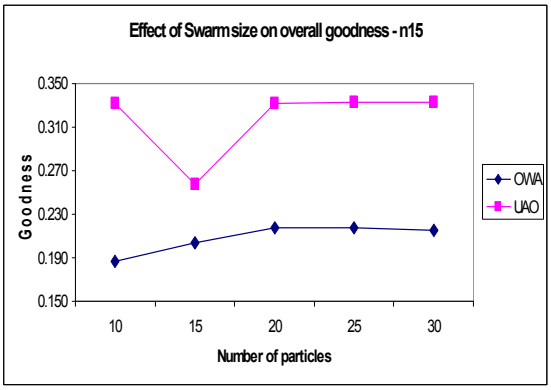
(b)



(c)



(d)



(e)

Figure 9.2: Effect of swarm size on overall goodness for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

the number of particles increases diversity and reduces the possibility of getting trapped in local minima, thereby resulting in higher quality solutions.

9.2.2 Effect of Acceleration Coefficients

The effect of acceleration coefficients was investigated with different values of the coefficients as given in Table 9.1. Other parameters were kept as follows: number of particles = 20, $w = 0.72$, and $V_{max} = 5$. Tables 9.9 and 9.10 respectively provide the results for OWA and UAO operators with respect to the three sets of acceleration coefficients. The values $c_1 = c_2 = 1.49$ (along with inertia weight = 0.72) were specifically chosen, since they are often used in the literature and they ensure convergence [244].

Table 9.9: Effect of acceleration coefficients on the test cases, for OWA. Good = average overall goodness, Time = Run time (in seconds). % imp shows the improvement achieved by one set of values of c_1 and c_2 over the other set of values. Statistically significant improvement is in italics.

Case	$c_1 = c_2 = 0.5$		$c_1 = c_2 = 1.49$		$c_1 = c_2 = 2.0$		% imp 0.5 vs 1.49	% imp 2.0 vs 1.49	% imp 2.0 vs 0.5
	Good	Time	Good	Time	Good	Time			
n15	0.218 ±0.031	29.8	0.224 ±0.048	29.3	0.218 ±0.043	31.2	-2.95	-2.78	0.0
n25	0.333 ±0.031	118.7	0.333 ±0.027	111.1	0.323 ±0.033	120.5	0.0	-2.91	-2.91
n33	0.313 ±0.053	445.3	0.303 ±0.033	497.9	0.312 ±0.029	546.7	3.14	3.11	-0.03
n40	0.318 ±0.036	1132.9	0.330 ±0.037	1248.5	0.327 ±0.041	1428.8	-4.04	-1.00	2.93
n50	0.263 ±0.039	4291.1	0.262 ±0.052	5285.4	0.275 ±0.043	5444.8	0.58	*4.89*	4.34

It is observed in Table 9.9 that each set of acceleration coefficients produced

Table 9.10: Effect of acceleration coefficients on the test cases, for UAO. Good = average overall goodness, Time = Run time (in seconds). % imp shows the improvement achieved by one set of values of c_1 and c_2 over the other set of values. Statistically significant improvement is in italics.

Case	$c_1 = c_2 = 0.5$		$c_1 = c_2 = 1.49$		$c_1 = c_2 = 2.0$		% imp 0.5 vs 1.49	% imp 2.0 vs 1.49	% imp 2.0 vs 0.5
	Good	Time	Good	Time	Good	Time			
n15	0.332 ±0.001	28.7	0.333 ±0.002	29.7	0.332 ±0.001	32.5	-0.12	-0.12	0.0
n25	0.335 ±0.005	120.3	0.338 ±0.010	119.3	0.337 ±0.009	127.4	-1.03	-0.51	0.52
n33	0.337 ±0.005	455.9	0.338 ±0.006	506.1	0.336 ±0.004	617.7	-0.25	-0.64	-0.39
n40	0.342 ±0.009	1081.8	0.328 ±0.060	1327.2	0.350 ±0.011	1280.5	*3.96*	*6.09*	2.22
n50	0.335 ±0.004	4505.4	0.336 ±0.004	5658.1	0.335 ±0.003	5766.1	-0.13	-0.13	0.0

results of the almost the same quality when compared with the other set. For example, values of $c_1 = c_2 = 0.5$ produced slightly better results than $c_1 = c_2 = 1.49$ for test cases *n50* and *n33*, but the latter set of coefficients performed better than the former for *n40* and *n15*. Similarly, $c_1 = c_2 = 2.0$ produced better results for some cases and worse results for others, when compared with $c_1 = c_2 = 1.49$ and $c_1 = c_2 = 0.5$. However, the t-test showed that, in general, the percentage improvements reported in Table 9.9 were not statistically significant.

With respect to the UAO operator, a trend similar to that of the OWA operator was observed. Table 9.10 shows that the percentage improvements achieved by any set of c_1 and c_2 compared to another set was at most 1% in the majority of cases. An exception from this trend was the case of *n40*, where $c_1 = c_2 = 0.5$ achieved an improvement of 3.96% over $c_1 = c_2 = 1.49$, $c_1 = c_2 = 2.0$ achieved an improvement of 6.09% over $c_1 = c_2 = 1.49$, and $c_1 = c_2 = 2.0$ achieved an improvement of

2.22% over $c_1 = c_2 = 0.5$. A t-test validation showed that all improvements were insignificant, with the exception of *n40* when comparing $c_1 = c_2 = 1.49$ with other sets of c_1 and c_2. In general, the results indicate that the convergence of PSO is independent of the acceleration coefficients with respect to the values used.

9.2.3 Effect of Inertia Weight

Table 9.11: Effect of inertia weight on the test cases, for OWA. Good = average overall goodness, Time = Run time (in seconds). % imp shows the improvement achieved by one value of w over the other value. Statistically significant improvement is in italics.

Case	w = 0.72		w = 0.95		w = 0.4		% imp 0.72 vs 0.95	% imp 0.72 vs 0.4	% imp 0.95 vs 0.4
	Good	Time	Good	Time	Good	Time			
n15	0.218 ±0.031	29.8	0.233 ±0.053	28.9	0.213 ±0.022	28.2	-6.76	2.40	*8.58*
n25	0.333 ±0.031	118.7	0.323 ±0.032	128.5	0.328 ±0.031	125.6	2.88	1.46	-1.46
n33	0.313 ±0.053	445.3	0.291 ±0.026	426.1	0.306 ±0.023	439.9	6.98	2.09	-5.26
n40	0.318 ±0.036	1132.9	0.322 ±0.028	1095.7	0.331 ±0.024	1105.1	-1.49	-4.28	-2.75
n50	0.263 ±0.039	4291.1	0.258 ±0.044	4285.1	0.273 ±0.040	3872.9	1.93	-3.53	-5.57

The effect of the inertia weight, w, is empirically investigated in this section. Tables 9.11 and 9.12 respectively show the results obtained for the fuzzy PSO with the OWA and UAO operators. The effect of w on performance was studied with three values, namely $w = 0.72$, $w = 0.95$, and $w = 0.4$. Other parameters were kept as follows: number of particles = 20, $c_1 = c_2 = 0.5$, and $V_{max} = 5$.

Table 9.11 suggests that there was no clear trend as which value of w produced

Table 9.12: Effect of inertia weight on the test cases, for UAO. Good = average overall goodness, Time = Run time (in seconds). % imp shows the improvement achieved by one value of w over the other value. Statistically significant improvement is in italics.

Case	w = 0.72		w = 0.95		w = 0.4		% imp 0.72 vs 0.95	% imp 0.72 vs 0.4	% imp 0.95 vs 0.4
	Good	Time	Good	Time	Good	Time			
n15	0.332 ±0.001	28.7	0.332 ±0.001	29.1	0.332 ±0.002	27.7	0.0	0.0	0.0
n25	0.335 ±0.005	120.3	0.331 ±0.005	127.2	0.333 ±0.008	125.4	1.03	0.70	-0.33
n33	0.337 ±0.005	455.9	0.337 ±0.005	455.6	0.340 ±0.008	428.1	0.0	-0.81	-0.88
n40	0.342 ±0.009	1081.8	0.344 ±0.011	1103.6	0.345 ±0.011	1025.4	-0.75	-0.91	-0.16
n50	0.335 ±0.004	4505.4	0.335 ±0.003	4528.5	0.335 ±0.004	4345.3	0.0	0.0	0.0

the best results. For example, $w = 0.72$ produced better solutions than $w = 0.95$ for *n25*, *n33*, and *n50*, but for *n15* and *n40*, $w = 0.95$ produced better results than $w = 0.72$. Similarly, comparisons of $w = 0.72$ with $w = 0.4$, and $w = 0.95$ with $w = 0.4$ did not show any clear pattern as which value of w performed better compared to the other. The statistical t-test also showed that none of the improvements, whether achieved by $w = 0.72$, $w = 0.95$, or $w = 0.4$, were significant. As for UAO, the difference between the overall goodness achieved by the three values of inertia weight was generally less than 1% for all test cases. The t-test showed that the improvements were insignificant. These observations suggest that the fuzzy PSO was insensitive to the inertia weight for both the OWA and UAO operators with respect to the three values of w used.

9.2.4 Effect of Velocity Clamping

The effect of velocity clamping was also empirically studied. Tables 9.13 and 9.15 respectively show the results obtained for fuzzy PSO with OWA and UAO operators. The effect was studied with three values of velocity clamping, with one value fixed at $V_{max} = 5$ for all cases, while the other two were variable, proportional to the test case size. These variable values were $\lceil V_{max} = 10\% \rceil$ and $\lceil V_{max} = 20\% \rceil$ of the test case size. The inspiration for taking 10% and 20% size of the test case comes from mutation rates in genetic algorithms. Note that both V_{max} in PSO and mutation rate in GA perturb the solution, and therefore the functions of both parameters is more or less the same. A number of studies [34, 115, 158, 159] have used the mutation rate up to 20% or more. Therefore, the basis of choosing a variable value of V_{max} is this observation. Other PSO parameters were kept as follows: number of particles = 20, $c_1 = c_2 = 0.5$, and inertia weight $w = 0.72$.

Table 9.13: Effect of velocity clamping on the test cases, for OWA. % imp shows the improvement achieved by one value of V_{max} compared to the other value. NA = Not Applicable.

Case	$V_{max} = 5$ Goodness	$V_{max} = 10\%$ Goodness	$V_{max} = 20\%$ Goodness	% imp 5 vs 10%	% imp 5 vs 20%	% imp 10% vs 20%
n15	0.218 ±0.031	0.220 ±0.048	0.212 ±0.038	-1.00	2.46	3.55
n25	0.333 ±0.031	0.328 ±0.036	The value of V_{max} is 5 here	1.54	NA	-1.54
n33	0.313 ±0.053	0.314 ±0.032	0.277 ±0.032	-0.48	*11.36*	*13.36*
n40	0.318 ±0.036	0.334 ±0.038	0.313 ±0.024	-5.30	1.52	6.93
n50	0.263 ±0.039	The value of V_{max} is 5 here	0.266 ±0.038	NA	-1.16	-1.15

Table 9.13 shows that velocity clamping did not significantly improve the so-

Table 9.14: Average algorithm run time (in seconds) for different values of V_{max} given in Table 9.13.

Test Case	Run time		
	$V_{max} = 5$	$V_{max} = 10\%$	$V_{max} = 20\%$
n15	29.8	28.8	28.9
n25	118.7	122.4	Same runtime as for $V_{max} = 5$
n33	445.3	433.0	452.5
n40	1132.9	1075.1	1101.9
n50	4291.1	Same runtime as for $V_{max} = 5$	4386.7

Table 9.15: Effect of velocity clamping on the test cases, for UAO. % imp shows the improvement achieved by one value of V_{max} compared to the other value. NA = Not Applicable.

Case	$V_{max} = 5$ Goodness	$V_{max} = 10\%$ Goodness	$V_{max} = 20\%$ Goodness	% imp 5 vs 10%	% imp 5 vs 20%	% imp 10% vs 20%
n15	0.332 ±0.001	0.331 ±0.001	0.332 ±0.001	0.263	-0.009	-0.27
n25	0.335 ±0.005	0.332 ±0.007	The value of V_{max} is 5 here	0.843	NA	-0.84
n33	0.337 ±0.005	0.337 ±0.006	0.339 ±0.007	-0.024	-0.609	-0.58
n40	0.342 ±0.009	0.346 ±0.013	0.342 ±0.010	-1.207	-0.055	1.15
n50	0.335 ±0.004	The value of V_{max} is 5 here	0.335 ±0.005	NA	0.095	0.10

lution. The overall goodness for the test cases varied between 1.0% and 6.93%, with the exception of *n33*. For *n33*, improvements of 11.36% (for comparison of $V_{max} = 5$ with $V_{max} = 20\%$) and 13.36% (for comparison of $V_{max} = 10\%$ with $V_{max} = 20\%$) were obtained. A t-test validation also showed that all improvements less than 13.36% were statistically insignificant. As for UAO, the results in Table 9.15 suggest a trend similar to that of OWA. It is observed in Table 9.15 that velocity clamping had a very slight impact on the quality of overall goodness, with all

Table 9.16: Average algorithm run time (in seconds) for different values of V_{max} given in Table 9.15.

Test Case	Run time		
	$V_{max} = 5$	$V_{max} = 10\%$	$V_{max} = 20\%$
n15	28.7	28.5	28.9
n25	120.3	119.9	Same runtime as for $V_{max} = 5$
n33	455.9	441.7	454.5
n40	1081.8	1098.6	1083.9
n50	4505.4	Same runtime as for $V_{max} = 5$	4867.0

values having less than 1.5% improvements. The t-test confirmed that all the improvements were statistically insignificant. In general, the results in Tables 9.13 and 9.15 suggest that velocity clamping did not have a significant effect on the quality of the overall goodness for the values used for V_{max}.

9.3 Comparison of OWA and UAO

Table 9.17 compares the OWA and UAO operators using linear regression analysis (performed with a confidence level of 95%), with the number of particles as the independent variable and the overall goodness of solution as the dependent variable. The objective of the regression analysis was to study the effect of increasing the number of particles on the overall goodness while using the OWA and UAO operators. The data for the analysis consisted of overall goodness for each particle set in Tables 9.2 to 9.6. Since, for each test case, 30 runs were done for each particle set, the regression coefficients in Table 9.17 was obtained using 150 values (5 values $\times$ 30 runs). For example, for test case *n15*, Table 9.6 shows the overall goodness for

Table 9.17: Comparison of OWA and UAO for FPSO.

Case	Regression coefficients		Ratio = $\frac{UAO}{OWA}$	Comment
	OWA	UAO		
n15	0.309	0.227	0.735	UAO increases goodness slower than OWA at the rate of 0.735
n25	0.238	0.472	1.983	UAO increases goodness almost twice as fast as OWA
n33	0.1	0.329	3.290	UAO increases goodness almost thrice as fast as OWA
n40	0.289	0.392	1.356	UAO increases goodness faster than OWA at the rate of 1.356
n50	0.213	0.254	1.192	UAO increases goodness faster than OWA at the rate of 1.192

OWA and UAO, with five different number of particles (from 10 up to 30). Table 9.17 also provides the regression coefficients for OWA and UAO in columns 2 and 3 respectively. The ratio of UAO and OWA is given in column 4. Based on this ratio, the sensitivity of OWA and UAO was evaluated with respect to the increasing swarm size. An interpretation of the ratio in column 4 is given in column 5.

In Table 9.17, it is observed that UAO was more sensitive than OWA with respect to the increasing number of particles. For almost all cases, UAO increased the overall goodness significantly faster than OWA. This is quite obvious for cases *n25* and *n33* where the rate of UAO is respectively twice and thrice that of OWA. For cases *n40* and *n50*, UAO also had a faster rate than OWA. Only for *n15*, OWA was faster than UAO, as observed in Table 9.17. Thus, the general conclusion is that, as the number of particles are increased, the rate at which UAO increases the overall goodness is significant as compared to the rate of the OWA operator.

9.4 Conclusions

A fuzzy multi-objective particle swarm optimization algorithm for the DLAN topology design problem was proposed and investigated in this chapter. The performance of the algorithm was evaluated with respect to different parameters of the fuzzy PSO algorithm. Results showed that the larger swarm sizes produced better results than medium or small sizes. An investigation of acceleration coefficients suggested that for the three set of values of acceleration coefficients used, there was no significant difference in the quality of final solutions obtained. Results also revealed that the fuzzy PSO was insensitive to the inertia weight, with respect to the three values used. As for velocity clamping, the results suggested that the parameter did not have a significant effect on the quality of the solutions with the three values used. With respect to the performance of OWA and UAO, it was found that, in general, UAO performed better than OWA.

The next chapter presents a comprehensive comparison of the techniques proposed and discussed in this thesis is presented.

Chapter 10

Comparison of Techniques

Chapters 5 to 9 presented a number of iterative algorithms as applied to solve the DLAN topology design problem. This chapter presents an overall comparison of the results obtained for each of the proposed algorithms. As mentioned earlier, the proposed algorithms are divided into two categories: ones which operate on single solutions, such as stochastic evolution, simulated evolution, and simulated annealing, and ones which are population-based swarm intelligence algorithms, namely ant colony optimization and particle swarm optimization. Therefore, the comparisons presented below are also categorized in the same way. The results for these comparisons have been presented and discussed in earlier chapters, but are consolidated here for the sake of clear comparisons.

10.1 Comparison of Single Solution Algorithms

This section compares the performance of the single solution algorithms presented in this thesis. Chapter 5 showed that the variant of variant of StocE with tabu

search characteristics (TFStocE) demonstrated the best performance among different versions of StocE algorithms. For the simulated evolution algorithm, the results in Chapter 6 suggested that the simulated evolution algorithm with tabu search characteristics and dynamic bias (DTFSimE) had the best performance among all proposed variants. As for simulated annealing, the variants were discussed in Chapter 7, and it was found that the simulated annealing algorithm with tabu search characteristics (TEFSA) was the best compared to the other variants. Therefore, this section mutually compares these best algorithms.

Table 10.1: Comparison of TFStocE, DTFSimE, and TEFSA using OWA. % imp denote percentage improvements. Statistically significant improvement is in italics.

Case	TFStocE Goodness	DTFSimE Goodness	TEFSA Goodness	% imp TEFSA vs TFStocE	% imp TEFSA vs DTFSimE	% imp DTFSimE vs TFStocE
n15	0.120 ± 0.053	0.206 ± 0.052	0.389 ± 0.056	*69.15*	*47.04*	*41.75*
n25	0.160 ± 0.035	0.240 ± 0.008	0.500 ± 0.125	*68.00*	*52.00*	*33.33*
n33	0.099 ± 0.045	0.229 ± 0.061	0.429 ± 0.192	*76.92*	*46.62*	*56.77*
n40	0.131 ± 0.053	0.340 ± 0.122	0.489 ± 0.047	*73.21*	*30.47*	*61.47*
n50	0.178 ± 0.044	0.350 ± 0.143	0.329 ± 0.121	*45.90*	-6.38	*49.14*

Table 10.1 compares TFStocE, DTFSimE, and TEFSA with respect to the overall goodness using the OWA operator. The average run time is given in Table 10.2. It is obvious from Table 10.1 that, in general, TEFSA produced the best results among the three schemes, also validated by the t-test. An exception was observed in the case of *n50*, where DTFSimE was able to achieve slightly better results than

Table 10.2: Average run time (in seconds) of algorithms in Table 10.1.

Test Case	Run time		
	TFStocE	DTFSimE	TEFSA
n15	4.6	134.2	89.5
n25	30.1	354.2	314.8
n33	36.2	1080.7	764.7
n40	145.1	2861.0	1499.5
n50	1341.1	7042.8	4295.4

Table 10.3: Comparison of TFStocE, DTFSimE, and TEFSA using UAO. % imp denote percentage improvements. Statistically significant improvement is in italics.

Case	TFStocE Goodness	DTFSimE Goodness	TEFSA Goodness	% imp TEFSA vs TFStocE	% imp TEFSA vs DTFSimE	% imp DTFSimE vs TFStocE
n15	0.245 ± 0.032	0.446 ± 0.061	0.365 ± 0.016	*32.88*	*-22.19*	*45.07*
n25	0.275 ± 0.008	0.301 ± 0.006	0.412 ± 0.064	*33.25*	*26.94*	*8.64*
n33	0.224 ± 0.032	0.303 ± 0.004	0.411 ± 0.072	*45.50*	*26.28*	*26.07*
n40	0.318 ± 0.032	0.297 ± 0.122	0.470 ± 0.079	*32.34*	*36.81*	-7.07
n50	0.256 ± 0.025	0.281 ± 0.000	0.374 ± 0.050	*31.55*	*24.87*	*8.90*

TEFSA. However, this improvement by DTFSimE was not statistically significant. It is also observed that TFStocE was the worst performer among the three schemes, with results much inferior to both DTFSimE and TEFSA.

As for UAO, the trends are very much similar to that of OWA. Observe from Table 10.3 that TEFSA demonstrated the best performance for almost all test cases. The only exception to this was case *n15* where DTFSimE had statistically better performance than TEFSA, as validated by the t-test. Again, TFStocE produced the

Table 10.4: Average run time (in seconds) of algorithms in Table 10.3.

Test Case	Run time		
	TFStocE	DTFSimE	TEFSA
n15	0.7	116.9	88.5
n25	16.0	312.9	322.4
n33	11.3	775.1	757.0
n40	143.1	2526.3	1564.4
n50	54.2	5013.3	3485.6

worst results. However, the extent of degradation in the quality of results produced by TFStocE compared to DTFSimE and TEFSA was not as significant as was observed for OWA. The average run time is given in Table 10.4.

10.2 Comparison of Population Based Algorithms

This section compares the performance of the two population based algorithms, namely, FACO and FPSO, developed in this thesis. The FACO and FPSO algorithms were proposed in Chapters 8 and 9 respectively. Tables 10.5 and 10.7 compares the two algorithms for the OWA and UAO operators respectively. Although the results were discussed in detail in the previous two chapters, these tables present the consolidated best results found in each of the two chapters along with the corresponding parameter setup.

With respect to the OWA operator, Table 10.5 (with average run time given in Table 10.6) suggests that FACO showed statistically better performance than FPSO for the majority of cases, as validated by the t-test. This observation is prominent in cases *n15, n25*, and *n33*. For *n40* and *n50*, FACO had a milder deterioration than FPSO. However, this deterioration was not statistically significant. In general,

Table 10.5: Comparison of FACO and FPSO for OWA. dep = pheromone deposit rate, evap = pheromone evaporation rate, % imp = percentage improvement achieved by FACO. OG = overall goodness. Statistically significant improvement is in italics.

Case	FACO				FPSO					% imp FACO vs FPSO
	ants	dep	evap	OG	par	V_{max}	c_1, c_2	w	OG	
n15	30	0.2	0	0.313 ± 0.032	20	5	0.5	0.95	0.233 ± 0.053	*25.56*
n25	30	0.6	0.2	0.430 ± 0.023	25	5	0.5	0.72	0.340 ± 0.030	*20.93*
n33	30	0.8	0.3	0.362 ± 0.022	20	10%	0.5	0.72	0.314 ± 0.032	*13.26*
n40	25	0.4	0.1	0.333 ± 0.028	30	5	0.5	0.72	0.337 ± 0.026	-1.20
n50	30	0.8	0.3	0.270 ± 0.032	30	5	0.5	0.72	0.287 ± 0.034	-6.30

Table 10.6: Average run time (in seconds) of algorithms in Table 10.5.

Test Case	Run time	
	FACO	FPSO
n15	50.3	28.9
n25	234.0	153.0
n33	553.3	433.0
n40	1521.2	1736.7
n50	5923.3	6328.1

Table 10.7: Comparison of FACO and FPSO for UAO. dep = pheromone deposit rate, evap = pheromone evaporation rate, % imp = percentage improvement achieved by FACO. OG = overall goodness. Statistically significant improvement is in italics.

Case	FACO				FPSO					% imp FACO vs FPSO
	ants	dep	evap	OG	par	V_{max}	c_1, c_2	w	OG	
n15	30	0.8	0.3	0.334 ± 0.002	20	5	1.49	0.72	0.333 ± 0.002	0.30
n25	30	0.4	0.1	0.363 ± 0.008	30	5	0.5	0.72	0.339 ± 0.008	*6.61*
n33	25	0.8	0.3	0.349 ± 0.006	20	5	0.5	0.4	0.340 ± 0.008	*2.58*
n40	30	0.6	0.2	0.352 ± 0.006	30	5	0.5	0.72	0.351 ± 0.012	0.28
n50	30	0.6	0.2	0.336 ± 0.004	30	5	0.5	0.72	0.336 ± 0.003	0.0

it can be claimed that FACO performed better than FPSO for OWA.

As for UAO, the trends in Table 10.7 (with average run time given in Table 10.8) are somewhat similar to that of OWA. In *n25* and *n33*, FACO showed statistically significant improvement in the quality of overall goodness compared to FPSO. However, for the other three cases, both FACO and FPSO showed equal performance as there was no statistically significant difference in the results. Thus, it can be fairly claimed that FACO also demonstrated better results than FPSO for UAO.

Since the swarm size is a common factor in both FACO and FPSO, it is important to highlight the effect of this factor on the improvement of results. An observation from Tables 10.5 and 10.7 shows the relation of swarm size with the best results: for both FACO and FPSO, it is observed that the best results were obtained for large swarm sizes. For example, for FACO, the best results were obtained when the

Table 10.8: Average run time (in seconds) of algorithms in Table 10.7.

Test Case	Run time	
	FACO	FPSO
n15	31.3	29.7
n25	268.5	187.0
n33	528.1	428.1
n40	1561.9	1659.2
n50	6478.8	7074.9

largest number of ants, i.e. 30, were chosen. There were some instances, such as *n40* in Table 10.5 and *n33* in Table 10.7, where a number of ants equal to 25 produced the best results. As for FPSO, the larger swarm size (25 and 30 particles) produced the best results. There were some instances in both tables where a medium swarm size of 20 produced the best results. In general, it can be said that the swarm size in the two swarm intelligence techniques was directly proportional to the improvement achieved in the quality of results.

10.3 Overall Comparison of OWA and UAO

This section provides an overall comparison of the OWA and UAO operators. This comparison is based on the results presented in Chapters 5 to 9. In Chapter 5, UAO produced much better results than OWA for TFStocE. Results in Chapter 6 revealed that for DTFSimE, UAO performed better than OWA for the number of hops and reliability objectives, and worse than OWA for the delay objective. However, for the cost objective, both UAO and OWA produced results of equal quality. With regard to TEFSA, results in Chapter 7 showed that UAO was better than OWA for the cost objective, and worse than OWA for the delay and number of hops objectives.

As for the reliability objective, both OWA and UAO showed equal performance. As far as FACO and FPSO are concerned, results in Chapters 8 and 9 suggested that UAO performed better than OWA for both algorithms. From the above discussion, it can be fairly claimed that UAO is preferred to OWA.

10.4 Overall Best Algorithm

In Section 10.1, it was found that in the category of single solution algorithms, TEFSA produced the best results among the three algorithms. As for the category of population based algorithms, the results in Section 10.2 suggested that FACO produced better results than FPSO. However, it will be interesting to compare the two best algorithms from each category. Tables 10.9 and 10.10 provide a comparison of TEFSA and FACO by using the OWA and UAO operators respectively. It is observed from both tables that TEFSA achieved statistically significant improvement over FACO for all test cases, and for both OWA and UAO.

Table 10.9: Comparison of FACO and TEFSA for OWA. dep = pheromone deposit rate, evap = pheromone evaporation rate, Time = run time (in seconds), % imp = percentage improvement achieved by TEFSA. Statistically significant improvement is in italics.

Case	FACO					TEFSA		% imp
	ants	dep	evap	Goodness	Time	Goodness	Time	
n15	30	0.2	0	0.313 ±0.032	50.3	0.389 ±0.056	89.5	*19.54*
n25	30	0.6	0.2	0.430 ±0.023	234.0	0.500 ±0.125	314.8	*14.00*
n33	30	0.8	0.3	0.362 ±0.022	553.3	0.429 ±0.192	764.7	*15.62*
n40	25	0.4	0.1	0.333 ±0.028	1521.2	0.489 ±0.047	1499.5	*31.90*
n50	30	0.8	0.3	0.270 ±0.032	5923.3	0.329 ±0.121	4295.4	*17.93*

Table 10.10: Comparison of FACO and TEFSA for UAO. dep = pheromone deposit rate, evap = pheromone evaporation rate, Time = run time (in seconds), % imp = percentage improvement achieved by TEFSA. Statistically significant improvement is in italics.

Case	FACO					TEFSA		% imp
	ants	dep	evap	Goodness	Time	Goodness	Time	
n15	30	0.8	0.3	0.334 ±0.002	31.3	0.365 ±0.016	88.5	*8.49*
n25	30	0.4	0.1	0.363 ±0.008	268.5	0.412 ±0.064	322.4	*11.89*
n33	25	0.8	0.3	0.349 ±0.006	528.1	0.411 ±0.072	757.0	*15.09*
n40	30	0.6	0.2	0.352 ±0.006	1561.9	0.470 ±0.079	1564.4	*25.11*
n50	30	0.6	0.2	0.336 ±0.004	6478.8	0.374 ±0.050	3485.6	*10.16*

10.5 Conclusion

An overall comparison of the proposed techniques was presented in this chapter. Since simulated annealing, simulated evolution, and stochastic evolution are algorithms which operate on a single solution, their best results were mutually compared. The comparison revealed that among TFStocE, DTFSimE, and TEFSA, it was TEFSA that generally produced the best results. On the other hand, ant colony optimization and particle swarm optimization are two swarm intelligence techniques that maintain and evolve a collection of candidate solutions. A comparison of the two swarm-based approaches showed that FACO had a better performance than FPSO. An overall comparison showed that TEFSA produced the best results among all algorithms proposed in this thesis. Moreover, an overall comparison of OWA and UAO showed that UAO performed better than OWA.

The next chapter provides a brief summary of the thesis and some directions for future research.

Chapter 11

Conclusion

This thesis addressed the problem of the topology design of distributed local area networks, modelled as a multi-objective optimization problem. The first main objective of the thesis was the design and analysis of some iterative heuristics and swarm intelligence algorithms to address the DLAN topology design problem. This objective was accomplished by engineering a number of algorithms, such as simulated evolution, stochastic evolution, simulated annealing, ant colony optimization, and particle swarm optimization for the DLAN topology design problem. The second main objective was to address the multi-objective nature of the problem, and was accomplished by using fuzzy logic to aggregate individual objectives into a multi-objective aggregation function. Hybridization of single-solution algorithms was also investigated and new hybrid algorithms for the DLAN topology design problem were proposed and evaluated.

The following sections briefly highlight the key findings and contributions of this thesis, followed by a short discussion on directions for future research.

11.1 Summary

Chapter 3 provided details on the formulation of a multi-objective topology design problem for distributed local area networks. All necessary problem-specific information, including assumptions, objectives, and constraints, were discussed in the chapter. The chapter also discussed the integration of the multiple design objectives into a single objective function using fuzzy logic.

Chapter 4 proposed and discussed a new fuzzy aggregation function, namely the unified and-or (UAO) operator. The proposed UAO operator was theoretically and empirically compared with the well-known ordered weighted average (OWA) operator. The UAO operator exhibited mathematical properties similar to that of the OWA operator, and empirically performed better than OWA for the test instances. The chapter also discussed a structured decision-making approach based on fuzzy logic, with specific application to the problem of topology design of distributed local area networks.

Chapter 5 proposed a fuzzy multi-objective technique based on the stochastic evolution algorithm, termed as 'FStocE'. A variant of the proposed stochastic evolution algorithm, 'TFStocE', was also proposed. This variant introduced tabu search characteristics in the FStocE algorithm. The two variants were mutually compared empirically, using the OWA and UAO operators. It was found that, in general, TFStocE produced better results than FStocE for both the OWA and UAO operators. Moreover, an investigation was also done on a dynamic value of R, which is an important parameter of the standard stochastic evolution algorithm. The results suggested that the proposed approach for computing a dynamic R produced inferior

solutions compared to the TFStocE algorithm, for both the OWA and UAO operators. As far as the effectiveness of the OWA and UAO operators are concerned, the investigation found that UAO performed much better than OWA in optimizing each of the four design objectives.

Chapter 6 proposed and investigated a fuzzy multi-objective algorithm based on a simulated evolution algorithm, namely FSimE. A variant of FSimE, known as 'TFSimE', was also proposed. TFSimE incorporated tabu search characteristics in the *allocation* phase of the FSimE algorithm. The comparison suggested that TFSimE generally produced better results than FSimE. This improvement was observed in both cases when OWA and UAO operators were used for aggregation of single objectives. Another issue investigated in the chapter was the usage of dynamic *bias* value. Since bias is the only parameter in the simulated evolution algorithm, its proper value has an impact on the quality of the solution. Results suggested that the proposed approach for TFSimE based on dynamic bias value, denoted as DTFSimE, produced better results than TFSimE for all test cases with respect to OWA and the majority of test cases for UAO. As far as the relative performance of the OWA and UAO operators was concerned, it was found that UAO performed much better than OWA for the *number of hops* and *reliability* design objectives, while UAO had an inferior performance for the *cost* and *delay* objectives.

Chapter 7 investigated the effectiveness of a fuzzy simulated annealing algorithm for the DLAN topology design problem. A fuzzy simulated annealing algorithm, termed 'FSA', was proposed. Two variants of FSA were also developed. The first variant introduced tabu search characteristics into FSA, and was named TFSA. The second variant, namely TEFSA, introduced simulated evolution characteristics into

TFSA. A comparison of the three variants of the simulated annealing algorithms revealed that, generally, TEFSA produced better results than TFSA and FSA. This trend was observed for both the OWA and UAO operators. Moreover, mutual comparison of OWA and UAO with respect to the four design objectives suggested that both operators had more or less similar results. Another issue discussed in the chapter was the proposal of a dynamic value of an important parameter of the simulated annealing algorithm, namely the Markov chain, M. Although the proposed computation of dynamic M produced lower quality results than FSA, degradation in quality was not very great.

Chapter 8 proposed and discussed a fuzzy multi-objective ant colony optimization algorithm. Since heuristic value is an important factor of the ant colony algorithm, a fuzzy heuristic value was proposed. Furthermore, since pheromone deposit and evaporation, and the number of ants, are important parameters that play a key role in the algorithm's search direction, empirical analysis was done to study the effect of these parameters. As far as pheromone deposit and evaporation was concerned, better results were obtained when the difference in pheromone deposit and evaporation rates was high. This was a general trend for both OWA and UAO. As for the number of ants, relatively better results were obtained when the number was high. As for the mutual comparison of OWA and UAO, results suggested that UAO produced better results in the context of its usage in the fuzzy ant colony optimization algorithm.

Chapter 9 presented a fuzzy multi-objective particle swarm optimization algorithm for the DLAN topology design problem. A preliminary analysis of the effect of the number of PSO parameters was provided. With respect to swarm sizes, it

was found that larger sizes produced the best results compared to medium or small sizes. With respect to acceleration coefficients, results suggested that having different values of acceleration coefficients had no significant effect on the quality of final solutions obtained, with respect to the three values tested. A similar observation was made for inertia weight, where results revealed that the fuzzy PSO was insensitive to the inertia weight to a considerable extent, for the three values of the inertia weight used. As for velocity clamping, results suggest that the parameter also did not have a very significant effect on the quality of the solution with respect to the tested values. With respect to the performance of OWA and UAO, it was found that, in general, UAO performed better than OWA.

Chapter 10 provided an overall comparison of the techniques proposed in this thesis. Since simulated annealing, simulated evolution, and stochastic evolution are algorithms which operate on a single solution, their best results were mutually compared. On the other hand, ant colony optimization and particle swarm optimization are two swarm intelligence techniques, which were compared with one another. The comparison revealed that among TFStocE, DTFSimE, and TEFSA, the general trend is that TEFSA produced the best results. As for FACO and FPSO, the comparison suggested that FACO had a better performance than FPSO. Also, an overall comparison showed that TEFSA produced the best results among all algorithms proposed in this thesis. Moreover, an overall comparison of OWA and UAO showed that UAO had better performance than OWA.

11.2 Future Research

Some directions for future research are summarized below.

Efficient Approaches for Dynamic Parameters

Chapters 5, 6, and 7 discussed some approaches to dynamically adjust parameter values. However, most of these approaches did not prove to be efficient in producing high-quality solutions. More research is needed to devise highly efficient ways to deal with dynamic assignment of parameter values. Furthermore, the parameters in ACO and PSO are statically assigned by the user. Some research is also needed to develop mechanisms to dynamically adjust their parameter values.

Hybridization of tabu search with ACO and PSO

Results in this thesis suggested that introducing tabu search characteristics into different algorithms resulted in better solutions. However, the hybridization was done only on algorithms which operate on single solutions, such as simulated annealing, simulated evolution, and stochastic evolution. The promising results encourage research on incorporating tabu search characteristics or features of simulated evolution into swarm intelligence algorithms.

Extension of Fuzzy PSO to the *lbest* Model

The fuzzy PSO introduced in Chapter 9 made use of the *gbest* PSO model. Results for the proposed fuzzy PSO suggested that the algorithm was not very effective as compared to ACO. Effectiveness of fuzzy PSO can possibly be enhanced by extending the algorithm for the *lbest* model, since previous research has suggested that the *lbest* model may be more effective due to its better ability to escape local minima. Therefore, a recommendation is to exploit the *lbest* model for fuzzy PSO.

More In-depth Study of the Effects of the PSO parameters

Chapter 9 provided a preliminary analysis of the FPSO algorithm with at most three values for each parameter. More in-depth study is required to asses the effects of the PSO paramterers.

Application of Other Techniques to DLAN Topology Design Problem

Other techniques such as genetic algorithms, differential evolution, and estimation of distribution methods have been applied to a number of optimization problems. A recommendation is that these techniques be engineered for the DLAN topology design problem and that a comparative study be done with the techniques proposed in this thesis.

Other Aggregation Techniques

The multi-objective aspects of the DLAN topology design problem presented in this paper were addressed by fuzzy logic based aggregation functions such as UAO and OWA. However, other techniques presented in Chapter 2 should also be exploited.

Application of the UAO Operator to other Multi-objective Problems

The study of the UAO operator showed better performance as compared to the OWA operator for the DLAN topology design problem. Therefore, application of UAO to other multi-objective optimization problems should also be studied.

Bibliography

[1] H. Adiche. *Fuzzy Genetic Algorithm for VLSI Floorplan Design.* MS Thesis, King Fahd University, Saudi Arabia, 1997.

[2] K. K. Aggarwal and S. Rai. Reliability Evaluation in Computer Communication Networks. *IEEE Transactions on Reliability*, 30(1):32–35, 1981.

[3] C. Ai-ling, Y. Gen-ke, and W. Zhi-ming. Hybrid Discrete Particle Swarm Optimization Algorithm for Capacitated Vehicle Routing Problem. *Journal of Zhejiang University*, 4(7):607–614, 2006.

[4] A. Al-Mulhem, A. Amin, and H. Youssef. Stochastic Evolution Algorithm for Technology Mapping. In *8th Great Lakes Symposium on VLSI*, pages 380–385, February 1998.

[5] I. Alaya, C. Solnon, and K. Ghedira. Ant Colony Optimization for Multi-objective Optimization Problems. In *19th IEEE International Conference on Tools with Artificial Intelligence*, pages 450–457, 2007.

[6] F. Altiparmak and B. Dengiz. Reliability Estimation of Computer Communication Networks: ANN Models. In *8th IEEE International Symposium on Computers and Communication*, pages 1–6, 2003.

[7] M. Alves and J. Climaco. A Review of Interactive Methods for Multiobjective Integer and Mixed-integer Programming. *European Journal of Operations Research*, 180(1):99–115, 2007.

[8] P. Angeline. *Evolutionary Optimization Versus Particle Swarm Optimization: Philosophy and Performance Differences*. V. W. Porto, N. Saravanan, D. Waagen, and A. Eiben (Eds.), Evolutionary Programming VII, pages 601-610. Springer, 1998.

[9] P. Angeline. Using Selection to Improve Particle Swarm Optimization. In *IEEE Congress on Evolutionary Computation*, pages 84–89, 1998.

[10] D. Angus. *The Current State of Ant Colony Optimization Applied to Dynamic Problems*. Technical Report TR 009, University of Melbourne, Australia, 2006.

[11] R. Armafianzas and J. Lozano. A Multiobjective Approach to the Portfolio Optimization Problem. In *IEEE Congress on Evolutionary Computation*, pages 1388–1395, 2005.

[12] M. Atiqullah and S. Rao. Reliability Optimization of Communication Networks Using Simulated Annealing. *Microelectron Reliability*, 33(9):1303–1319, 1993.

[13] H. Bandemer and S. Gottwald. *Fuzzy Sets, Fuzzy Logic, Fuzzy Methods with Applications*. John Wiley & Sons, 1996.

[14] S. Bandyopadhyay, S. Saha, U. Maulik, and K. Deb. A Simulated Annealing-Based Multiobjective Optimization Algorithm: AMOSA. *IEEE Transactions on Evolutionary Computation*, 12(3):269–283, 2009.

[15] B. Baran and M. Schaerer. A Multiobjective Ant Colony System for Vehicle Routing Problem with Time Windows. In *21st IASTED International Conference on Applied Informatics*, pages 97–102, 2004.

[16] T. Bartz-Beielstein, P. Limbourg, K. Parsopoulos, M. Vrahatis, J. Mehnen, and K. Schmitt. Particle Swarm Optimizers for Pareto Optimization with Enhanced Archiving Techniques. In *IEEE Congress on Evolutionary Computation*, pages 1780–1787, 2003.

[17] U. Baumgartner, C. Magele, and W. Renhart. Pareto Optimality and Particle Swarm Optimization. *IEEE Transactions on Magnetics*, 40(2):1172–1175, 2004.

[18] J. C. Bezdek, B. Spillmann, and R. Spillmann. Fuzzy Relation Spaces for Group Decision Theory: An Application. *Fuzzy Sets & Systems*, 4:5 – 14, 1979.

[19] A. L. Blumel, E. G. Hughes, and B. A. White. Fuzzy Autopilot Design using a Multiobjective Evolutionary Algorithm. In *IEEE Congress on Evolutionary Computation*, pages 80–83, 2000.

[20] E. Bonabeau, M. Dorigo, and G. Thraulaz. *From Natural to Artificial Swarm Intelligence*. Oxford University Press, 1999.

[21] G. Bordogna, M. Fedrizzi, and G. Pasi. A Linguistic Modelling of Consensus in Group Decision Making based on OWA Operators. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 27(1):126–133, January 1997.

[22] M. S. Bright and T. Arslan. Multiobjective Design Strategies for High-level Low-power Design of DSP Systems. In *IEEE International Symposium on Circuits and Systems*, pages 80–83, 1999.

[23] B. Bullnheimer, R. Hartl, and C. Strauss. *An Improved Ant System Algorithm for the Vehicle Routing Problem.* Technical Report POM-10/97, Institute of Management Science, University of Vienna, 1997.

[24] B. Bullnheimer, R. Hartl, and C. Strauss. *Applying the Ant System to the Vehicle Routing Problem.* I. H. Osman, S. Voβ, S. Martello and C. Roucairol (Eds.), Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academics, 1998.

[25] B. Bullnheimer, R. Hartl, and C. Strauss. A New Rank Based Version of the Ant System: A Computational Study. *Central European Journal for Operations Research and Economics*, 7(1):25–38, 1999.

[26] R. Caballero, L. Rey, F. Ruiz, and M. Gonzalez. *An Algorithmic Package for the Resolution and Analysis of Convex Multiple Objective Problems.* G. Fandel, T. Gal, (Eds.), 12th International Conference on Multiple Criteria Decision Making, Germany, Springer-Verlag, pages 275-284, 1997.

[27] S. Chaharsooghi and A. Kermani. An intelligent multi-colony multi-objective ant colony optimization (ACO) for the 01 knapsack problem. In *IEEE Congress on Evolutaionry Computation*, pages 1195–1202, 2008.

[28] S. Chamberland and S. Pierre. On the expansion problem of cellular wireless networks. In *4th International Workshop on Mobile and Wireless Communications Network*, pages 25–29, 2002.

[29] A. Charnes and W. W. Cooper. *Management Models and Industrial Applications of Linear Programming*. John Wiley, 1961.

[30] A. Chattopadhyay and C. Seeley. A Simulated Annealing Technique for Multiobjective Optimization of Intelligent Structures. *Smart Materials & Structures*, 3(3):98–106, 1994.

[31] P. Chen and C. Huang. Biobjective Power Dispatch using Goal-attainment Method and Adaptive Polynomial Networks. *IEEE Transactions on Energy Conversion*, 19(4):741–747, 2004.

[32] M. Chiampi, C. Ragusa, and M. Repetto. Fuzzy Approach for Multiobjective Optimization in Magnetics. *IEEE Transactions on Magnetics*, 32(3):1234 – 1237, 1996.

[33] H. Cho, S. Oh, and D. Choi. A New Evolutionary Programming Approach Based on Simulated Annealing with Local Cooling Schedule. In *IEEE World Congress on Computational Intelligence*, pages 598–602, May 1998.

[34] H. Cho, B. Wang, and S. Roychowdhury. Automatic Rule Generation for Fuzzy Controllers using Genetic Algorithms: A Study on Representation Scheme and Mutation Rate. In *IEEE World Congress on Computational Intelligence*, pages 1290–1295, 1998.

[35] C. Chow and H. Tsui. Autonomous Agent Response Learning by a Multi-species Particle Swarm Optimization. In *IEEE Congress on Evolutionary Computation*, pages 778–785, 2004.

[36] C. Coello-Coello. *An Empirical Study of Evolutionary Techniques for Multiobjective Optimization in Engineering Design*. PhD thesis, Department of Computer Science, Tulane University, 1996.

[37] C. Coello-Coello. *Ant Colony System for the Design of Combinational Logic Circuit*. J. Miller, A. Thompson, P. Thomson and T. Fogarty (Eds.), Evolvable Systems: From Biology to Hardware, pages 21-30, 2000.

[38] C. Coello-Coello. A Short Tutorial on Evolutionary Multiobjective Optimization. In *IEEE/ACM 1st International Conference on Evolutionary Multi-Criterion Optimization, Lecutre Notes in Computer Science, Vol. 1993, Springer*, pages 21 – 35, 2001.

[39] C. A. Coello-Coello. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems*, 1(3):269 – 308, 1999.

[40] C. A. Coello-Coello and M. Lechuga. MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization. In *IEEE Congress on Evolutionary Computation*, pages 1051–1056, 2002.

[41] J. Cohon. *Multiobjective Programming and Planning*. Academic Press, New York, 1978.

[42] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed Optimization by Ant Colonies. In *European Conference on Artificial Life*, pages 134–142, 1991.

[43] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant System for Job-shop Scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science*, 34:39–53, 1994.

[44] D. Corne, M. Dorigo, and F. Glover (Eds.). *New Ideas in Optimization*. McGraw-Hill, 1999.

[45] D. Costa and A. Hertz. Ants can Color Graphs. *Journal of the Operational Research Society*, 48:295–305, 1997.

[46] Jack Crosby. *Computer Simulation in Genetics*. John Wiley & Sons, 1973.

[47] D. Cvetković and I. Parmee. Preferences and Their Application in Evolutionary Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation*, 6(1):42–57, 2002.

[48] L. Dae-Hyun, C. Hoon, P. Lae-Jeong, H. Cheol, and H. Seung. A Stochastic Evolution Algorithm for the Graph Covering Problem and its Application to the Technology Mapping. In *IEEE Congress on Evolutionary Computation*, pages 475 –479, May 1996.

[49] C. Darwin. *Inception of Darwin's Evolutionary Theory, http://en.wikipedia.org/wiki/ Charles_Darwin*. Retrieved on April 19, 2008.

[50] I. Das and J. Dennis. Normal-boundary Interaction: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems. *SIAM Journal of Optimization*, 8:631–657, 1998.

[51] M. Dawande and R. Gupta. An Integer-Programming Approach to the Bi-criteria Multicasting Problem in Optical Networks. *IEEE Transactions on Communications*, 55(4):752–765, 2007.

[52] R. de Silva and G. Ramalho. Ant System for the Set Covering Problem. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 3129 –3133, October 2001.

[53] K. Deb. *Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems.* Technical Report CI-49/98, University of Dortmund, Germany, 1998.

[54] K. Deb and D. Goldberg. An Investigation of Niche and Species Formation in Genetic Function Optimization. In *3rd International Conference on Genetic Algorithms*, pages 42–50, 1989.

[55] B. Dengiz and C. Alabas. A Tabu Search Algorithm for Computer Networks Design. *Problems in Modern Applied Mathematics*, 4(2):363–366, 2000.

[56] B. Dengiz and C. Alabas. A Simulated Annealing Algorithm for Design of Computer Communication Networks. In *World Multiconference on Systemics, Cybernetics, and Informatics*, pages 188–193, 2001.

[57] B. Dengiz, F. Altiparmak, and A. Smith. Efficient Optimization of All-Terminal Reliable Networks Using an Evolutionary Approach. *IEEE Transactions on Reliability*, 46(1):18–26, 1997.

[58] B. Dengiz, F. Altiparmak, and A. Smith. Local Search Genetic Algorithm for Optimal Design of Reliable Network. *IEEE Transactions on Evolutionary Computation*, 1(3):179–188, 1997.

[59] K. Doerner, W. Gutjahr, R. Hartl, C. Strauss, and C. Stummer. Ant Colony Optimization in Multiobjective Portfolio Selection. In *4th International Conference on Metaheuristics*, pages 125–131, 2001.

[60] K. Doerner, R. F. Hartl, and M. Teiman. Are COMPETants More Competent for Problem Solving? The Case of Full Truckload Transportation. *Central European Journal of Operations Research*, 11(3):115–141, 2003.

[61] J. Dombi. A General Class of Fuzzy Operators, the De Morgan Class of Fuzzy Operators and Fuzziness Measures Induced by Fuzzy Operators. *Fuzzy Sets and Systems*, 8:149–163, 1982.

[62] J. Dombi. Basic Concepts for a Theory of Evaluation: The Aggregative Operator. *European Journal of Operational Research*, 10:282–293, 1982.

[63] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, 1992.

[64] M. Dorigo, G. Di Caro, and L. Gambardella. *Ant Algorithms for Discrete Optimization*. Tech. Rep. IRIDIA/98-10, University of Brussels, 1998.

[65] M. Dorigo, M. Maniezzo, and A. Colorni. *The Ant Systems: An Autocatalytic Optimizing Process*. Technical Report 91-016, Milan Polytechnic, 1991.

[66] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B*, 26:29–42, 1996.

[67] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B*, 26(1):1 – 13, 2006.

[68] M. Dorigo and T. Stützle. *The Ant Colony Optimization Metaheuristic*. D. Corne and M. Dorigo and F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, pp. 11-32, 1999.

[69] D. Dubois and H. Prade. Operations in Fuzzy-valued Logic. *Information and Control*, 43:224–240, 1979.

[70] D. Dubois and H. Prade. A Class of Fuzzy Measures Based on Triangular Norms. *International Journal of General Systems*, 8:105–116, 1982.

[71] L. Duckstein. *Multiobjective Optimization in Structural Design: The Model Choice Problem*. North-Holland, Amsterdam, 1984.

[72] R. Eberhart and X. Hu. Human Tremor Analysis Using Particle Swarm Optimization. In *IEEE Congress on Evolutionary Computation*, pages 1927–1930, 1999.

[73] R. Eberhart and J. Kennedy. A New Optimizer using Particle Swarm Theory. In *6th International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[74] R. Eberhart, P. Simpson, and R. Dobbins. *Computational Intelligence PC Tools*. Academic Press, 1996.

[75] R. Elbaum and M. Sidi. Topological Design of Local-Area Networks Using Genetic Algorithm. *IEEE/ACM Transactions on Networking*, pages 766–778, October 1996.

[76] T. A. Ely, W. A. Crossley, and E. A. Williams. Satellite Constellation Design for Zonal Coverage using Genetic Algorithms. In *8th AAS/AIAA Space Flight Mechanics Meeting*, pages 124–129, 1998.

[77] A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. John Wiley Sons, 2005.

[78] A. P. Engelbrecht and A. Ismail. Training Product Unit Neural Networks. *Stability and Control: Theory and Application*, 2(2):59–74, 1999.

[79] C. Ersoy and S. Panwar. Topological Design of Interconnected LAN/MAN Networks. *IEEE Journal on Selected Area in Communications*, 24(8):1172–1182, 1993.

[80] L. R. Esau and K. C Williams. On Teleprocessing System Design. A Method for Approximating the Optimal Network. *IBM System Journal*, 5:142–147, 1966.

[81] L. Escudero. An Inexact Algorithm for the Sequential Ordering Problem. *European Journal of Operations Research*, 37:232–253, 1998.

[82] H. Etawil and A. Vannelli. Utility Function Based Hybrid Algorithm for Channel Routing. In *IEEE International Symposium on Circuits and Systems*, pages 258–261, 1998.

[83] P. Fetterolf. Design of Data Networks with Spanning Tree Bridges. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 298–300, 1990.

[84] J. Fieldsend and S. Singh. A Multiobjective Algorithm Based Upon Particle Swarm Optimisation, An Efficient Data Structure and Turbulence. In *U.K. Workshop on Computational Intelligence*, pages 37–44, 2002.

[85] D.B. Fogel. An Introduction to Simualted Evolutionary Optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, Jan 1994.

[86] C. Fonseca and P. Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion, and Generalization. In *5th International Conference on Genetic Algorithms*, pages 416–423, 1993.

[87] C. Fonseca and P. Fleming. Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms - Part 1: A Unified Formulation. *IEEE Transaction on Systems, Man, and Cybernetics - Part A*, 28(1):26–37, 1998.

[88] A. Fortin, N. Hail, and B. Jaumard. A Tabu Search Heuristic for the Dimensioning of 3G Multi-service Networks. In *IEEE Wireless Communications and Networking Conference*, pages 1439–1447, 2003.

[89] M. Frank. On the Simultaneous Associativity of $F(x, y)$ and $x + y - F(x, y)$. *Aequationes Mathematicae*, 19:194–226, 1979.

[90] A. Fraser. Simulation of Genetic Systems by Automatic Digital Computers. *Australian Journal of Biological Sciences*, 10:484–491, 1957.

[91] A. Fraser and D. Burnell. *Computer Models in Genetics*. McGraw-Hill, 1970.

[92] L. Gambardella and M. Dorigo. *Ant-Q: A Reinforcement Learning Approach to the Travelling Salesman Problem*. 12th International Conference on Machine Learning, A. Prieditis and S. Russell (Eds.), Morgan Kaufmann, pages 252-260, 1995.

[93] L. Gambardella, E.D. Taillard, and G. Agazzi. *ACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows*. D. Corne, M. Dorigo, and F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, pages 63-76, 1999.

[94] L.M. Gambardella and M. Dorigo. Solving Symmetric and Asymmetric TSPs by Ant Colonies. In *IEEE Congress on Evolutionary Computation*, pages 622–627, 1996.

[95] A. Gaspar Cunha, P. Oliveira, and A. J. Covas. *Genetic Algorithms in Multi-objective Optimization Problems: an Application to Polymer Extrusion*. A. S. Wu (Ed.), Genetic and Evolutionary Computation Conference, pages 129-130, 1999.

[96] S. Gass and T. Saaty. The Computational Algorithm for the Parametric Objective Function. *Naval Research Logistics Quarterly*, 2:39 – 45, 1955.

[97] F. Gembicki. *Vector Optimization for Control with Performance and Parameter Sensitivity Indices*. Ph.D. Thesis, Case Western Reserve University, USA, 1974.

[98] M. Gen, K. Ida, and J. Kim. A Spanning Tree-Based Genetic Algorithm for Bicriteria Topological Network Design. In *IEEE Congress on Evolutionary Computation*, pages 164–173, May 1998.

[99] R. Ghazi and A. Arabpour. Optimal Multi-Objective VAr Planning Using Accelerated Ant Colony and Analytical Hierarchy Process Methods. In *IEEE/PES Transmission and Distribution Conference & Exhibition: Asia and Pacific*, pages 1–7, 2005.

[100] F. Glover. Tabu Search - Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[101] F. Glover. *Tabu Search: A Tutorial. Technical Report*. Graduate School of Business Administration, University of Colorado at Boulder, 1990.

[102] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[103] D. Goldberg and R. Lingle. Alleles, Loci, and the Traveling Salesman Problem. In *1st International Conference on Genetic Algorithms*, pages 154–159, 1985.

[104] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, 1989.

[105] S. Goss, S. Aron, J. Deneubourg, and J. Pasteels. Self-organized Shortcuts in the Argentine Ant. *Naturwissenschaften*, 76:579–581, 1989.

[106] M. Gravel, W. L. Price, and C. Gagne. Scheduling Continuous Casting of Aluminium using a Multiple Objective Ant Colony Optimization Metaheuristic. *European Journal of Operational Research*, 143(1):218–229, 2002.

[107] P. Gray, W. Hart, L. Painton, C. Phillips, M. Trahan, and J. Wagner. A Survey of Global Optimization Methods. In *Sandia National Laboratories, http://www.cs.sandia/gov/opt/survey*, 1997.

[108] G. Greenwood, X. Hu, and J. D'Ambrosio. *Fitness Functions for Multiple Objective Optimization Problems: Combining Preferences with Pareto Rankings.* R. Belew and M. Vose (Eds.), Foundation of Genetic Algorithms, Vol. 4, Morgan-Kaufmann, pages 437-455, 1997.

[109] J. Gu, Q. Tan, N. Li, J. Zhang, and N. Mao. A New ACO with Immune Ability. In *5th International Conference on Machine Learning and Cybernetics*, pages 4278 – 4281, 2006.

[110] A. Gupta and W. Dally. Topology Optimization of Interconnection Networks. *IEEE Computer Architecture Letters*, 5(1):10–13, 2006.

[111] S. Habib. Redesigning Network Topology with Technology Considerations. In *9th IFIP/IEEE International Symposium on Integrated Network Management*, pages 207–219, May 2005.

[112] Y.Y. Haimes, L. S. Lasdon, and D.A. Wismer. On a Bicriterion Formulation of the Problems of Integrated System Identification and System Optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 1:296 – 297, 1971.

[113] H. Hamacher. Ueber Logische Verknupfungen Unschalfer Aussagen und deren Zugehoerige Bewertungs-funktione. *Progress in Cybernetics and Systems Research*, 3:276–288, 1978.

[114] J. Harmatos, A. Szentes, and I. Godor. Planning of Tree-topology UMTS Terrestrial Access Networks. In *11th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, Vol. 1*, pages 353 – 357, 2000.

[115] R. Haupt. Optimum Population Size and Mutation Rate for a Simple Real Genetic Algorithm that Optimizes Array Factors. In *IEEE Antennas and Propagation Society International Symposium*, pages 1034 – 1037, 2000.

[116] C. Heitzinger. *Simulation and Inverse Modelling of Semiconductor Manufacturing Processes*. PhD Thesis, Vienna University of Technology, http://www.iue.tuwien.ac.at/phd/heitzinger/node30.html, 2002.

[117] S.L. Ho, Y. Shiyou, N. Guangzheng, E. Lo, and H. Wong. A Particle Swarm Optimization Based Method for Multiobjective Design Optimizations. *IEEE Transactions on Magnetics*, 41(5):1756–1759, 2005.

[118] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[119] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, 1984.

[120] X. Hu and R. Eberhart. Multiobjective Optimization using Dynamic Neighborhood Particle Swarm Optimization. In *IEEE Congress on Evolutionary Computation*, pages 1677–1681, May 2002.

[121] X. Hu, R. Eberhart, and Y. Shi. Particle Swarm with Extended Memory for Multiobjective Optimization. In *IEEE Swarm Intelligence Symposium*, pages 193–197, 2003.

[122] Y. Ijiri. *Management Goals and Accounting for Control*. In E. Atrek, R. H. Gallagher, K. M. Ragsdell, and O. C. Zienkiewicz (Eds.), New Directions in Optimum Structural Design, John Wiley & Sons, 1965.

[123] S. Iredi, D. Merkle, and M. Middendorf. Bi-Criterion Optimization with Multi Colony Ant Algorithms. In *IEEE/ACM 1st International Conference on Evolutionary Multi-Criterion Optimization, Lecture Notes in Computer Science, Vol. 1993, Springer*, pages 359–372, 2001.

[124] H. Jgou and C. Guillemot. Entropy Coding With Variable-Length Rewriting Systems. *IEEE Transactions on Communications*, 55(3):444–452, 2007.

[125] W. Jakob, M. Gorges-Schleuter, and C. Blume. *Application of Genetic Algorithms to Task Planning and Learning*. R. Manner and B. Manderick (Eds.), 2nd Workshop on Parallel Problem Solving from Nature, North-Holland, pages 291-300, 1992.

[126] Y. Jeon, J. C. Kim, J. O. Kim, J. Shin, and K. Lee. An Efficient Simulated Annealing Algorithm for Network Reconfiguration in Large-scale Distribution Systems. *IEEE Transactions on Power Delivery*, 517(4):1070–1078, 2002.

[127] L. Jingpeng and R. Kwan. A Fuzzy Simulated Evolution Algorithm for the Driver Scheduling Problem. In *IEEE Congress on Evolutionary Computation*, pages 1115–1122, May 2001.

[128] L. Jingpeng and R. Kwan. A Fuzzy Evolutionary Approach with Taguchi Parameter Setting for the Set Covering Problem. In *IEEE Congress on Evolutionary Computation*, pages 1203–1208, May 2002.

[129] G. Jones, R. Brown, D. Clark, P. Willett, and R. Glen. *Searching Databases of Two-Dimensional and Three-Dimensional Chemical Structures using Genetic Algorithms*. S. Forrest (Ed.), 5th International Conference on Genetic Algorithms, pages 597-602, 1993.

[130] H. Jutler. Linear Model with Several Objective Functions. *Ekonomika i matematiceckije Metody (in Polish)*, 3:397–406, 1967.

[131] J. Kacprzyk. Group Decision Making with a Fuzzy Linguistic Majority. *Fuzzy Sets & Systems*, 11:105 – 118, 1986.

[132] J. Kacprzyk, M. Fedrizzi, and H. Nurmi. Group Decision Making and Consensus under Fuzzy Preferences and Fuzzy Majority. *Fuzzy Sets & Systems*, 49:21 – 31, 1992.

[133] C. Kahraman, D. Ruan, and I. Doan. Fuzzy Group Decision-making for Facility Location Selection. *Information Sciences*, 157:135–153, 2003.

[134] E. Karasan, O. Karasan, N. Akar, and M. Pinar. Mesh Topology Design in Overlay Virtual Private Networks. *Electronics Letters*, 38(16):939–941, 2002.

[135] H. Kawamura, M. Yamamoto, K. Suzuki, and A. Ohuchi. Multiple Ant Colonies Algorithm Based on Colony Level Interactions. *IEICE Transactions on Fundamentals*, E83-A(2):371–379, 1999.

[136] G.E. Keiser. *Local Area Networks*. McGraw-Hill, 1989.

[137] J. Kennedy. Small Worlds and Mega Minds: Effects of Neighborhood Topology on Particle Swarm Performance. In *IEEE Congress on Evolutionary Computation*, pages 1931–1938, 1999.

[138] J. Kennedy and R. Eberhart. Particle Swarm Optimization. In *IEEE International Joint Conference on Neural Networks*, pages 1942–1948, 1995.

[139] J. Kennedy and R. Eberhart. *The Particle Swarm: Social Adaptation in Information Processing Systems*. D. Corne, M. Dorigo, and F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, pages 379-387, 1999.

[140] J. Kennedy and R. C. Eberhart. Particle Swarm Optimization. In *IEEE International Joint Conference on Neural Networks, Vol. 4*, pages 1942–1948, 1995.

[141] J. Kennedy and R. Medes. Population Structures and Particle Swarm Performance. In *IEEE Congress on Evolutionary Computation*, pages 1671–1676, 2002.

[142] A. Kershenbaum. *Telecommunications Network Design Algorithms*. McGraw-Hill, 1993.

[143] S. Khajehpour and D. Grierson. Conceptual Design using Adaptive Computing. In *Genetic and Evolutionary Computation Conference*, pages 62–67, 2001.

[144] H. Kin, Y. Hayashi, and K. Nara. The Performance of Hybridized Algorithm of GA, SA, and TS for Thermal Unit Maintenance Scheduling. In *IEEE Congress on Evolutionary Computation*, pages 114–119, 1995.

[145] S. Kirkpatrick, C. Gelatt Jr., and M.Vecchi. Optimization by Simulated Annealing. *Science*, pages 498–516, May 1983.

[146] R. Kling and P. Banerjee. ESP: Placement by Simulated Evolution. *IEEE Transactions on Computer-Aided Design*, 8(3):245–256, 1989.

[147] R. Kling and P Banerjee. Optimization by Simulated Evolution with Applications to Standard Cell Placement. In *Proceedings of 27th Design Automation Conference*, pages 20–25, 1990.

[148] R. Kling and P. Banerjee. Empirical and Theoretical Studies of the Simulated Evolution Method Applied to Standard Cell Placement. *IEEE Transactions on Computer-Aided Design*, 10(10):1303–1315, October 1991.

[149] R. M. Kling. *Optimization by Simulated Evolution and its Application to Cell Placement*. Ph.D. Thesis, University of Illinois, Urbana, 1990.

[150] T. Koopmans and M. Beckmann. Assignment Problems and the Location of Economic Activities. *Econometrica*, 25:53–76, 1957.

[151] J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *American Matehmatical Society*, 7(1):48–50, 1956.

[152] A. Kumar, M. Pathak, and Y. Gupta. Genetic Algorithm-Based Reliability Optimization for Computer Network Expansion. *IEEE Transactions on Reliability*, 24:63–72, 1995.

[153] A. Kurapati and S. Azarm. Immune Network Simulation with Multiobjective Genetic Algorithms for Multidisciplinary Design Optimization. *Engineering Optimization*, 33:245–260, 2000.

[154] P. Laarhoven and E. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer Academic, Norwell, Massachusetts, 1987.

[155] H. Li and V. Yen. *Fuzzy Sets and Fuzzy Decision-Making*. CRC Press, USA, 1995.

[156] S. Li, Y. Yang, and C. Teng. Fuzzy Goal Programming With Multiple Priorities via Generalized Varying-Domain Optimization Method. *IEEE Transactions on Fuzzy Systems*, 12(5):596–604, 2004.

[157] W. Liao, Y. Chen, and S. Wang. Goal-attainment Method for Optimal Multiobjective Harmonic Filter Planning in Industrial Distribution Systems. *IEE Generation, Transmission and Distribution*, 49(5):557–563, 2002.

[158] M. Lim, S. Rahardja, and B. Gwee. A GA Paradigm for Learning Fuzzy Rules. *Fuzzy Sets & Systems*, 82:177–186, 1996.

[159] J. Liska and S. S. Melsheimer. Complete Design of Fuzzy Login System using Genetic Algorithms. In *3rd IEEE International Conference on Fuzzy Systems*, pages 1377–1382, 1994.

[160] D. Loughlin and S. Ranjithan. The Neighborhood Constraint Method: A Genetic Algorithm-Based Multiobjective Optimization Technique. In *7th International Conference on Genetic Algorithms*, pages 666–673, 1997.

[161] L.M. Gambardella M. Dorigo. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[162] V. Maniezzo. *Exact and Approximate Nondeterministic Tree-search Procedures for the Quadratic Assignment Problem.* Technical Report CSR 98-1, University of Bologna, Italy, 1998.

[163] V. Maniezzo and A. Colorni. The Ant System Applied to the Quadratic Assignment Problem. *IEEE Transactions on Knowledge and Data Engineering*, 11(5):769–778, 1999.

[164] V. Maniezzo, A. Colorni, and M. Dorigo. *The Ant System Applied to the Quadratic Assignment Problem.* Technical Report IRIDIA/94-28, Universite Libre de Bruxelles, Belgium, 1994.

[165] C. Mariano and E. Morales. *A Multiple Objective Ant-Q Algorithm for the Design of Water Distribution Irrigation Networks.* Technical Report HC-9904, Instituto Mexicano de Tecnologa del Agua, Mexico, 1999.

[166] Marimin, M. Umano, I. Hatono, and H. Tamura. Linguistic Labels for Expressing Fuzzy Preference Relations in Fuzzy Group Decision Making. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 28(2):205–218, 1998.

[167] I. Matsuba. Optimal Simulated Annealing Method and its Application to Combinatorial Problems. In *IEEE International Joint Conference on Neural Networks*, pages 541–546, 1989.

[168] P. R. McMullen. An Ant Colony Optimization Approach to Addressing a JIT Sequencing Problem with Multiple Objectives. *Artificial Intelligence in Engineering*, 15(3):309–317, 2001.

[169] J. M. Mendel. Fuzzy Logic Systems for Engineering: A Tutorial. *Proceedings of the IEEE*, 83(3):345–377, March 1995.

[170] D. Merkle, M. Middendorf, and H. Schmeck. Ant Colony Optimization for Resource-Constrained Project Scheduling. *IEEE Transactions on Evolutionary Computation*, 6(4):333 – 346, 2002.

[171] R. Michel and M. Middendorf. An Island Model based Ant System with Lookahead for the Shortest Supersequence Problem. In *5th International Conference on Parallel Problem Solving from Nature*, pages 692–701, 1998.

[172] R. Michel and M. Middendorf. *An ACO Algorithm for the Shortest Common Supersequence Problem.* D. Corne, M. Dorigo, and F. Glover (Eds.), New Methods in Optimization, McGraw-Hill, 1999.

[173] K. Miettinen. Some Methods for Nonlinear Multi-objective Optimization. In *IEEE/ACM 1st International Conference on Evolutionary Multi-Criterion Optimization, Lecture Notes in Computer Science, Vol. 1993, Springer*, pages 1 – 20, 2001.

[174] G. A. Miller. *The Organization of Lexical Memory.* The Pathology of Memory, G. A. Talland and N. C. Waugh (Eds.), New York Academic, 1969.

[175] M. Minhas and S. Sait. A Parallel Tabu Search Algorithm for Optimizing Multiobjective VLSI Placement. In *International Conference on Computational Science and its Applications*, pages 587–595, May 2005.

[176] T. Miyoshi, S. Shimizu, and Y. Tanaka. Fast Topological Design with Simulated Annealing for Multicast Networks. In *7th International Symposium on Computers and Communications*, pages 959–966, 2002.

[177] M. Mizumoto. Fuzzy Sets and Their Operations. *Information and Control*, 48:30–48, 1981.

[178] M. Mizumoto. Comparison of Various Fuzzy Reasoning Methods. In *2nd International Fuzzy Systems Association Congress*, pages 2–7, 1987.

[179] O. A. Mohammed and G. F. Uler. Genetic Algorithms for the Optimal Design of Electromagnetic Design. In *Conference on the Annual Review of Progress in Applied Computational Electromagnetics*, pages 386–393, 1995.

[180] D. C. Montgomery. *Design and Analysis of Experiments*. 3rd Ed., John Wiley & Sons, 1991.

[181] J. Moore and R. Chapman. *Application of Particle Swarm to Multiobjective Optimization*. Department of Computer Science and Software Engineering, Auburn University, 1999.

[182] M. Mostafa and S. Eid. A Genetic Algorithm for Joint Optimization of Capacity and Flow Assignment in Packet Switched Networks. In *17th National Radio Science Conference*, pages C51 – C56, 2000.

[183] S. Mostaghim and J. Teich. Strategies for Finding Good Local Guides in Multi-objective Particle Swarm Optimization (MOPSO). In *IEEE Swarm Intelligence Symposium*, pages 26–33, 2003.

[184] D. Mueller, H. Graeb, and U. Schlichtmann. Trade-Off Design of Analog Circuits using Goal Attainment and "Wave Front" Sequential Quadratic Programming. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6, 2007.

[185] T. Murata, H. Oshida, and M. Gen. Rule-based Weight Definition for Multi-objective Fuzzy Scheduling with the OWA Operator. In *26th Annual Conference of the Industrial Electronics Society*, pages 2756 – 2761, 2000.

[186] S. Nahar, S. Sahni, and E. Shragowitz. Experiments with Simulated Annealing. In *22nd Design Automation Conference*, pages 748–752, 1985.

[187] Y. Nakamichi and T. Arita. Diversity Control in Ant Colony Optimization. In *Inaugural Workshop on Artificial Life*, pages 70–78, 2001.

[188] J. Nash. The Bargaining Problem. *Econometrica*, 18:155–162, 1950.

[189] G. Nemhauser, A. Rinnooy Kan, and M. Todd (Eds.). *Optimization*. North-Holland, 1989.

[190] I. Oliver, D. Smith, and J. Holland. *A Study of Permutation Operators on the Traveling Salesman Problem*. J. J. Grefenstette (Ed.), Genetic Algorithms and Their Applications: Proceedings of the 2nd International Conference on Genetic Algorithms. New Jersey: Lawrence Erlbaum Associates, 1986.

[191] G. Oltean, C Miron, and E. Moccan. Multiobjective Optimization for Analog Circuits Design based on Fuzzy Logic. In *9th International Conference on Electronics, Circuits and Systems*, pages 777 – 780, 2002.

[192] B. Ombuki, M. Nakamura, Z. Nakao, and I. Onaga. Evolutionary Computation for Topological Optimization of 3-Connected Computer Networks. In *IEEE Conference on Systems, Man, and Cybernetics*, pages 659–664, 1999.

[193] M. Omran. *Particle Swarm Optimization Methods for Pattern Recognition and Image Processing*. PhD Thesis, University of Pretoria, 2005.

[194] I. H. Osman. Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem. *Annals of Operations Research*, 41:421–451, 1993.

[195] S. Palaniappan, S. Zein-Sabatto, and A. Sekmen. Dynamic Multiobjective Optimization of War Resource Allocation using Adaptive Genetic Algorithms. In *IEEE SoutheastCon*, pages 160 – 165, 2001.

[196] C. Papadimiriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

[197] G. T. Parks. Multiobjective Pressurized Water Reactor Reload Core Design by Nondominated Genetic Algorithm Search. *Nuclear Science and Engineering*, 124(1):178–187, 1996.

[198] K. Parsopoulos, D. Tasoulis, and M. Vrahatis. Multiobjective Optimization using Parallel Vector Evaluated Particle Swarm Optimization. In *IASTED In-*

ternational Conference on Artificial Intelligence and Applications, pages 823–828, 2004.

[199] K. Parsopoulos and M. Vrahatis. Particle Swarm Optimization Method in Multiobjective Problems. In *ACM Symposium on Applied Computing*, pages 603–607, 2002.

[200] K. Parsopoulos and M. Vrahatis. Recent Approaches to Global Optimization Problems through Particle Swarm Optimization. *Natural Computing*, 1:235–306, 2002.

[201] J. Pérez and J. Basterrechea. Comparison of Different Heuristic Optimization Methods for Near-Field Antenna Measurements. *IEEE Transactions on Antennas and Propagation*, 55(3):549 – 555, 2007.

[202] C. D. Perttunen. Nonparametric Cooling Schedules in Simulated Annealing using the Normal Score Transformations. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 609–612, 1991.

[203] S. Pierre and A. Elgibaoui. A Tabu Search Approach for Designing Computer Network Topologies with Unreliable Components. *IEEE Transactions on Reliability*, 46(3):350–359, 1997.

[204] S. Pierre and G. Legault. A Genetic Algorithm for Designing Distributed Computer Network Topologies. *IEEE Transactions on Systems, Man, Cybernetics*, 28(2):249–258, April 1998.

[205] B. Pollack-Johnson and M. Liberatore. Incorporating Quality Considerations into Project Time/Cost Tradeoff Analysis and Decision Making. *IEEE Transactions on Engineering Management*, 53(4):534–542, 2006.

[206] V. Prasad and W. Kuo. Reliability Optimization of Coherent Systems. *IEEE Transactions on Reliability*, 49(3):323–330, 2000.

[207] P. Premprayoon and P. Wardkein. Topological Communication Network Design using Ant Colony Optimization. In *The 7th International Conference on Advanced Communication Technology*, pages 1147 – 1151, 2005.

[208] R. C. Prim. Shortest Connection Networks and Some Generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[209] D. Quagliarella and A. Vicini. *Coupling Genetic Algorithms and Gradient Based Optimization Techniques*. D. Quagliarella, J. Périaux, C. Poloni, and G. Winter (Eds.), Genetic Algorithms and Evolution Strategies in Engineering and Computer Science. Recent Advances and Industrial Applications, West Sussex, England, John Wiley & Sons, Chapter 14, pages 289-309, 1997.

[210] T. Ray and K. M. Liew. A Swarm Metaphor for Multiobjective Design Optimization . *Engineering Optimization*, 34(2):141–153, 2002.

[211] P. M. Reed, B. S. Minsker, and D. E. Goldberg. A Multiobjective Approach to Cost Effective Long-term Groundwater Monitoring using an Elitist Nondominated Sorted Genetic Algorithm with Historical Data. *Journal of Hydroinformatics*, 3(2):71–89, 2001.

[212] W. Reeves. Particle Systems - A Technique for Modelling a Class of Fuzzy Objects. *ACM Transactions on Graphics*, 2(2):91–108, 1983.

[213] M. Reyes-Sierra and C. A. Coello-Coello. Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art. *International Journal of Computational Intelligence Research*, 2(3):287–308, 2006.

[214] J. H. Reynolds and E. D. Ford. Multicriteria Assessment of Ecological Process Models. *Ecology*, 80(5):538–553, 1999.

[215] Y. Saab and V. Rao. An Evolution Based Approach to Partitioning ASIC Systems. In *26th ACM/IEEE Design Automation Conference*, pages 767–770, 1989.

[216] Y. Saab and V. Rao. Stochastic Evolution: A Fast Effective Heuristic for Some Generic Layout Problems. In *27th ACM/IEEE Design Automation Conference*, pages 26–31, 1990.

[217] Y. Saab and V. Rao. Combinatorial Optimization by Stochastic Evolution. *IEEE Transactions on Computer Aided Design*, 10(4):525–535, April 1991.

[218] S. Sait, M. Ali, and A. Zaidi. Multiobjective VLSI Cell Placement using Distributed Simulated Evolution Algorithm. In *IEEE International Symposium on Circuits and Systems*, pages 6226–6229, May 2005.

[219] S. Sait, M. Faheemuddin, M. Minhas, and S. Sanaullah. Multiobjective VLSI Cell Placement Using Distributed Genetic Algorithm. In *Genetic and Evolutionary Computation Conference*, pages 1585 – 1586, 2005.

[220] S. Sait and H. Youssef. *Iterative Computer Algorithms and their Application to Engineering*. IEEE Computer Science Press, USA, Dec. 1999.

[221] R. Saravanan. *Manufacturing Optimization Through Intelligent Techniques*. CRC Press, USA, 2006.

[222] B. Sarif, M. Abd-El-Barr, S. Sait, and U. Al-Saiari. Fuzzified Ant Colony Optimization Algorithm for Efficient Combinational Circuits Synthesis. In *IEEE Congress on Evolutionary Computation*, pages 1317 – 1324, 2004.

[223] D. Savic. *Single-objective vs. Multiobjective Optimization for Integrated Decision Support*. Technical Report, University of Exeter, 2001.

[224] B. Schweizer and A. Sklar. Associative Functions and Abstract Semigroups. *Publicationes Mathematicae Debrecen*, 10:69–81, 1963.

[225] B. Secrest. *Travelling Salesman Problem for Surveillance Mission using Particle Swarm Optimization*. MS Thesis, Air Force Institute of Technology, USA, 2001.

[226] S. Shani and T. Gonzales. P-complete Approximation Problems. *Journal of ACM*, 23:555–565, 1976.

[227] X. Shi, Y. Liang, H. Lee, C. Lu, and Q. Wang. Particle Swarm Optimization-based Algorithms for TSP and Generalized TSP. *Information Processing Letters*, 103:169–176, 2007.

[228] Y. Shi and R. Eberhart. *Parameter Selection in Particle Swarm Optimization*. V. W. Porto, N. Saravanan, D. Waagen, and A. Eiben (Eds.), Evolutionary Programming VII, pp. 611-616. Springer, 1998.

[229] Y. Shou and B. Guo. A Lexicographic Approach for Selecting R & D Projects with Resource Constraints. In *IEEE International Engineering Management Conference*, pages 799–802, 2004.

[230] E. Shragowitz, J. Lee, and E. Kang. Application of Fuzzy Logic in Computer-aided VLSI Design. *IEEE Transactions on Fuzzy Systems*, 6(1):163 – 172, 1998.

[231] K. Singh and K. Deb. Comparison of Multi-Modal Optimization Algorithms Based on Evolutionary Algorithms. In *Genetic and Evolutionary Computation Conference*, pages 1305–1312, 2006.

[232] R. Soland. Multicriteria Optimization: A General Characterization of Efficient Solutions. *Decision Sciences*, 10:26–38, 1979.

[233] M. A. Sportack. *IP Routing Fundamentals*. Cisco Press, 1999.

[234] D. Srinivasan and T. H. Seow. Particle Swarm Inspired Evolutionary Algorithm (PS-EA) for Multiobjective Optimization Problems. In *IEEE Congress on Evolutionary Computation*, pages 2292–2297, 2003.

[235] T. Stützle and H. Hoos. The MAX-MIN Ant System and Local Search for the Travelling Salesman Problem. In *IEEE Congresss on Evolutionary Computation*, pages 309–314, 1997.

[236] T. Stützle and H. Hoos. *MAX-MIN Ant System and Local Search for Combinatorial Optimization Problems*. I. H. Osman, S. Voβ, S. Martello and C. Roucairol (Eds.), Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academics, pp. 137 - 154, 1998.

[237] R. Subrata and A. Zomaya. A Comparison of Three Artificial Life Techniques for Reporting Cell Planning in Mobile Computing. *IEEE Transactions on Parallel and Distributed Systems*, 14(2):142–153, 2003.

[238] P. Suganthan. Particle Swarm Optimizer with Neighborhood Optimizer. In *IEEE Congress on Evolutionary Computation*, pages 1958–1962, 1999.

[239] G. Sywerda. Uniform Crossover in Genetic Algorithms. In *3rd International Conference on Genetic Algorithms*, pages 2–9, 1989.

[240] X. Tan, W. Jin, and D. Zmo. The Application of Multi-Criterion Satisfactory Optimization. In *Computer Networks Design, Parallel and Distributed Computing, Applications and Technologies*, pages 660–664, 2003.

[241] D. Thompson and G. Bilbro. Comparison of a Genetic Algorithm with a Simulated Annealing Algorithm for the Design of an ATM Network. *IEEE Communications Letters*, 4(8):267–269, 2000.

[242] V. Torra. Weighted OWA Operators for Synthesis of Information. In *5th IEEE International Conference on Fuzzy Systems*, pages 966 – 971, 1996.

[243] F. van den Bergh. Particle Swarm Weight Initialization in Multi-layer Perceptron Artificial Neural Networks. In *Development and Practice of Artificial Intelligence Techniques*, pages 41–45, 1999.

[244] F. van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD Thesis, University of Pretoria, 2001.

[245] F. van den Bergh and A. P. Engelbrecht. Cooperative Learning in Neural Networks using Particle Swarm Optimizers. *South African Computer Journal*, 26:84–90, 2000.

[246] F. van den Bergh and A. P. Engelbrecht. Training Product Unit Networks using Cooperative Particle Swarm Optimizers. In *IEEE International Joint Conference on Neural Networks*, pages 126–132, 2001.

[247] F. van den Bergh and A. P. Engelbrecht. A New Locally Convergent Particle Swarm Optimizer. In *IEEE Conference on Systems, Man, and Cybernetics*, pages 96–101, 2002.

[248] S. Varadarajan, N. Ramakrishna, and M. Bayoumi. A Stochastic Evolution based Register Allocation using Multiport Memories. In *36th Midwest Symposium on Circuits and Systems*, pages 472 – 475, 1993.

[249] M. Vazquez and L. D. Whitley. A Hybrid Genetic Algorithm for the Quadratic Assignment Problem. In *Genetic and Evolutionary Computation Conference*, pages 169–178, 2000.

[250] I. Venanzi and A. Materazzi. Multi-objective Optimization of Wind-excited Structures. *Engineering Structures*, 29(6):983–990, 2006.

[251] K. Wang, L. Huang, C. Zhou, and W. Pang. Particle Swarm Optimization for Travelling Salesman Problem. In *International Conference on Machine Learning and Cybernetics*, pages 1583 – 1585, 2003.

[252] S. Weber. A General Concept of Fuzzy Connectives, Negations and Implications Based on t-Norms and t-Conorms. *Fuzzy Sets & Systems*, 11:115–134, 1983.

[253] H. Weiss. *Genetic Algorithm and Optimum Robot Design.* Technical Report, Institute of Robotics and Mechatronics, http://www.robotic.dlr.de/ Holger.Weiss/garep/node3.html (Accessed on March 23, 2005), 2003.

[254] A. White, J. Mann, and G. Smith. Genetic Algorithms and Network Ring Design. *Annals of Operations Research*, 6(1):347–371, 1999.

[255] P. Wilson and M. Macleod. Low Implementation Cost IIR Digital Filter Design using Genetic Algorithms. In *IEE/IEEE Workshop on Natural Algorithms in Signal Processing*, pages 1 – 8, 1993.

[256] A. Wright. *Genetic Algorithms for Real Parameter Optimization.* G.J Rawlins (Ed.), Foundations of Genetic Algorithms I, Morgan Kaufmann, San Mateo, pages 205-218, 1991.

[257] J. Wright and H. Loosemore. An Infeasibility Objective for Use in Constrained Pareto Optimization. In *IEEE/ACM 1st International Conference on Evolutionary Multi-Criterion Optimization, Lecture Notes in Computer Science, Vol. 1993, Springer*, pages 256–268, 2001.

[258] Z. Xu, Y. Li, and X. Feng. Constrained Multi-objective Task Assignment for UUVs using Multiple Ant Colonies System. In *ISECS International Colloquium on Computing, Communication, Control, and Management*, pages 462–466, 2008.

[259] R. Yager. Multiple Objective Decision-making using Fuzzy Sets. *International Journal of Man-Machine Studies*, 9:375–382, 1977.

[260] R. Yager. On Ordered Weighted Averaging Aggregation Operators in Multicriteria Decision-making. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):183–190, Jan 1988.

[261] R. Yager. Second Order Structures in Multi-criteria Decision Making. *International Journal of Man-Machine Studies*, 36:553–570, 1992.

[262] R. Yager. Criteria Importances in OWA Aggregation: An Application of Fuzzy Modelling. In *6th IEEE International Conference on Fuzzy Systems*, pages 1677 – 1682, 1997.

[263] F. Yang, M. Ling-he, Z. Lan, and C. Jia-lin. The Application of Pareto Ant Colony Algorithm in Multi-Objective Power Network Planning. In *IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, pages 794–798, 2008.

[264] H. Yin-Tsung and H. Jer-Sho. Simulated Evolution Based Code Generation for Programmable DSP Processors. In *IEEE Symposium on Circuits and Systems*, pages 2593 – 2596, 1997.

[265] H. Yoshida, K. Kawata, S. Fukuyama, Y. Takayama, and Y. Nakanishi. A Particle Swarm Optimization for Reactive Power and Voltage Control considering Voltage Security Assessment. *IEEE Transactions on Power Systems*, 15(4):1232–1239, 2000.

[266] H. Yoshida, K. Kawata, Y. Fukuyama, and Y. Nakanishi. A Particle Swarm Optimization for Reactive Power and Voltage Control considering Voltage Stability. In *IEEE International Conference on Intelligent System Applications to Power Systems*, pages 117–121, 1999.

[267] H. Youssef, S. Sait, and O. Issa. Computer-Aided Design of Structured Backbones. In *15th National Computer Conference and Exhibition*, pages 1–18, 1997.

[268] H. Youssef, S. Sait, and S. Khan. Fuzzy Simulated Evolution Algorithm for Topology Design of Campus Networks. In *IEEE Congress on Evolutionary Computation*, pages 180–187, 2000.

[269] H. Youssef, S. Sait, and S. Khan. An Evolutionary Algorithm for Network Topology Design. In *IEEE International Joint Conference on Neural Networks*, pages 744–749, 2001.

[270] H. Youssef, S. Sait, and S. Khan. Fuzzy Evolutionary Hybrid Metaheuristic for Network Topology Design. In *IEEE/ACM 1st International Conference on Evolutionary Multi-Criterion Optimization, Lecture Notes in Computer Science, Vol. 1993, Springer*, pages 400–415, 2001.

[271] H. Youssef, S. Sait, and S. Khan. Topology Design of Switched Enterprize Networks using a Fuzzy Simulated Evolution Algorithm. *Engineering Applications of Artificial Intelligence*, 15:327–340, 2002.

[272] H. Youssef, S. Sait, and S. Khan. A Fuzzy Evolutionary Algorithm for Topology Design of Campus Networks. *Arabian Journal for Science and Engineering*, 29(2b):195–212, 2004.

[273] Y. Yu. Multiobjective Decision Theory for Computational Optimization in Radiation Therapy. *Medical Physics*, 24:1445–1454, 1997.

[274] P. Yuan, C. Ji, Y. Zhang, and Y. Wang. Optimal Multicast Routing in Wireless Ad hoc Sensor Networks. In *IEEE International Conference on Networking, Sensing, and Control*, pages 367 – 371, 2004.

[275] L. Zadeh. Optimality and Non-Scalar-Valued Performance Criteria. *IEEE Transactions on Automatic Control*, 8:59 – 60, 1963.

[276] L. A. Zadeh. Fuzzy Sets. *Information Control*, 8:338–353, 1965.

[277] L. A. Zadeh. The Concept of a Linguistic Variable and its Application to Approximate Reasoning. *Information Sciences*, 8:199–249, 1975.

[278] H. J. Zimmerman. *Fuzzy Set Theory and its Applications*. Kluwer Academic Publishers, third edition, 1996.

Appendix A
Nomenclature

This appendix provides a list of symbols used in this thesis.

A	a fuzzy set.
α_{goal}	scalar variable in goal attainment method.
α_{SA}	cooling rate in simulated annealing.
α_{ant}	constant in ACO algorithm.
B_{ij}	delay per bit due to the network device feeding the link connecting LANs i and j, equal to $b_{i,j}/\omega$.
b_i	goal associated with and objective in goal attainment method.
b_{ij}	delay per packet.
β	variable in the Ordered Weighted Average operator.
β^e	variable in the OWA operator for link evaluation function in SimE.
β_{SA}	annealing constant.
β_{ant}	constant in ACO algorithm.
C_i	current cost of individual i in SimE.

c_1, c_2	acceleration coefficients in PSO.
d	total number of networking devices in the network, where nodes are connected to networking devices.
D_{nd}	delay due to network devices.
η	heuristic value in ACO.
ε_j	upper bounds in ε-constraint method.
G	syntactic rule which generates the terms in $T(\Omega)$.
g_i	goodness of individual i in SimE.
$g_m(x)$	set of inequality constraints.
γ	overall external traffic in bps.
γ_{ij}	external traffic in bps between nodes i and j.
$h_m(x)$	set of equality constraints.
L	number of links of the proposed tree topology.
λ_i	traffic in bits per second (bps) on link i.
$\lambda_{max,i}$	capacity in bps of link i.
M	markov chain in simulated annealing.
μ	membership function (overall goodness) of a solution in fuzzy logic.
N	semantic rule which associates with linguistic value its meaning.
N_i	neighborhood in l_{best} model of PSO.
n	number of nodes (i.e. LANs).
ν	variable in the Unified And-Or operator.
O_i	optimal cost of individual i in SimE.
Ω	name of linguistic variable in fuzzy logic.
ω	average packet size in bits.

p_i	maximum number of nodes that can be connected to node i.
$p_{\iota\psi}^k$	probability of moving from state ι to state ψ.
$\Re$	set of all real numbers.
R_s	reliability of the network.
R_i	reliability of a link i.
ϱ	evaporation/forgetting constant in ACO.
S	feasible region.
s	number of particles used in PSO.
T	$n \times n$ topology matrix where, $t_{ij} = 1$ if LANs i and j are connected and $t_{ij} = 0$ otherwise.
T_i	target level for objective function in goal programming method.
$T(\Omega)$	term set of Ω in fuzzy logic.
τ_i	pheromone trail on edge i in ACO.
V_{max}	velocity clamping.
$\mathbf{v_i}$	the current velocity of the particle.
W_i	weight associated with an objective function in weighted sum method.
w	inertia weight in PSO.
X	universe of discourse.
$\mathbf{x_i}$	the current position of the particle.
$\mathbf{y_i}$	the personal best position of the particle.
$\hat{\mathbf{y}}_\mathbf{i}$	the neighborhood best position of the particle.

Appendix B
Linear Regression Analysis

In many problems there are two or more variables that are related, and it is important to model and explore this relationship. Regression analysis is frequently used in this type of situation. In regression analysis data is analyzed from both designed and undesigned experiments. In *simple regression analysis*, the relationship between a single regressor variable x and a response variable y needs to be determined. The regressor variable x is usually assumed to be a variable controllable by the experimenter. When the experiment is designed, the experimenter chooses that values of x and observes the corresponding value of y [180]. The expected value of y for each value of x is given by the following mathematical model:

$$\mathrm{E(y|x)} = \vartheta_0 + \vartheta_1 x + \phi$$

where ϑ_0 is the *intercept*, ϑ_1 is called the *regression coefficient* associated with variable x, and ϕ is a random error with mean zero and variance σ^2. In the above equation, ϑ_1 is of special interest. It signifies the rate at which y changes if x is varied. A high value of ϑ_1 will cause y to change at a faster rate when x is varied, while a low ϑ_1 will have a slow effect on y when x is varied.

In the experiments conducted in this thesis, the following model was used:

$$\mathrm{E}(\text{Objective}) = \vartheta_0 + \vartheta_1(nodes) + \phi$$

Here, the regressor variable is the number of nodes, while the response variable is the design objective (e.g. cost, delay, hops, reliability etc.) Using the above model, many regression equations were developed to study the effect of change of number of nodes on cost, delay, hops, and reliability for the OWA and UAO operators.

Appendix C
Derived Publications

This appendix provides a list of publications that have been derived from work presented in this thesis. These publications have been published, currently being reviewed, or yet to be submitted.

Journal Publications

1. Salman A. Khan and Andries P. Engelbrecht, "A New Fuzzy Operator and its Application to Topology Design of Distributed Local Area Networks", International Journal of Information Sciences, Elsevier, Vol 177, no. 13, July 2007, pp. 2692 - 2711.

2. Salman A. Khan and Andries P. Engelbrecht, "Multi-objective Hybrid Simulated Annealing Algorithms for Topology Design of Switched Local Area Networks", Soft Computing Journal, Springer, Vol 13, no. 1, January 2009, pp. 45 - 61.

3. Salman A. Khan and Andries P. Engelbrecht, "Design and Analysis of Multi-objective Iterative Heuristics for Distributed Local Area Network Topology

Design", Under review, IEEE Transactions on Fuzzy Systems.

4. Salman A. Khan and Andries P. Engelbrecht, "Fuzzy Multi-objective Swarm Intelligence Algorithms for Distributed Local Area Network Topology Design", To be submitted, Swarm Intelligence Journal.

5. Salman A. Khan and Andries P. Engelbrecht, "Fuzzy Hybrid Stochastic Evolution and Simulated Evolution Algorithms for Topology Design of Switched Local Area Networks", To be submitted.

6. Salman A. Khan and Andries P. Engelbrecht, "Particle Swarm Optimization Approach to Multi-objective Topology Design of Switched Local Area Networks", To be submitted.

7. Salman A. Khan and Andries P. Engelbrecht, "A Comparison of Evolutionary and Swarm Intelligence Techniques for Multi-objective Topology Design of Switched Local Area Networks", To be submitted.

8. Salman A. Khan and Andries P. Engelbrecht, "A Comparison of Ordered Weighted Averaging and Unified And-Or Operators - Application to Evolutionary and Swarm Intelligence Techniques for Multi-objective Topology Design of Local Area Networks", To be submitted.

Conference Publications

1. Salman A. Khan and Andries P. Engelbrecht, "A fuzzy ant colony optimization algorithm for topology design of distributed local area networks ", In Proceedings of the IEEE Swarm Intelligence Symposium, 2008.

2. Salman A. Khan and Andries P. Engelbrecht, "Application of Ordered Weighted Averaging and Unified And-Or Operators to Multi-objective Particle Swarm Optimization Algorithm", Accepted in 6th IEEE International Conference on Fuzzy Systems and Knowledge Discovery, August 14-16, 2009

3. Salman A. Khan and Andries P. Engelbrecht, "Dynamic Assignment of Parameters in Iterative Heuristics - a case study of distributed local area network topology design.