Engineering Nature-Inspired Heuristics for the Open Shortest Path First Weight Setting Problem

by

Mohammed Aijaz Mohiuddin
(Student # 11365162)

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

April 2018

Engineering Nature-Inspired Heuristics for the Open Shortest Path First Weight Setting Problem

by
Mohammed Aijaz Mohiuddin

Abstract

Optimizing network performance is a challenging task. Network links may become under-utilized or over-utilized. It is important to find routes that will ensure a good balance between utilization and congestion. One approach to find the routes in the network is the open shortest path first (OSPF) routing protocol. The open shortest path first weight setting problem (OSPFWS) is an approach to find the best routes by finding a set of numerical weights that will achieve optimal routing. The OSPFWS problem, an NP-hard problem, is a multi-objective optimization problem, with the following conflicting objectives: to minimize the maximum utilization, to minimize the number of congested links, and to minimize the number of unused links. In order to solve this multi-objective optimization problem, a fuzzy operator is used to aggregate the sub-objectives into one objective function. A number of nature-inspired iterative heuristic algorithms are adapted in this thesis to solve the OSPFWS problem. These algorithms are simulated annealing (SA), simulated evolution (SimE), and particle swarm optimization (PSO). Moreover, a hybrid version of PSO, namely, fuzzy evolutionary PSO (FEPSO), is proposed and studied. A sensitivity analysis of the control parameters of these algorithms is presented. The proposed algorithms are

compared with some well-established multi-objective optimization algorithms such as the non-dominating sorting genetic algorithm (NSGA-II), weighted aggregation particle swarm optimization (WAPSO), and Pareto-dominance particle swarm optimization (PDPSO).

Keywords: Optimization, Open shortest path first routing, Fuzzy logic, Simulated annealing, Simulated evolution, Swarm intelligence, Particle swarm optimization, Multi-objective optimization.

Thesis Supervisor: Prof. Andries P. Engelbrecht
Thesis Co-supervisor: Dr. Salman A. Khan
Department of Computer Science
Degree: Doctor of Philosophy

Acknowledgements

All praise be to God Almighty, for his limitless blessing and guidance. He is the one, who made this thesis possible by His will and mercy.

I would like to sincerely thank my supervisor, Professor Andries P. Engelbrecht and co-supervisor, Dr Salman A. Khan for their help, support, and encouragement throughout this research endeavor. I commend and acknowledge them for their valuable time, consistent support and stimulating discussions. Every step working with them towards the completion of this thesis was fascinating, enjoyable and insightful.

I was motivated and supported fully by my beloved parents. My wife deserves a special mention for her dedication, patience and understanding. I also thank my colleagues at my workplace for their support.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

The Internet has become an integral part of our lives. Internet users demand uninterrupted and smooth functioning of various applications. One important element of this smooth functioning is to ensure that data loss should be minimized, if not totally eliminated. This demands utilization of network resources such as network bandwidth and the number of available links at their best, or rather, at *optimal* levels. The challenge lies in how such optimal levels can be achieved. For example, if the flow of data is optimally mapped or distributed onto the available network resources, then the data packets will not be dropped, thus achieving optimal performance with respect to data flow. This concern about optimal utilization of network resources is the primary focus of the ideas and methods presented in this work.

Optimization is a significant topic of research in various research domains including

science, medicine, engineering, business, and humanities. In many of these disciplines, optimization simply refers to "doing better". However, in the context of this thesis, optimization refers to a process of providing the best possible (optimal) solution to an NP-hard problem under the constraint of a limited computational budget.

The specific problem considered herein is known as the *open shortest path first weight setting* (OSPFWS) problem, which is classified as an NP-hard problem [47]. The OSPFWS problem is related to the open shortest path first (OSPF) routing protocol [119], which is widely used in Internet Protocol (IP) networks. In this thesis, the terms "network" and "graph" are used interchangeably and these terms represent a communication network. The term "link" is used under the context of the term "network" and the term "arc" or "edge" is used under the context of the term "graph". In the context of the OSPFWS problem, a node represents a network router and an edge represents a network link. The focus of this thesis is on adapting and applying a number of optimization techniques and their variants to the OSPFWS optimization problem, modelled as a multi-objective optimization problem. These sub-objectives have to be simultaneously optimized, namely to minimize "maximum utilization", to minimize the number of congested links, and to minimize the number of unused links. In the context of the OSPFWS problem, the ratio of load on the link to the capacity of the link is referred to as the "utilization" of that link. The maximum utilization value amongst all the network links is termed as "maximum utilization". Note that

these objectives are in conflict with one another, because minimizing one objective may increase or decrease the other two objectives and vice versa. An optimal solution is thus required, which provides a balance among the objectives.

This thesis applies various optimization techniques, including SA [91], SimE [93], and PSO [78] to solve the multi-objective OSPFWS problem.

This chapter provides the motivation of the thesis followed by objectives and methodology adopted in the thesis. The contributions of this thesis are then summarized. The organization of the thesis is discussed in the last section of this chapter.

1.1 Motivation

In practice, network administrators often aim to use network resources efficiently. Routing is an important factor in enhancing the efficiency of a network. OSPF routing is widely used by large organizations, both as an open standard and a mature protocol. It is also supported by most vendors of routing hardware and software. The OSPF routing protocol assigns a weight to each link, where a weight represents a numeric integer value. The weights are then used to find the shortest paths for each destination node from other nodes. The OSPF routing protocol does not take the capacities of the links into consideration when these shortest paths are determined. Consequently, some links are left unused while others become congested depending on traffic demands. To balance the distribution of traffic demands onto the existing

network links is therefore a challenge.

In its simplest form, the OSPFWS problem can be described as finding a weight setting that leads to an efficient distribution of traffic demands onto the network. There are different mechanisms that guide the setting of these weights. Cisco sets the default weights of links as inversely proportional to the links' capacities [45]. However, Cisco does not take traffic demands into consideration, which may result in congested or unutilized links. Thus, Cisco's policy of assigning weights may not always be optimum. Another possibility is to assign the unity weight to all the links. However, this will lead to inefficient utilization of the links. This inefficiency will happen because links with less capacity might be used frequently to obtain the shortest paths between source and destination nodes. Contrary to this, higher capacity links might not be selected for shortest paths and thus might not be used at all.

As the size of a network grows (which depends on the number of nodes and links), the solution space for the OSPFWS problem grows exponentially. Under such circumstances, intelligent heuristics are good candidates to be explored for better performance. The reason for being good candidates is that the intelligent heuristics explore the promising regions of the search space, instead of the whole search space. Literature has shown that well-established heuristics, such as SA, SimE, and PSO, have been successfully applied to several NP-hard problems similar to the OSPFWS problem [6, 71, 102, 104, 106, 109, 148, 157, 163, 164, 167, 170]. In addition, research

has revealed that hybridization of heuristics has generally shown to be more efficient and effective [90, 129, 161, 168, 176, 177] on other problems. This particular aspect provides the motivation to develop hybrid heuristics for the OSPFWS problem. The motivation for the research work undertaken in this study therefore stems from the complexity of the problem, utilizing nature-inspired algorithms to optimize the OSPF weights, and designing hybrid versions of the algorithms specifically to solve the multi-objective OSPFWS problem.

The OSPFWS problem is a multi-objective problem, where the objectives are in conflict with one another. Minimizing one objective may increase or decrease the other two objectives and vice versa. A great deal of research has been dedicated to address issues related to multi-objective optimization [28, 69, 87, 117, 148]. Amongst many other approaches [18, 22, 27, 54, 55, 63, 74, 113, 122, 179], fuzzy logic [180, 181] has been used as an approach to solve multi-objective optimization problems. Therefore, the motivation also arises to utilize fuzzy logic in the above algorithms to address the conflicting multi-objective nature of the OSPFWS problem.

1.2 Objectives

The primary objectives of this thesis are summarized as follows:

1. To model the OSPFWS problem as a multi-objective optimization problem

using fuzzy logic operators.

2. To formulate the fuzzy logic based cost function for the multi-objective OSPFWS problem.

3. To propose and analyze a multi-objective fuzzy simulated annealing (FSA) algorithm for the OSPFWS problem, and to study the performance of the algorithm.

4. To propose and analyze a multi-objective fuzzy simulated evolution (FSimE) algorithm for the OSPFWS problem, and to study the performance of the algorithm.

5. To propose a multi-objective fuzzy particle swarm optimization (FPSO) algorithm and its hybrid variants for the OSPFWS problem, and to study the performance of these algorithms. Furthermore, a sub-objective of the study is to compare the proposed PSO algorithms with two well-known PSO variants, namely WAPSO [133] and PDPSO [2]. This is to compare the proposed fuzzy logic approach with another single-objective function approach (weighted-aggregation) and with a Pareto-dominance approach.

6. To perform mutual comparisons of the proposed algorithms in order to determine which performs best, as well as to compare these algorithms with a well-established multi-objective optimization algorithm, such as NSGA-II.

1.3 Methodology

The following methodology is adopted for this research work:

1. A detailed review of the OSPFWS problem is done to understand the complexity of the problem. Limitations of the existing approaches to solve the OSPFWS problem are also reviewed.

2. A literature review of related work with respect to the OSPFWS problem is done.

3. The OSPFWS problem is formulated as a multi-objective problem using fuzzy logic.

4. Each proposed algorithm is described in detail, its control parameters optimized, and its efficiency in solving the OSPFWS problem is empirically analyzed.

5. Archiving of non-dominated solutions is performed for all the implemented algorithms in order to obtain a set of non-dominated solutions that balances the trade-off amongst the objectives of the OSPFWS problem. Multi-objective optimization (MOO) performance measures, namely overall non-dominated vector generation (ONVG), spacing and hypervolume [38, 68, 152] are used to compare the performance of the implemented algorithms.

6. All the implemented algorithms are evaluated on a number of test cases, and

their performances are compared amongst one another and to other start-of-the-art multi-objective algorithms.

7. Following the standard practice for evaluating the performance of any non-deterministic algorithm, the results are statistically validated through non-parametric testing using the Wilcoxon ranked-sum test [65].

1.4 Contributions

As discussed above, Internet traffic load balancing refers to the distribution of incoming network traffic across the available network links. This thesis studies the problem of obtaining optimal load balancing in a network when the OSPF routing protocol is used. Earlier work on the OSPFWS problem has considered only a single objective. This thesis modelled the OSPFWS problem as a multi-objective problem, and develops a number of heuristics to solve this multi-objective problem. Accordingly, the key contributions of this thesis are a more realistic model of the OSPFWS problem, as well as adaptation and improvement of algorithms to solve the problem. The specific contributions in terms of artifacts are enumerated as follows:

1. The OSPFWS problem is formulated as a multi-objective optimization problem consisting of three objectives. These objectives are to minimize maximum utilization, to minimize the number of congested links, and to minimize the number

of unused links. The aforementioned formulation has not been attempted in any previous studies.

2. Fuzzy logic is used to develop the cost function for the multi-objective OSPFWS problem.

3. SA, SimE, and PSO algorithms are developed to solve the multi-objective OSPFWS problem. In addition, Pareto-dominance PSO, weighted sum PSO and NSGA-II are used to compare the performance of the new algorithms to existing approaches.

4. A hybrid variant of PSO incorporating the characteristics of the SimE algorithm is developed.

5. The performance of all the implemented algorithms is empirically analyzed and mutually compared to assess which algorithm has the best performance amongst the tested algorithms.

The following are the contributions to the field of multi-objective optimization:

1. Developing an objective function involving the third objective, namely the number of unused links in the network besides the existing two objectives, namely maximum utilization and the number of congested links for a communication network problem.

2. Developing a set-based fuzzy hybridized evolutionary PSO algorithm, called FEPSO.

3. A first application and study of FEPSO on the OSPFWS problem.

4. Comparing the performance of FEPSO with six other algorithms, namely FSA, FSimE, FPSO, WAPSO, PDPSO and NSGA-II to determine which approach performs best.

5. The proposed algorithms and methodology can be applied on any traffic load balancing problem modelled as graphs.

1.5 Organization of Thesis

Chapter 2 provides a general overview of optimization methods. The chapter starts with a short discussion of optimization. This is followed by a more focused discussion of multi-objective optimization and multi-objective aggregation techniques. Another focus of this chapter is the background on fuzzy logic, relating to its use in multi-objective optimization. Some well-known fuzzy operators are discussed, with an emphasis on the unified and-or fuzzy operator which has been utilized in this thesis. This is followed by a discussion of the iterative optimization algorithms used in this thesis. In this context, detailed discussions of SimE, SA, and PSO are provided.

Chapter 3 provides a discussion on routing in general, followed by the discussion of two well-known routing protocols, namely the routing information protocol (RIP) and the open shortest path first protocol (OSPF). The chapter also provides a comprehensive literature review on the OSPFWS problem.

Chapter 4 reviews the multi-objective OSPFWS problem addressed in this thesis in sufficient detail. It includes a formal description of the problem, notation, assumptions, objective functions, and test cases used.

Chapter 5 provides the details of the multi-objective fuzzy SA algorithm developed for the OSPFWS problem. The proposed fuzzy logic based objective function is incorporated into the fuzzy SA algorithm. A detailed sensitivity analysis of the control parameters is done and the performance of the algorithm is evaluated. An empirical analysis and comparison with an existing objective function proposed by Sqalli *et al.* [149, 160] is done to evaluate the effectiveness of the fuzzy objective function in the context of the SA algorithm.

Chapter 6 follows an approach similar to that adopted in Chapter 5, but with respect to the SimE algorithm. The chapter first provides details of the implementation of the multi-objective fuzzy SimE algorithm for the OSPFWS problem, using the fuzzy logic based objective function. A sensitivity analysis of the *bias* parameter, which is the only control parameter of the SimE algorithm, is performed. Furthermore, the performance of the proposed fuzzy SimE algorithm is evaluated through

empirical analysis and comparison with the objective function proposed by Sqalli *et al.* [149, 160].

Chapter 7 discusses the implementation of five algorithms namely FPSO, FEPSO, WAPSO, PDPSO and NSGA-II for the OSPFWS problem. For each algorithm, control parameters are also tuned in this chapter.

Chapter 8 summarizes and compares the results obtained in Chapters 5 to 7. A comprehensive comparison of all the proposed algorithms is presented. The focus of this chapter is to determine which of the proposed algorithms performs the best.

Chapter 9 highlights the conclusions of this thesis and provides directions for future research.

The appendices provide lists of symbols and terminology used in this thesis, as well as 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 and concepts used in the thesis. The chapter covers both single-objective and multi-objective optimization, with emphasis on the latter. Since fuzzy logic is used to develop the multi-objective function, the necessary concepts of fuzzy logic are reviewed. The chapter also includes the details of iterative heuristics, with a focus on those used in this thesis. It ends with the discussion of MOO performance measures.

The outline of the chapter is as follows: Section 2.1 provides a brief discussion of optimization. This section also discusses scalarized and non-dominance based approaches for solving MOO problems. Fuzzy logic is discussed in Section 2.2. Op-

timization algorithms used in this thesis are discussed in Section 2.3. Section 2.4 discusses MOO performance measures used in this thesis to evaluate the performance of MOO algorithms.

2.1 Optimization

Optimization is the process of modifying a system such that some features of the system work more efficiently or use fewer resources. The main idea of optimization is to determine a solution that represents the values of a set of parameters in such a way that the objective function is maximized or minimized, subject to certain constraints [7]. A solution is a *feasible solution* if it satisfies all the design constraints. Thus the optimal solution is the best solution amongst all available feasible solutions.

Many real-world problems are optimization problems. Disciplines such as engineering, science, medicine, and business employ optimization algorithms on problems such as structural design, resource allocation, and planning.

With respect to constraints, there are three kinds of optimization problems: *unconstrained*, *boundary constrained* and *constrained*. Unconstrained problems do not have any type of predefined conditions or restrictions on the values that can be assigned to variables of the problem. Boundary constrained problems are also unconstrained problems with an exception. The values that can be assigned to decision variables are constrained to be in a certain range. Contrary to this, constrained

problems have one or more predefined conditions on the values that can be assigned to variables of the problem. The aim for unconstrained, boundary constrained and constrained problems, is to minimize or maximize an objective function. A formal definition of unconstrained maximization is given below [124]:

$$\begin{aligned} &\text{Given} \quad f : \mathbb{R}^N \quad \rightarrow \quad \mathbb{R} \\ &\text{Find} \quad \mathbf{x}^* \in \mathbb{R} \quad \text{for which} \quad f(\mathbf{x}^*) \geq f(\mathbf{x}) \end{aligned} \tag{2.1}$$

The vector $\mathbf{x}^*$ in Equation (2.1) is referred to as a *global maximizer*. The global maximum value of f is given by $f(\mathbf{x}^*)$. The maximum value within the complete search space is referred to as the *global maximum*, while the maximum value within a small region of search space is referred to as a *local maximum*. One characteristic that is used to classify optimization problems is the modality of the search space. A unimodal problem has only one optimum, while a multi-modal problem has many optima.

Constrained optimization problems have certain predefined conditions to be satisfied while optimizing the objective function. A solution that does not satisfy all of the conditions or constraints is referred to as an *infeasible solution*. A formal definition of a constrained maximization problem is given below:

$$
\begin{aligned}
&\text{Given } \; f : \mathbb{R}^N \;\; \rightarrow \;\; \mathbb{R} \\
&\text{Find } \mathbf{x}^* \in \mathbb{R} \text{ for which } f(\mathbf{x}^*) \geq f(\mathbf{x}) \; \forall \; \mathbf{x} \in \mathbb{R}^{'} \subseteq \mathbb{R}^n \\
&\text{Subject to } g_m(\mathbf{x}) \leq 0, \quad m = 1, \cdots, n_g \\
&\qquad\qquad h_m(\mathbf{x}) = 0, \quad m = n_g + 1, \cdots, n_g + n_h
\end{aligned}
\tag{2.2}
$$

where g_m and h_m are the inequality and equality constraints, and n_g and n_h are respectively the number of inequality and equality constraints.

A next level of categorization of optimization problems are based on the number of objectives. If only one objective is to be optimized, then the problem is referred to as a *single objective optimization* (SOO) problem. Contrary to this, a multi-objective optimization problem has more than one objective. In some cases of multiple objectives, optimizing one objective automatically optimizes the others. Such problems are considered as SOO problems. Contrary to this, for some problems optimizing one objective degrades at least one other objective. MOO algorithms have been developed to solve optimization problems with such conflicting objectives. A MOO algorithm has to find solutions that provide a trade-off between conflicting objectives. The set of solutions that achieve such a balance is referred to as the *Pareto front* [151].

Mathematically, a MOO problem is defined as follows:

$$
\begin{aligned}
&\text{Optimize} : \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \cdots, f_{n_o}(\mathbf{x})) \\
&\text{Subject to } g_m(\mathbf{x}) \leq 0, \quad m = 1, \cdots, n_g \\
&\qquad\qquad h_m(\mathbf{x}) = 0, \quad m = n_g + 1, \cdots, n_g + n_h \\
&\qquad\qquad \mathbf{x} \in [x_{min}, x_{max}]^{n_x}
\end{aligned}
\tag{2.3}
$$

where n_o is the number of objectives and n_x is the number of decision variables. The boundary constraints are represented by $\mathbf{x} \in [x_{min}, x_{max}]^{n_x}$. There are at least two conflicting objective functions (i.e. $n_o \geq 2$). Here, $\mathbf{x} = (x_1, x_2, ..., x_{n_x})$ is called the vector of decision variables. In multi-objective optimization, vectors are regarded as optimal if their components can not be improved without deterioration of any one of the other components [113]. This is referred to as *Pareto optimality*. The output of a multi-objective optimization algorithm is a set of optimal solutions known as *Pareto-optimal solutions*. Pareto-optimal solutions are also known 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.

One way to solve a multi-objective optimization problem is to scalarize multiple objectives into a single-objective function and then to optimize this single-objective function. Thus each candidate solution to the problem during the search will be evaluated using this scalarized single-objective function. The second way is to handle individual objectives separately and judge the candidate solution using the domi-

nance relation. The following sub-sections discuss a number of popular approaches to scalarize a MOO problem. The section also discusses the dominance based approach.

2.1.1 Weighted Sum Method

The weighted sum method [54, 179] is one of the first methods used to solve MOO problems. The constrained MOO problem [3, 158] is reformulated as a constrained SOO problem as follows:

$$\begin{aligned} &\text{Maximize } \textstyle\sum_{k=1}^{n_o} \lambda_k f_k(\mathbf{x}) \\ &\text{Subject to } g_m(\mathbf{x}) \leq 0, \quad m = 1, \cdots, n_g \\ &\qquad\qquad h_m(\mathbf{x}) = 0, \quad m = n_g + 1, \cdots, n + n_h \\ &\qquad\qquad \mathbf{x} \in [x_{min}, x_{max}]^{n_x} \end{aligned} \tag{2.4}$$

where $\lambda_k \geq 0$ for all $k = 1, ..., n_o$, and $\sum_{k=1}^{n_o} \lambda_k = 1$. A solution to the above problem is weakly Pareto-optimal. A weak Pareto-optimal solution refers to a solution in which all criteria can not simultaneously be improved. It is Pareto-optimal if $\lambda_k > 0$ for all $k = 1, ..., n_o$, or if the solution is unique [113]. The values of λ_k are usually set by the user and are problem dependent. The above weight values need to be tuned for each new problem to obtain the best results. These weights are used to scale the objectives to be in the same range with respect their output values. The purpose of these weights is also to define the relative importance of the individual objectives during the optimization process. A larger weight assigned to one objective

with respect to other objectives would guide the search into a region where this specific objective results in relatively better objective function values than the other objectives.

2.1.2 ε-Constraint Method

The ε-constraint approach [63] to solve a MOO problem optimizes with respect to one of the objectives while considering the other objectives as constraints. The decision-maker identifies the most important objective to optimize. The optimization problem is formulated as:

$$\begin{aligned} &\text{Maximize} \quad f_{k^*}(\mathbf{x}) \\ &\text{Subject to } f_k(\mathbf{x}) \leq \varepsilon_k \qquad \text{for all} \quad k = 1, ..., n_o, \quad k \neq k^* \\ &\qquad g_m(\mathbf{x}) \leq 0, \quad m = 1, \cdots, n_g \\ &\qquad h_m(\mathbf{x}) = 0, \quad m = n+1, \cdots, n_g + n_h \\ &\qquad \mathbf{x} \in [x_{min}, x_{max}]^{n_x} \end{aligned} \tag{2.5}$$

where $k^* \in \{1, ..., n_o\}$, and ε_k are upper bounds for the objectives $f_k \neq f_{k^*}$. The solution to Equation (2.5) is weakly Pareto-optimal, because the main objective is optimized while satisfying other objectives within a certain bound. These bounds are user defined. However, a solution $\mathbf{x}^*$ is Pareto-optimal if and only if Equation (2.5) is solved for every $k^* = 1, ..., n_o$, where $\varepsilon_k = f_k(\mathbf{x}^*)$ for $k = 1, ..., n_o$, and $k \neq k^*$ [113]. To ensure Pareto optimality with this method, n_o different problems have to

be solved to optimize each of the n_o objectives.

2.1.3 Lexicographic Ordering

Lexicographic ordering ranks the objectives in order of their importance [22]. The decision maker assigns the importance to various objectives. The optimum solution, $\mathbf{x}^*$, is then obtained by optimizing the objective functions individually in order of their importance. The solution found by the current objective is fed into the next objective.

The subscripts of the objectives denote the objective function number as well as the priority of the objective. Therefore, $f_1(\mathbf{x})$ and $f_{n_o}(\mathbf{x})$ represent the most and least important objective functions, respectively. Consequently, the first problem is formulated as follows:

$$\begin{aligned} &\text{Maximize} \quad f_1(\mathbf{x}) \\ &\text{Subject to } g_m(\mathbf{x}) \leq 0, \quad m = 1, \cdots, n_g \\ &\qquad\qquad h_m(\mathbf{x}) = 0, \quad m = n_g + 1, \cdots, n + n_h \\ &\qquad\qquad \mathbf{x} \in [x_{min}, x_{max}]^{n_x} \end{aligned} \tag{2.6}$$

The solution of Equation (2.6) is referred to as $\mathbf{x}_1^*$. Then, the second problem is formulated as:

$$
\begin{aligned}
&\text{Maximize} \quad f_2(\mathbf{x}) \\
&\text{Subject to } g_m(\mathbf{x}) \leq 0, \quad m = 1, \cdots, n_g \\
&\qquad\qquad h_m(\mathbf{x}) = 0, \quad m = n_g + 1, \cdots, n + n_h \\
&\qquad\qquad \mathbf{x} \in [x_{min}, x_{max}]^{n_x} \\
&\text{and} \qquad f_1(\mathbf{x}) = \mathbf{x}^*
\end{aligned}
\tag{2.7}
$$

The solution of Equation (2.7) is referred to as $\mathbf{x}_2^*$. This procedure is then continued until solutions to all n_o objectives are found. The solution, $\mathbf{x}_{n_o}^*$, obtained at the end is the desired solution, $\mathbf{x}^*$.

2.1.4 Goal Programming

Goal programming [18, 74] is one of the first methods exclusively developed for MOO [113]. The ideal values of the objectives are defined as targets. These ideal values could be maximum or minimum attainable values depending on the type of objective. These targets are then incorporated into the optimization problem [22]. The decision-maker specifies the ideal values T_k $(k = 1, ..., n_o)$ of the objectives. Absolute deviations from these target values are minimized as much as possible [113]. For a maximization problem, goals are of the form $f_k(\mathbf{x}) \geq T_k$. The simplest form of goal programming [35] is formulated as follows:

$$\begin{aligned} &\text{Maximize} \quad \textstyle\sum_{k=1}^{n_o} |f_k(\mathbf{x}) - T_k| \\ &\text{Subject to } g_m(\mathbf{x}) \leq 0, \quad m = 1, \cdots, n_g \\ &\qquad\qquad h_m(\mathbf{x}) = 0, \quad m = n_g + 1, \cdots, n_g + n_h \\ &\qquad\qquad \mathbf{x} \in [x_{min}, x_{max}]^{n_x} \end{aligned} \tag{2.8}$$

2.1.5 Dominance-based Approach

When simultaneously optimizing all of the objectives of a MOO problem, an approach is needed to determine if one solution is better than another with reference to all the objectives. The dominance relation provides such a relational operator. A formal definition of domination by considering the MOO problem defined in Equation (2.3) is as follows.

Domination: A decision vector $\mathbf{x}_1$ dominates a decision vector $\mathbf{x}_2$ (denoted by $\mathbf{x}_1 \prec \mathbf{x}_2$) if and only if

1. $\mathbf{x}_1$ is not worse than $\mathbf{x}_2$ in all objectives, i.e. $f_k(\mathbf{x}_1) \geq f_k(\mathbf{x}_2), \forall\ k = 1, ..., n_o$, and

2. $\mathbf{x}_1$ is strictly better than $\mathbf{x}_2$ in at least one objective, i.e. $\exists\ k = 1, ..., n_o$: $f_k(\mathbf{x}_1) > f_k(\mathbf{x}_2)$.

Similarly, an objective vector $\mathbf{f}_1$ dominates another objective vector $\mathbf{f}_2$ if $\mathbf{f}_1$ is not worse than $\mathbf{f}_2$ in all objective values, and $\mathbf{f}_1$ is better than $\mathbf{f}_2$ in at least one of the

objective values. Objective vector dominance is denoted by $\mathbf{f}_1 \prec \mathbf{f}_2$. Thus, solution $\mathbf{x}_1$ is better than solution $\mathbf{x}_2$ if $\mathbf{x}_1 \prec \mathbf{x}_2$ (i.e. $\mathbf{x}_1$ dominates $\mathbf{x}_2$), which happens when $\mathbf{f}_1 \prec \mathbf{f}_2$. Amongst a set of solutions $Q^{'}$, the non-dominated set of solutions Q are those that are not dominated by any member of the set Q'. MOO algorithms that evaluate the candidate solution based on the dominance relation produce a set of non-dominated solutions. The task of a MOO algorithm is to find and maintain a diverse set of non-dominated solutions. Two dominance-based MOO algorithms namely Pareto-dominance PSO (PDPSO) [2] and NSGA-II [28] are used in this thesis. A description of NSGA-II is presented in Section 2.3.3 and PDPSO is described in Section 7.4.

2.2 Fuzzy Logic and Multi-objective Optimization

In addition to the aforementioned multi-objective aggregation techniques, fuzzy logic has also been used for multi-objective optimization. A number of studies have reported the use of fuzzy logic for MOO applications in a variety of domains [20, 76, 127, 130]. Since one of the focuses of this thesis is on fuzzy logic, an overview of fuzzy logic is provided in this section.

The theory of fuzzy sets [180, 181] is based on multi-valued logic wherein a statement can be partly true and partly false at the same time. Fuzzy logic expresses the degree of truthfulness of a statement by a *membership function*, μ, in the range [0,1].

A value of $\mu = 1$ indicates that the statement is true, whereas $\mu = 0$ indicates that the statement is false. One distinction between fuzzy logic and binary logic is that the latter allows a statement to be only completely false or completely true, which is not the case with fuzzy logic.

Fuzzy logic approaches to MOO replace the vector-based objective function with a fuzzy scalar function [155]. This approach is useful to determine solutions for the problems with uncertainties. For example, if the target is to optimize the utilization of network, then the terms such as "low utilization" or "high utilization" are uncertain. Such uncertain situations can be formulated in optimization algorithms to optimize multiple objectives of the problem. A framework for reflecting such uncertainties is conveniently provided by fuzzy logic, thus giving a strong motivation for considering a fuzzy logic approach to MOO problems.

Most MOO problems, many of which are also combinatorial optimization problems are constrained or unconstrained. These problems have been shown to be NP-hard in nature [155]. To solve these NP-hard problems, heuristics are employed, which are based on human knowledge acquired through experience and understanding of problems. Such human knowledge can be expressed in natural language of fuzzy logic. The fuzzy logic includes numerical information (examples like traffic flow on each link and delay on each link) and linguistic information (examples like maximum utilization of network and non-congestion). This natural language representation of

the problem further supports the use of fuzzy logic for solving MOO problems.

The outline of subsequent sub-sections is as follows: Section 2.2.1 discusses fuzzy sets. A brief discussion of fuzzy reasoning is presented in Section 2.2.2. Section 2.2.3 defines the linguistic variable. Fuzzy rules and fuzzy logic systems are discussed in Sections 2.2.4 and 2.2.5 respectively. Lastly, some commonly used fuzzy operators are described in Section 2.2.6.

2.2.1 Fuzzy Set Theory

A crisp set, $\mathbf{C}$, is usually defined as a set of elements or objects, $c \in \mathbf{C}$, that can be finite, countable, or uncountable. Each element either belongs to a set or not. However, for most practical problems, objects do not have crisp (1 or 0) membership to sets. Fuzzy set theory (FST) aims to represent uncertain information, such as "low utilization" or "high utilization". Such vague information is difficult to represent in classical (crisp) set theory.

A fuzzy set is characterized by a membership function which provides a measure of the degree that each element belongs to the fuzzy set [112, 185]. A fuzzy set, Y, of a universe of discourse, C, is defined as $Y = \{(c, \mu_Y(c))| \forall\ c\ \in\ C\}$, where $\mu_Y(c)$ is a membership function of c with respect to fuzzy set Y. Figure 2.1 shows an example of a membership function.

Set operations such as union, intersection, and complement, which are used in

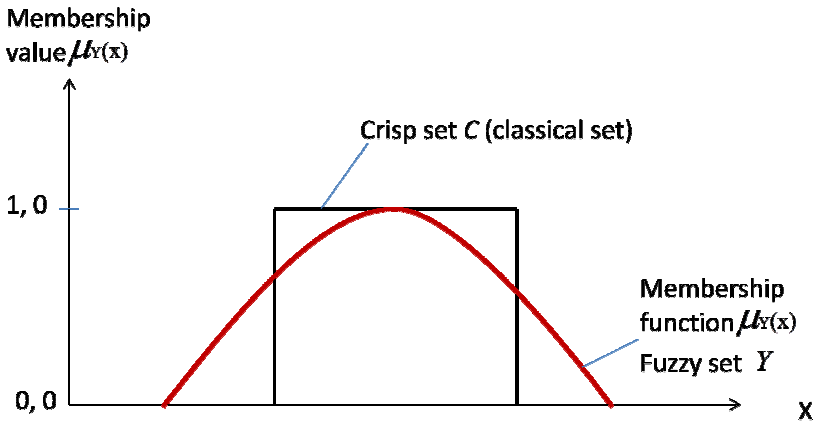


Figure 2.1: Membership function for a fuzzy set Y.

crisp sets, are also defined on fuzzy sets. A number of operators exist to represent fuzzy union and fuzzy intersection. Fuzzy union operators are known as *s-norm* operators. The s-norm operators are commonly known as "*ORing*" functions since they implement the OR operation between the membership functions under consideration. In terms of their mathematical properties, an s-norm operator satisfies the commutativity, monotonicity, associativity, and $\mu_{Y \bigcup \{0\}} = \mu_Y$ properties. Some examples of s-norm operators are given below (where Y and Z are fuzzy sets of universe of discourse, C) [112]:

- Maximum operator: $\mu_{Y \bigcup Z}(c) = \max\{\mu_Y(c), \mu_Z(c)\}$.
- Bounded sum operator: $\mu_{Y \bigcup Z}(c) = \min\{1, \mu_Y(c) + \mu_Z(c)\}$.

- Algebraic sum operator: $\mu_{Y \bigcup Z}(c) = \{\mu_Y(c) + \mu_Z(c) - \mu_Y(c)\mu_Z(c)\}$.

- Drastic sum operator: $\mu_{Y \bigcup Z}(c) = \mu_Y(c)$ if $\mu_Z(c) = 0$, $\mu_{Y \bigcup Z}(c) = \mu_Z(c)$ if $\mu_Y(c) = 0$, or $\mu_{Y \bigcup Z}(c) = 1$ if $\mu_Y(c), \mu_Z(c) > 0$.

Fuzzy intersection operators are known as *t-norm* operators. The t-norm operators represent the "ANDing" function since they implement the AND operation between the membership functions under consideration. t-norms also satisfy the commutativity, monotonicity, associativity, and $\mu_{Y \bigcap \{1\}} = \mu_Y$ properties. Examples of fuzzy intersection operators are [112]:

- Minimum operator: $\mu_{Y \bigcap Z}(c) = \min\{\mu_Y(c), \mu_Z(c)\}$.

- Algebraic product operator: $\mu_{Y \bigcap Z}(c) = \{\mu_Y(c)\mu_Z(c)\}$.

- Bounded product operator: $\mu_{Y \bigcap Z}(c) = \max\{0, \mu_Y(c) + \mu_Z(c) - 1\}$.

- Drastic product operator: $\mu_{Y \bigcap Z}(c) = \mu_Y(c)$ if $\mu_Z(c) = 1$, $\mu_{Y \bigcap Z}(c) = \mu_Z(c)$ if $\mu_Y(c) = 1$, or $\mu_{Y \bigcap Z}(c) = 0$ if $\mu_Y(c), \mu_Z(c) < 1$.

The membership function for the fuzzy complement operator is defined as

$$\mu_{\overline{Z}}(c) = 1 - \mu_Z(c)$$

2.2.2 Fuzzy Reasoning

Fuzzy logic [181] is a mathematical discipline that was exclusively developed to map the reasoning to mathematical representation. In contrast to classical reasoning, where a proposition is either true or false, fuzzy logic establishes an approximate truth value for a proposition based on *linguistic variables* and *inference rules*. A linguistic variable represents a variable whose values are words or sentences in natural or artificial language [180]. A domain expert creates *rules* with linguistic variables by using hedges, e.g. "more", "less", "few", and connectors such as AND, OR, and NOT. These rules are then used by an inference engine to facilitate decision-making.

2.2.3 Linguistic Variables

A linguistic variable is characterized by a quintuple $(\Omega,\ T(\Omega),\ C,\ R,\ \aleph)$, 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,
- C is a universe of discourse,
- R is a syntactic rule which generates the terms in $T(\Omega)$, and
- $\aleph$ is a semantic rule which associates a meaning with each linguistic value.

$\aleph(\kappa)$ denotes a fuzzy subset of C for each $\kappa \in T(\Omega)$. To clarify the meaning of a linguistic variable, consider the following example: Let C be the universe of *link utilization*, U is the fuzzy subset *link utilization near 0.5*, and $\mu_U(\bullet)$ is the membership function for U. Here, *link utilization* is a linguistic variable, i.e. $\Omega =$ *link utilization*. The linguistic values of *link utilization* can be defined as $T(\Omega) = \{$*very low utilization, low utilization, utilization near 0.5, high utilization, very high utilization*$\}$. Each linguistic value is related with a membership function which associates a meaning to that value. The universe of discourse, C, is a possible range of *link utilization*. $\aleph(\kappa)$ defines a fuzzy set for each linguistic value, $\kappa \in T(\Omega)$.

2.2.4 Fuzzy Rules

A major component of a fuzzy logic system are *rules*. The rules are expressed as logical implications, constructed as "IF-THEN" rules. The rules define relations between linguistic values of outcome (i.e. the consequent) and linguistic values of condition (i.e. the antecedent) [1]. For example, IF *link utilization is low* and *number of congested links is low* and *number of unused links is low* THEN *the solution is good*. Here *link utilization, number of congested links, number of unused links*, and *solution* are linguistic variables and *low* and *good* are linguistic values.

Rules are generally formed by domain experts, but can also be extracted from numerical data, depending on the nature of the problem. While constructing a rule,

the following important aspects should be considered [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.

2.2.5 Fuzzy Logic System

A fuzzy logic system (FLS) [112] is a model of fuzzy based decision making in various applications, such as in the engineering domain, as illustrated in Figure 2.2. A FLS consists of three components: a fuzzifier, inference engine, and a defuzzifier. The fuzzifier converts crisp input data into fuzzy input sets. The fuzzifier is needed to activate rules, which are expressed in terms of linguistic variables. The inference engine is governed by the rules which 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 converts fuzzy output to crisp values. The defuzzifier is used if an application requires crisp output data. In many optimization applications, crisp output is not required, in which case the defuzzifier is not used.

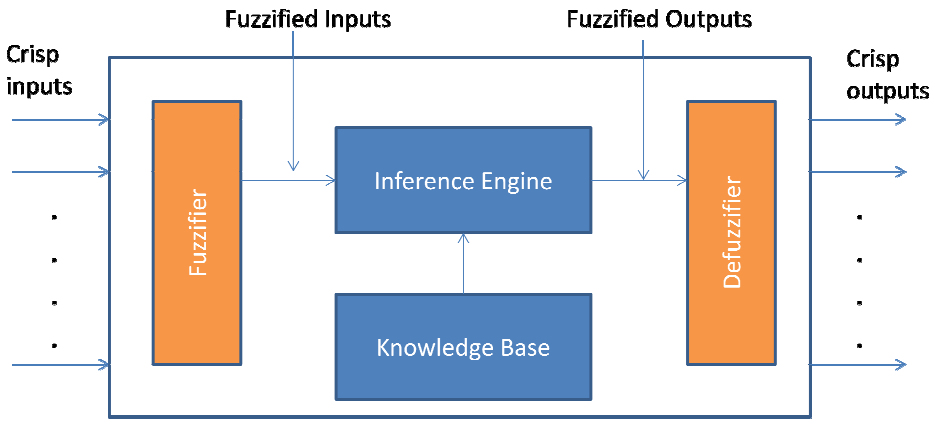


Figure 2.2: Fuzzy logic system.

2.2.6 Common Fuzzy Operators

t-norms play an important role in fuzzy logic and many other areas [5]. 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 defining that rule. Consequently, a need arises to develop t-norm operators that will effectively handle the multi-objective nature of a problem by efficiently incorporating the characteristics of different objectives (through their membership functions) into one fuzzy rule. Furthermore, since the OSPFWS 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, in-

cluding Dombi's operator [31, 32], Einstein's operator [105], Hamacher's operator [64], Frank's operator [49], Weber's operators [171], Dubois and Prade's operator [33, 34], Schweizer's operators [153], Mizumoto's operators [114, 115], Werners operator [105], and the unified and-or operator [86]. These operators are defined below:

$$\textbf{Dombi}: \qquad D(\mu_Y, \mu_Z) = \frac{1}{1 + \left(\left(\frac{1}{\mu_Y} - 1 \right)^{\delta} + \left(\frac{1}{\mu_Z} - 1 \right)^{\delta} \right)^{\frac{1}{\delta}}}, \qquad \delta > 0 \tag{2.9}$$

$$\textbf{Einstein}: \qquad E(\mu_Y, \mu_Z) = \frac{\mu_Y \mu_Z}{2 - (\mu_Y + \mu_Z - \mu_Y \mu_Z)} \tag{2.10}$$

$$\textbf{Hamacher}: \qquad H(\mu_Y, \mu_Z) = \frac{\mu_Y \mu_Z}{\delta + (1 - \delta)(\mu_Y + \mu_Z - \mu_Y \mu_Z)}, \qquad \delta \geq 0 \tag{2.11}$$

$$\textbf{Frank}: \qquad F(\mu_Y, \mu_Z) = \log_{\delta} \left(1 + \frac{(\delta^{\mu_Y} - 1)(\delta^{\mu_Z} - 1)}{\delta - 1} \right), \qquad \delta > 0, \delta \neq 1 \tag{2.12}$$

$$\textbf{Weber 1}: W_1(\mu_Y, \mu_Z) = max \left\{ 0, \left(\frac{\mu_Y + \mu_Z - 1 + \delta \mu_Y \mu_Z}{1 + \delta} \right) \right\}, \delta > -1 \tag{2.13}$$

$$\textbf{Weber 2}: W_2(\mu_Y, \mu_Z) = \max \{0, (1 + \delta)\mu_Y + (1 + \delta)\mu_Z - \delta \mu_Y \mu_Z - (1 + \delta)\}, \delta > -1 \tag{2.14}$$

$$\textbf{Dubois and Prade}: \qquad DP(\mu_Y, \mu_B) = \frac{\mu_Y \mu_Z}{max\{\mu_Y, \mu_Z, \delta\}}, \qquad 0 \leq \delta \leq 1 \tag{2.15}$$

$$\textbf{Schweizer 1}: \qquad S_1(\mu_Y, \mu_Z) = \sqrt[\delta]{max\left\{0, (\mu_Y^{\delta} + \mu_Z^{\delta} - 1)\right\}}, \qquad \delta > 0 \tag{2.16}$$

Schweizer 2 : $S_2(\mu_Y, \mu_Z) =$

$$1 - \sqrt[\delta]{(1-\mu_Y)^\delta + (1-\mu_Z)^\delta - (1-\mu_Y)^\delta(1-\mu_Z)^\delta}\ , \quad \delta > 0 \tag{2.17}$$

Werners : $Y(\mu_Y, \mu_Z) = \delta \times \min\{\mu_Y, \mu_Z\} + (1-\delta) \times \frac{1}{2}(\mu_Y + \mu_Z), \quad 0 \leq \delta \leq 1$ (2.18)

Unified and-or (UAO):

$$U(\mu_Y, \mu_Z) = \frac{\mu_Y \mu_Z + \nu \max\{\mu_Y, \mu_Z\}}{\nu + \max\{\mu_Y, \mu_Z\}} = \begin{cases} \mathrm{I}_\star = \mu_{Y \cup Z}(c) & \text{if } \nu > 1 \\ \mathrm{I}^* = \mu_{Y \cap Z}(c) & \text{if } 0 \leq \nu \leq 1 \end{cases} \tag{2.19}$$

In Equation (2.19), μ_Y represents the membership value of set Y, μ_Z represents the membership value of set Z, and $U(\mu_Y, \mu_Z)$ represents the value of the overall objective function; I^* represents the AND operation using the UAO operator, and $I_\star$ denotes the OR operation using the UAO operator. There are certain limitations for many of the above operators. For example, the Einstein, Hamacher, and Dubois and Prade operators result in a zero overall membership value if the membership value of any one of the objectives is zero, thus totally negating the impact of the membership values of other objectives. This is due to the multiplication between the membership values of the objectives in the numerator. For the operators of Dombi, Frank, Weber 1, Weber 2, Schweizer 1, and Schweizer 2, the parameter δ is bounded by a lower limit, but there is no upper limit defined. This makes it quite difficult to find an appropriate value of δ. The same concern also exists for Dombi's and

Frank's operators. The above issues, however, are not found with Werners' operator and the unified and-or operator, which make them appealing for applications with multi-criteria decision-making. Furthermore, in comparison to the Werners' operator (and all other operators mentioned above), the UAO operator has a unique feature: The UAO operator has the capability to behave as a t-norm or an s-norm by using a single equation. This is in contrast to all other operators mentioned above, which maintain two different equations for t-norm and s-norm representation. The behavior of the UAO operator is controlled by a variable $\nu \geq 0$, whose value decides whether the function would behave as AND or OR. The UAO operator has been used in a number of applications in the domain of fuzzy multi-objective optimization and decision-making [83, 85, 86, 87, 88, 89].

2.3 Optimization Algorithms

For an optimization problem, a set of possible solutions may exist. In combinatorial optimization (CO) as well as in non-CO, the best solution is a candidate solution selected from a set of possible, feasible solutions. CO has a strong relationship with discrete mathematics, probability theory, algorithmic computer science, and complexity theory [131].

CO algorithms can be grouped into two broad categories of algorithms, namely *exact algorithms* and *approximation algorithms*. Exact algorithms try to reach an ex-

act solution. Approximation algorithms try to reach an approximate solution closer to the best (or rather optimal) solution. Approximation algorithms may use either a deterministic or a random (stochastic) search strategy. Examples of exact algorithms include linear programming, dynamic programming, branch-and-bound, and backtracking [67]. Examples of approximation algorithms include local search, constructive greedy methods, and many general iterative algorithms.

Many CO problems are NP-hard. Due to their complexity, NP-hard problems can not be solved by exact techniques. Instead, approximation algorithms, also known as *heuristics*, are rather used. A heuristic explores only a sub-region of the total search space [150]. A heuristic aims to obtain "excellent" feasible solutions instead of the best solution. Because of this feature, the execution time for a heuristic, in general, is remarkably less than that of an exact algorithm.

Heuristic algorithms which attempt to improve a complete solution by making controlled stochastic moves are called iterative heuristics. Heuristic algorithms which attempt to construct a solution in a piecewise manner are called constructive heuristics. Constructive heuristics are faster than iterative heuristics, but may produce solutions of lower quality when compared to iterative heuristics [150]. For highly constrained problems, constructive heuristics may even fail to find a feasible solution [150]. Esau-William's algorithm [41], Prim's algorithm [138], and Kruskal's algorithm [97] are examples of constructive heuristics. Contrary to this, iterative

heuristics have proven to be effective for a variety of NP-hard problems in the field of engineering [13, 14, 37, 82, 98] and science [25, 144, 178].

The subsequent sections provide an overview of a number of heuristic algorithms, namely SA [121], SimE [92, 94], NSGA-II [28], and PSO [79].

2.3.1 Simulated Annealing

SA is a popular heuristic algorithm proposed by Kirkpatrick *et al.* [91]. It is derived from the analogy of the physical annealing process of metals. SA operates on a single solution. The neighborhood solution of the candidate (or current) solution is generated by randomly selecting an element and changing its value, which is known as a *move* or *perturbation*. If a move results in an improvement in objective function value, then such a move is referred to as a *good move*. Consequently, if a move does not result in an improvement in objective function value, then such a move is referred to as a *bad move*. All good moves are accepted. However, bad moves are stochastically accepted. The probability of accepting 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, a decrease in temperature leads to a decrease in the probability of accepting bad moves. In the last part of the search, SA behaves as a greedy search algorithm, accepting only good moves.

A general outline of the SA algorithm is given in Algorithm 2.1. SA takes an initial

solution, S_0, initial temperature, T_0, cooling rate, α, and the length of the Markov chain, M, as inputs. The Markov chain M represents the amount of time for which the annealing must be applied at a particular temperature. The temperature, T, is reduced per iteration by the cooling rate. The value of M increases as the temperature decreases, allowing the algorithm to perform more moves when the probability of selecting bad moves is low. This increase in the value of M is controlled by a constant β.

Algorithm 2.1 Simulated annealing $(S_0, T_0, \alpha, \beta, M, T_{max})$ [150]

```
1: (*S_0 is the initial solution *)
2: (*BestS is the best solution *)
3: (*T_0 is the initial temperature *)
4: (*α is the cooling rate *)
5: (*β a constant *)
6: (*T_max is the total allowed time for the annealing process *)
7: (*M represents the time until the next parameter update *)
8: begin
9:       T = T_0;
10:      CurS = S_0;
11:      BestS = CurS; /* BestS is the best solution seen so far */
12:      CurCost = Cost(CurS);
13:      BestCost = Cost(BestS);
14:      Time = 0;
15: repeat
16:      Call Metropolis(CurS, CurCost, BestS, BestCost, T, M);
17:      Time = Time + M;
18:      T = αT;
19:      M = βM;
20: until (Time ≥ T_max);
21: return (BestS)
22: end
```

The core function of the SA algorithm is done in the *Metropolis procedure* (see

Algorithm 2.2 Metropolis ($CurS$, $CurCost$, $BestS$, $BestCost$, T, M) [150]

```
1: begin
2: repeat
3:     NewS = Neighbor(CurS);
4:     NewCost = Cost(NewS);
5:     ΔCost = (NewCost − CurCost);
6:     if (ΔCost > 0) then
7:         CurS = NewS;
8:         if (NewCost > BestCost) then
9:             BestS = NewS;
10:        end if
11:    else
12:        if (r < e^{−ΔCost/T}) then
13:            CurS = NewS;
14:        end if
15:    end if
16:    M = M − 1;
17: until (M = 0)
18: end (* of Metropolis *)
```

Algorithm 2.2), which simulates the annealing process at a given temperature T. It takes the current solution $CurS$, current temperature T, and a value of M. In each iteration, a neighbor solution, $NewS$, of the current solution, $CurS$, is generated. If the cost of $NewS$, i.e. $NewCost$ assuming maximization, is greater than the cost of $CurS$, i.e. $CurCost$, then $CurS$ is replaced by $NewS$. If $NewCost$ is also greater than the $BestCost$, then the best solution, $BestS$, found so far is replaced by $CurS$.

If $NewCost$ is greater than $CurCost$ then a random number r is sampled from a uniform distribution in the range $(0, 1)$. If $r < e^{-\Delta/T}$, where $\Delta Cost$ = $CurCost$ - $NewCost$, then the current solution is replaced by the new solution. The probability of accepting bad moves is given by $P(r < e^{-\Delta/T})$.

The aforementioned control parameters have an impact on the convergence and quality of solutions produced by the SA algorithm [150]. Therefore, it is necessary that appropriate values of these parameters are determined for best results. One parameter that affects the convergence of SA is the initial temperature, T_0. The initial temperature should be appropriately set, so that all transitions (i.e. moves) are accepted initially. A very high initial temperature increases the algorithm's execution time, since the algorithm would extensively navigate the search space in a haphazard manner (thus increasing exploration). In contrast, a very low value of T_0 favors too much exploitation, leading to premature convergence, because the algorithm rejects bad solutions even in the early steps of the search. A number of approaches have been reported in the literature to find a suitable value for T_0 [21, 75, 110, 135].

With reference to the cooling rate, α, a high value causes the temperature to decrease slowly. This may help the algorithm to escape from local minima, since the capability of the algorithm for accepting bad solutions would persist for a considerable amount of time. In other words, the algorithm is inclined towards more exploration. Contrary to this, a very low value of α forces the algorithm to quickly lose the tendency of accepting a bad solution, thus causing the algorithm to become stuck in a local minimum. Since small changes in the solution are desired, the value of α is typically chosen in the range of 0.8 to 0.99 [100].

The length of the Markov chain, denoted by M, is also an important parameter.

The Markov chain represents the number of times that the algorithm makes perturbations at a particular temperature. An appropriate value of M is determined so as to accept a minimum number of transitions in each iteration. A very high value of M increases the execution time, since the algorithm performs more transitions than necessary. If M is too small, the solution might not be perturbed enough to search for better solutions in the current neighborhood. Thus, M controls a balance between exploration and exploitation.

Parameter β (where $\beta > 1$) is used to increase the value of M as temperature is decreased to allow the algorithm to make more moves at lower temperatures. In the early stages of the algorithm execution, a few moves are performed as these are sufficient to escape local minima. As the temperature is decreased, the algorithm's tendency to accept bad moves is reduced, thus reducing the chances of escaping local minima. In this scenario, more moves are desired at a particular temperature to escape local minima, thus requiring a higher value of M.

Some applications of simulated annealing in multi-objective optimization

Suppapitnarm [163] proposed a SA algorithm for multi-objective optimization where an archive of non-dominated solutions are maintained amongst the competing objectives. When the search ends, the designer chooses a particular solution from the obtained Pareto-optimal solutions. A weighted sum approach was used to aggregate

the objectives into a single objective function. Bandyopadhyay [6] attempted a similar approach as that of Suppapitnarm [163], and tested it on many mathematical problems. Bandyopadhyay evaluated the new solution on domination status against the current solution and the solutions present in the archive. Smith [157] proposed a multi-objective SA algorithm for optimizing code division multiple access (CDMA) mobile telecommunication networks. Venanzy and Materazzi [170] designed a multi-objective optimization approach based on SA for optimizing wind-excited structures. Li and Landa-Silva [104] implemented an evolutionary multi-objective SA algorithm incorporating both local and evolutionary search. They found improved results on bi-objective travelling salesman problem instances.

2.3.2 Simulated Evolution

SimE is a search strategy proposed by Kling and Banerjee in 1987 [95]. It is derived from the analogy of a biological evolution process, which results in heritable changes over many generations. SimE operates on a single solution, where this solution is known as the *population* as per the terminology used in [95]. Each population consists of elements or individuals. Pseudo-code for the SimE algorithm is given in Algorithm 2.3. The algorithm iteratively carries out three steps: *Evaluation, Selection* and *Allocation*. These steps are explained below.

Algorithm 2.3 Simulated Evolution(M, L) [150]

```
1: /* M: Set of movable elements; */
2: /* L: Set of locations; */
3: /* B: Selection bias; */
4: /* Stopping criteria and selection bias can be automatically adjusted; */
5: INITIALIZATION;
6: repeat
7:     EVALUATION:
8:         ForEach m ∈ M Do g_m = O_m / C_m EndForEach;
9:     SELECTION:
10:        ForEach m ∈ M Do
11:          If Selection(m,B) Then P_s = P_s ∪ {m}
12:            Else P_r = P_r ∪ {m}
13:          EndIf
14:        EndForEach;
15:    Sort the elements of P_s;
16:    ALLOCATION:
17:        ForEach m ∈ P_s Do Allocation(m) EndForEach;
18: until A stopping criterion is met;
19: Return (BestSolution);
20: End Simulated Evolution
```

Evaluation

In the evaluation step, the goodness of each element is calculated. The goodness lies in the range $[0, 1]$. Goodness is defined as follows [150]:

$$g_i = \frac{O_i}{C_i} \tag{2.20}$$

where O_i is an estimate of the optimal cost of individual i, and C_i is the actual cost of i in its current location. The goodness measure quantifies the closeness of the element to its optimal value. A higher goodness value means that the element is near to its optimal value.

Selection

Selection is the process of selecting those elements which have low goodness in the current solution. These elements are referred to as *bad elements*. In this step, for each element, a uniform random number r, in the range $(0, 1)$ is generated. If $r \leq 1 - g_m + B$, then the element is selected for change. g_m is the goodness of a movable element. B is a *selection bias* used to control the number of individuals selected. A high value of B allows more individuals to be selected for relocation, thus allowing more changes in the current solution. A low value of B indicates small changes in the current solution. A typical range for B is $[-0.2, 0.2]$. Inputs to the selection process

are elements of population P and their respective goodness values, whereas outputs are a selection set P_s and the set of remaining elements P_r of population P. The whole process is illustrated in Algorithm 2.4.

Algorithm 2.4 Selection($m,\ B$) [150]

```
1: /* m: is a particular movable element; */
2: /* B: Selection Bias; */
3:     If r ≤ 1 − g_m + B Then Return True
4:     Else Return False
5:     EndIf
6: End Selection
```

Allocation

After selection, the next step is to re-allocate the bad elements. Therefore, input to the allocation step is the elements from selection set P_s. Each element from P_s is considered individually and trial alterations are made (different random values are assigned to the element). The trial that leads to the best population with respect to the objective being optimized is accepted and made part of the solution. After the allocation step, the solution is always accepted, irrespective of whether the solution is of higher quality or lower quality than the one from which it was generated. This allows the algorithm to perform up-hill and down-hill moves through the search space, thus enabling the algorithm to escape local minima.

Some applications of simulated evolution in multi-objective optimization

A multi-objective SimE algorithm was proposed to improve a manufacturing cell in the design of aircraft and gas turbine engines [164]. Tang *et al.* discussed several SimE based multi-objective techniques [166]. Lee *et al.* [102] proposed a multi-objective SimE algorithm and applied it to a multi-objective aircraft spare parts allocation problem to find a set of non-dominated solutions. Sait *et al.* [148] solved a multi-objective VLSI cell placement problem by proposing a multi-objective fuzzy SimE algorithm.

2.3.3 Genetic Algorithms

Genetic algorithms (GAs) are the most frequently used and oldest iterative optimization algorithms. GAs emulate the natural process of evolution as a means of progressing toward an optimum. Initially suggested by Fraser [50], Fraser and Burnell [51], and Crosby [24], and popularized by Holland [66], the GA was inspired by Darwinian theory [26]. The foundation of GAs 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.

A GA operates on a set of solutions, referred to as a *population*. Each solution in this population is referred to as a *chromosome*, comprised of individual elements called *genes*. In every iteration, a new set of solutions is generated. These new solutions

are known as *offspring*. To generate offspring, a GA performs three operations in an iterative manner. These operations are *selection*, *crossover*, and/or *mutation*. The mutation operation is optional.

One crucial element of a GA is efficient representation of candidate solutions in the form of a chromosome. Furthermore, a suitable fitness function needs to be formulated to reflect the *quality* of each chromosome, or solution. 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. As the search progresses, better quality solutions are expected to be produced.

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

Selection

A pair of chromosomes (called parents) is selected to generate offspring. Generation of high-quality offspring is dependent on the choice of parents. The selection process usually favors chromosomes with high fitness values (though chromosomes with low fitness also have a non-zero probability of being selected as parents), because these chromosomes are more likely to produce stronger offspring. The selection operator is also used to select the new population. A number of selection methods such as roulette-wheel selection [61], rank-based selection [62], tournament selection [61], and

elitism [62] have been proposed in the literature.

Crossover

The primary function of crossover is to provide an inheritance mechanism whereby offspring inherit characteristics of both the parents. Crossover facilitates diversification in the search only with respect to the finite values assigned to genes during initialization. Crossover is applied at a user-defined probability, with values typically varying between 0.4 and 0.8 [172]. Many crossover mechanisms have been reported in the literature such as simple [61], cyclic [126], order [61], partially mapped [60], uniform [165], arithmetic [61], and heuristic crossover [173], amongst others.

Mutation

The key purpose of mutation is to perturb chromosomes in order to introduce characteristics which are absent in the parent population. The purpose of mutation is to increase population diversity so as to favor the exploration. 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 rate*, p_m. The most suitable value of p_m is problem-dependent [128].

2.3.4 Non-dominating sorting genetic algorithm II

To solve multi-objective optimization problems, a number of variants of GA have been developed. One such variant is NSGA-II which is by far the most successful GA approach exclusively developed to solve MOO problems [28]. The NSGA-II algorithm was used in this thesis to compare with other implemented algorithms. NSGA-II produces a set of Pareto-optimal solutions.

NSGA-II employs the concept of non-domination between two solutions in order to determine if one solution is better than another. NSGA-II uses crowding distance to ensure that a diverse set of non-dominated solutions are found. The steps involved in the NSGA-II algorithm is summarized in Figure 2.3. These steps are described below.

1. This step initializes the population using uniformly distributed random numbers.

2. All the objective function values are calculated in this step.

3. The population is ranked in this step according to non-dominating criteria. The first non-dominating front is generally assigned a rank of one. Similarly the second non-dominating front has a rank of two and so on.

4. Parents are selected and the crossover operation is applied to generate the offspring solutions. The mutation operator is then applied to these offspring solu-

tions.

5. The offspring and parent populations are combined together in order to implement elitism and non-dominating sorting is applied on the combined population.

6. The crowding distance of each solution is calculated in this step. The crowding distance is measured as the distance of the biggest cuboid containing the two neighboring solutions of the same non-dominating front in the objective space (see Figure 2.4). The higher the value of the crowding distance, the higher is the probability of the solution to be selected for the next generation. The solutions at the extremes of the non-dominating front are assigned a large value of crowding distance so as to incorporate the extremities of the non-dominating front.

7. Selection is done according to the crowding distance operator. The crowding distance operator functions as follows: a solution $\mathbf{x}$ wins the tournament with another solution $\mathbf{y}$ if (a) solution $\mathbf{x}$ has a better rank than solution $\mathbf{y}$, or, (b) if the solutions $\mathbf{x}$ and $\mathbf{y}$ have the same rank, but solution $\mathbf{x}$ has a larger crowding distance than solution $\mathbf{y}$.

8. Replace the old parent population by the better members of the combined population. The solutions of the lower ranking fronts are selected initially to replace the parent population. If all the solutions of a front can not be accommodated

in the parent population, the solutions having large crowding distance will have preference to replace the parent solutions.

Some applications of genetic algorithms in multi-objective optimization

There are a huge number of applications of NSGA-II in a variety of disciplines. Only a few of them are discussed here. Fonseca [42] developed a multi-objective optimization genetic algorithm for controller design and non-linear system identification. Abdullah *et al.* [96] presented several approaches related to the development of multi-objective genetic algorithms for multi-objective problems in the field of reliability design and optimization. Murata *et al.* [120] applied a multi-objective genetic algorithm to a flowshop scheduling problem to optimize three objectives, namely makespan, tardiness, and flowtime. Rabbani *et al.* [139] used NSGA-II combined with a clustering approach to solve the bi-objective location routing problem for waste collection. Hu *et al.* [70] used NSGA-II for multi-objective optimization for combined gas and electricity network expansion planning.

2.3.5 Particle Swarm Optimization

PSO is a population-based iterative heuristic, inspired by the flocking behavior of birds. It was proposed by Kennedy and Eberhart [79]. PSO maintains a swarm of candidate solutions. Each candidate solution is referred to as a *particle*, and possesses

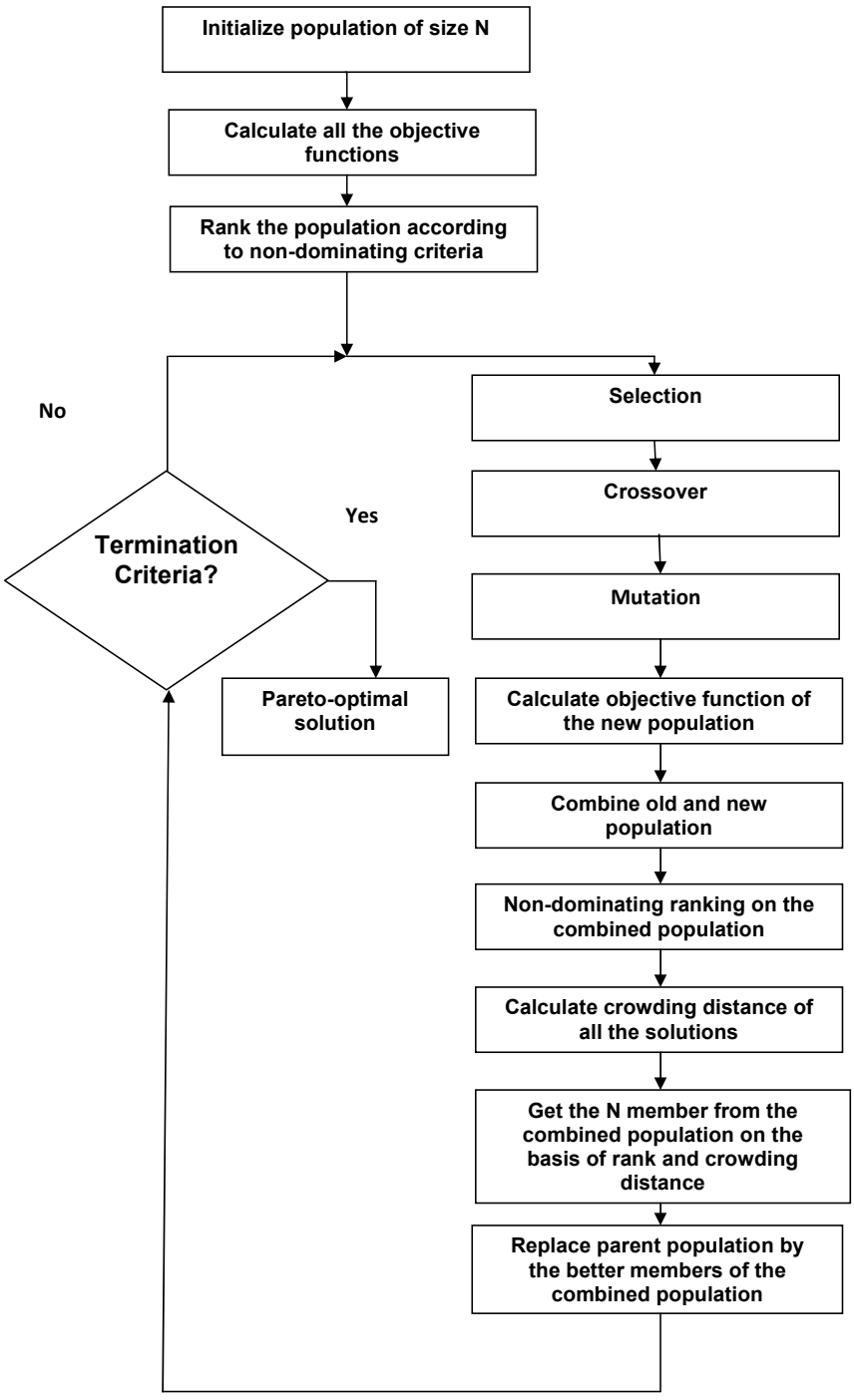


Figure 2.3: NSGA-II algorithm [28].

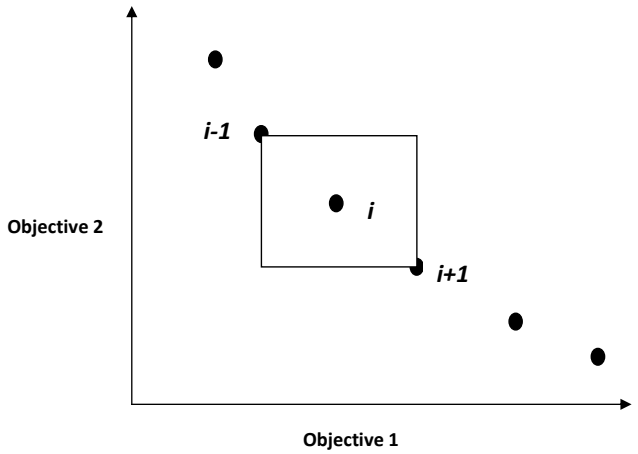


Figure 2.4: Crowding distance of a solution [28].

a memory to hold the neighborhood best position and the particle's personal best position.

PSO is initialized with a group of random particles. In every iteration, each particle position is updated by following two "best" vectors. The first vector is the best solution that the particle has achieved so far. This vector is referred to as the *personal best position*. Another "best" vector that is tracked by the PSO is the best vector obtained so far by particles in a neighborhood. This best vector is referred to as the *neighborhood best position*.

Each particle in the swarm maintains the following information:

1. $\mathbf{x}_i$: the current position of the particle;

2. $\mathbf{v}_i$: the current velocity of the particle;

3. $\mathbf{y}_i$: the personal best position of the particle; and

4. $\hat{\mathbf{y}}_i$: the neighborhood best position of the particle.

The velocity update step is specified separately for each dimension, $j = 1, ..., n_d$, where v_{ij} represents the j^{th} dimension of the velocity vector associated with the i^{th} particle. The velocity or step size of each particle i is updated using

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)] \tag{2.21}$$

where w is the *inertia weight*, c_1 and c_2 are *acceleration coefficients*, and $r_{1j}, r_{2j} \sim$ U(0,1) are two independent random values. These random values induce a stochastic component in the search process. Apart from v_{ij}, Equation (2.21) has two other main components: the cognitive component, $c_1\mathbf{r}_1[\mathbf{y}_i(t) - \mathbf{x}_i]$, and the social component, $c_2\mathbf{r}_2[\hat{\mathbf{y}}_i(t) - \mathbf{x}_i]$. These components represent a stochastic weighting between the particle position and personal best position. The position $\mathbf{x}_i$ of particle i is updated using $\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1)$. Pseudo-code for the standard PSO algorithm is given in Algorithm 2.5.

Particles within a swarm move to become more similar to their "better" neighbors. Particles within a neighborhood influence one another by exchanging information about the success of each particle in that neighborhood. Two well known PSO versions

Algorithm 2.5 PSO()

(* n_s is the size of the swarm *)
For Each particle $i \in 1...n_s$ **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 $i \in 1...n_s$ **Do**
        Evaluate the fitness of particle i
        Update $\mathbf{y}_i$
        Update $\hat{\mathbf{y}}_i$
        For each dimension $j \in 1...n_d$ **Do**
            Apply velocity update using Equation (2.21)
        End For
        Apply position update using $\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1)$
    End For
Until some convergence criterion is satisfied
End PSO

differ in the neighborhood topology used: These topologies are the star topology and the ring topology. All particles are interconnected in the star topology and thus each particle communicates with every other particle. For each particle, the entire swarm becomes its neighborhood. The result of the star topology is that all particles are attracted to one global best position. The implementation of the PSO using the star topology is referred to as the *global best* (gbest) PSO.

Contrary to this, the ring topology has each particle connected to $n_{\mathcal{N}}$ number of immediate neighbors. Each particle's neighborhood includes itself and its immediate neighbors. Each particle tries to imitate its best neighbor by moving closer to the best solution found within the neighborhood. The implementation of the PSO using

the ring topology is referred to as the *local best* (lbest) PSO.

Due to faster transfer of best position knowledge throughout the swarm, gbest PSO usually converges faster than lbest PSO. Early research claimed that gbest PSO should not be used because of its faster or premature convergence [72, 80, 108, 134] and susceptibility of being trapped in a local minimum [36, 38, 81]. Recently, Engelbrecht [39] disproved this claim and found that on a number of test functions, lbest PSO converged prematurely than gbest PSO, and lbest PSO is equally susceptible of being trapped in a local minimum as gbest PSO. Engelbrecht [39] claimed that both the algorithms (lbest PSO and gbest PSO) are equally good at finding good solutions and equally bad at finding worse solutions. Best of lbest PSO and gbest PSO is very problem dependent.

PSO Parameters

The performance of PSO algorithms depends on several control parameters [78]. These include the swarm size, inertia weight, acceleration coefficients, and velocity clamping. The influence of these parameters on performance is discussed below:

- **Swarm size**

  A larger swarm size results in increased computational complexity per iteration (because larger parts of the search space is covered), but favors higher diversity. Higher diversity facilitates more exploration of the search space [4]. Thus, a

larger swarm size may need fewer iterations to reach a good solution compared to smaller swarms. The optimal swarm size value should be determined for each problem instance.

- **Acceleration coefficients c_1 and c_2**

  The acceleration coefficients, c_1 and c_2, associated with the cognitive and social components respectively play an important role in the convergence ability of PSO algorithms. Varying these parameters has the effect of varying the strength of the pull towards the two best positions. Values of $c_1 = c_2 = 0$ indicates absence of both the cognitive and social components, and particles keep moving at their current speed until they hit a boundary of the search space [38]. With $c_1 > 0$ and $c_2 = 0$, each particle does hill climbing. Each particle searches for the best position in its neighborhood, and replaces the current best position if the new position is better [38]. However, with $c_2 > 0$ and $c_1 = 0$, the entire swarm is attracted to a single point, $\hat{\mathbf{y}}$. The entire swarm becomes a stochastic hill-climber. Furthermore, having $c_1 >> c_2$ causes each particle to be attracted to its own personal best position to a very high extent, resulting in excessive wandering. Contrary to this, $c_2 >> c_1$ results in particles being more strongly attracted to the neighborhood best position, thus causing particles to rush prematurely towards optima [38].

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

  The velocity update by Equation (2.21) includes three terms. These three terms provide the step size of particles. This velocity update quickly explodes to large values, specially for particles far from the neighborhood best and personal best positions [38]. If a particle's velocity exceeds a specified maximum velocity (V_{max}), the particle's velocity is set to the maximum velocity. Thus, velocity clamping confines the step size determined from the Equation (2.21). The value of V_{max} is very important and problem dependent. Global exploration is possible with large values of V_{max} while smaller values facilitate local exploitation. A too small value of V_{max} may take more time to reach an optimum. A too large value of V_{max} risks the possibility of ignoring promising regions of the search space. This leads to faster movement of the particles. Thus a good value of V_{max} should be found for each different problem to balance between (1) moving too fast or too slow, and (2) exploration and exploitation. However, the use of V_{max} is optional.

- **Inertia weight**

  The inertia weight, w, was introduced into PSO by Shi and Eberhart [154] in an attempt to control the velocity or step sizes. The purpose of the inertia weight is to control the impact of the previous velocities on the current velocity, thus

influencing the trade-offs between exploration and exploitation abilities of the algorithm. A larger value of w helps in searching new areas and hence increases diversity. A smaller value of w helps in concentrating a promising search area to refine a candidate solution. However, the impact of w also depends on the values of c_1 and c_2 for ensuring a guaranteed convergence to an equilibrium state [38].

Set based PSO

The solution representation for the OSPFWS problem is a set of weights on the network links. This particular aspect provides the motivation to reformulate the representation of PSO particles as sets in this thesis. This section provides a review of existing discrete PSO algorithms.

Correa *et al.* [23] proposed an algorithm having set-based characteristics for the ontology alignment problem. This algorithm possesses problem specific characteristics where each element of a particle position have its own partial objective function value. Veenhuis attempted to propose a generic set-based implementation of the PSO algorithm [169]. Velocities and positions are defined as sets. In each iteration, updating velocities and positions always increases the set size. To avoid this, a size reduction operator was introduced that uses the distance between any two set elements. Neethling and Engelbrecht proposed a set-based

PSO called the SetPSO for the RNA structure prediction problem [123]. Particles were defined as a set of stems. Though the SetPSO was a generic set implementation, its performance was found to be poor for the multidimensional knapsack problem (MKP) [101] compared to other PSO methods. Chen *et al.* proposed a generic set-based PSO called the S-PSO [19]. The S-PSO was applied to the travelling salesman problem (TSP) and the MKP. Each particle position is a fixed size set where for each "dimension" of the set an element is chosen from a set of available elements. Velocity is defined as a set which grows in size as the algorithm is executed. Wu *et al.* applied a variant of the S-PSO to the problem of cloud computing workflow scheduling [174]. Khan and Engelbrecht applied a set based fuzzy PSO to optimize topology design of distributed local area networks [89]. Velocities and positions are defined as a set of links. The velocity update is defined as a link exchange operation, which removes a single link in the position and replaces it by another. The size of the position set is fixed in their implementation. Langeveld and Engelbrecht formulated PSO algorithm in terms of set-theory and applied it to the MKP and the feature selection problem (FSP) [101]. The algorithm proposed by Langeveld and Engelbrecht is generic and does not include any problem domain specific features. Xiao *et al.* [175] presented a set-based PSO to address mapping and scheduling problems on heterogenous embedded systems. Xiao *et al.* [175] found the

performance of the proposed set-based PSO to be better than other heuristics such as ant colony optimization (ACO).

Some applications of particle swarm optimization in multi-objective optimization

Masehian and Sedighizadeh [109] presented a PSO based multi-objective algorithm for robot motion planning. The goal was to determine shortest and smoothest paths from a source location to the destination location. The multi-objective aspect of the flexible job-shop scheduling problem was addressed by Huang *et al.* [71]. The goal was to optimize the utilization of multiple machine resources. Urade and Patel [167] proposed a dynamic PSO algorithm for mathematical functions with multiple objectives by setting a dynamic value for swarm size. Lian [106] presented a multi-objective PSO algorithm for a job-shop scheduling problem. The goal was to optimize three objectives, namely the run time of every machine, earlierness time (no tardiness), and the processing time of every job.

2.4 MOO performance measures

One of the main aims of multi-objective optimization is to find a set of best solutions, i.e. the Pareto front. To assess the quality of Pareto fronts, many performance metrics, namely *generational distance*, *spread*, *spacing*, *overall non-dominated vector generation* (ONVG) and the *hypervolume* have been suggested in the literature [38, 68, 152]. To calculate the generational distance and spread, knowledge of the true Pareto front (desired Pareto front) is required. For the OSPFWS problem, such knowledge of the true Pareto front is not available. Because of this reason, in this thesis, three MOO performance metrics, namely overall non-dominated vector generation (ONVG) [68], spacing [28, 29, 68, 152], and hypervolume [28] are used to evaluate the performance of each implemented algorithm.

The ONVG metric measures the number of distinct non-dominated solutions obtained in the Pareto front. The higher the value of the ONVG measure, the better the performance of the algorithm.

The spacing metric represents a relative distance measure between consecutive solutions in the obtained non-dominated set. In other words, the spacing measure indicates the distance variance of neighboring solutions in the obtained non-dominated set. Spacing is calculated by the following equation:

$$S = \sqrt{\frac{1}{|Q|}\sum_{i=1}^{|Q|}\left(d_i - \overline{d}\right)^2} \tag{2.22}$$

where $d_i = \min_{k \in Q}\{\sum_{m=1}^{M} |f_m^i - f_m^k|\}$ and $\overline{d}$ is the average value of all d_i's, $\overline{d} = \frac{\sum_{i=1}^{|Q|} d_i}{|Q|}$. Here, Q represents the obtained non-dominated solution set and f_m^i is the m^{th} objective function value of solution i belonging to the set Q. The distance measure is the minimum value of the sum of the absolute difference in objective function values between the i^{th} solution and any other solution in the obtained non-dominated set. When the solutions are nearly spaced, the corresponding distance measure will be small. Thus, an algorithm finding a set of non-dominated solutions having smaller spacing (S) is better.

The hypervolume is the n-dimensional space that is enclosed by a n-dimensional set of points. With respect to multi-objective optimization, this set of points are treated as n-dimensional objective values. The hypervolume metric calculates the volume in the objective space covered by the members of obtained Pareto front. Mathematically, for each solution i, a hypercube is constructed with reference point W and the solution i as the diagonal corners of the hypercube. The reference point can simply be found by constructing a vector of worst objective function values. Thereafter, the hypervolume (HV) is calculated as follows:

$$HV = \sum_{i=1}^{|Q|} h_i \tag{2.23}$$

where Q represents the obtained non-dominated solution set and h_i is the volume of the hypercube constructed with solution i. An algorithm with a large value for HV

metric is desirable.

2.5 Conclusion

This chapter provided a brief overview of some optimization methods with an emphasis on evolutionary and swarm intelligence techniques. Most of the methods discussed in the chapter are used in this thesis to solve the OSPFWS 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

Routing and Open Shortest Path First Protocol

This chapter starts by discussing routing in computer networks. It is followed by a description of two interior gateway routing protocols (IGP), namely, the routing information protocol (RIP) and the open shortest path first (OSPF) routing protocol. Since the focus of this thesis is on OSPF routing and weight setting on network links, a detailed review of existing research work related to the OSPFWS problem is provided.

The outline of the chapter is as follows: Section 3.1 discusses the fundamental concepts of routing in computer networks. Section 3.2 describes the RIP and the Bellman-Ford algorithm. Details of the OSPF routing protocol and Dijkstra's algorithm are presented in Section 3.3. Lastly, the literature review related to OSPFWS

is provided in Section 3.4.

3.1 Routing in Computer Networks

Routing is a fundamental task in any computer network, consisting of finding a best or shortest path from a source node to a destination node. Routing is performed by routers in computer networks. These routers route the packets based on their destination addresses. Routing tables help routers to determine the path from the source to the destination. The routing table provides the address of the next hop router to be visited to reach a destination. The routing information present in these routing tables is collected by routing protocols.

Routing is a complex problem in large networks (such as the Internet), due to many potential intermediate nodes a packet may traverse before reaching its destination. These intermediate nodes route the traffic based on the routing tables.

Routing protocols are categorized into two general classes: (1) *distance vector routing protocols* and (2) *link state protocols*. Distance vector routing protocols use the Bellman-Ford algorithm [111]. A distance vector routing protocol enables a router to inform topology changes periodically to all the neighbors of the router. Each router share its routing table with all its neighbors. A router using a distance vector protocol does not have knowledge of the entire path to a destination. An example of a distance vector routing protocol is RIP. Contrary to this, link state routing protocols use the

Dijkstra algorithm [99]. In the link state routing protocol, every router constructs a map of complete connectivity from one router to another router in the form of a graph. An example of a link state protocol is OSPF.

Routing protocols are also categorized as interior gateway routing protocols (IGP) and exterior gateway routing protocols (EGP). An autonomous system is a group of networks and routers under the authority of a single administration. A protocol that works within an autonomous system is referred to as an *IGP*. A protocol that manages routing between autonomous systems is referred to as an *exterior gateway routing protocol* (EGP). OSPF [119] and RIP [145] are examples of IGP, while the border gateway protocol (BGP) is an example of an EGP. Because the focus of this research is on IGP, the following two sections describe the RIP and OSPF routing protocols.

3.2 Routing Information Protocol

RIP is based on request and response message communication [145]. Each router broadcasts a request message to all its neighbors and waits for response messages. In response, all the neighboring routers send their latest routing table to the requesting router. Then the router uses the latest routing table to perform routing. The routing metric in RIP is hop count, which is limited to 15 hops. If the hop count exceeds 15, then the route is considered as unreachable. By this phenomenon, RIP prevents the

formation of routing loops. Each RIP router transmits a full update of routing tables every 30 seconds. In the early days of RIP development, the networks were small and traffic was not significant. As networks grew in size, massive traffic bursts for every 30 seconds was evident. The disadvantage of RIP is that it takes much time to converge initially (i.e., for each router to have the latest and complete routing table information) and it is not scalable. There is no concept of areas or boundaries in RIP networks. RIP is suitable for small networks or autonomous systems. The computation of the shortest paths for RIP is performed using the Bellman-Ford algorithm, described below.

Bellman-Ford Algorithm

The Bellman-Ford algorithm determines the shortest path between the source and destination nodes of the network [111]. In a network, nodes may be directly or indirectly connected. For example, nodes 1 and 2 are directly connected and nodes 1 and 5 are indirectly connected as shown in Figure 3.1. A value is assigned to each link referred to as the "*link cost*". Consider d_{ij} as the cost between nodes i and j and also assume that D_{ij} is the computed minimum cost from node i to node j. If the two nodes are directly connected, then the cost d_{ij} takes a finite value. Based on the observation in Figure 3.1, nodes 4 and 6 are directly connected, thus $d_{46} = 15$. Nodes 1 and 6 are not directly connected, thus $d_{16} = \infty$. For nodes 4 and 6, the minimum cost is 2 if the path 4-3-6 is considered. Thus $D_{46} = 2$. For nodes 1 and

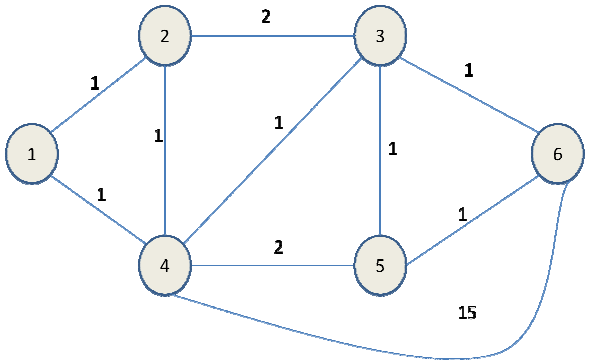


Figure 3.1: A network with link costs.

6, the minimum cost is 3 if the path 1-4-3-6 is considered. In order to calculate the minimum cost between two nodes in the network, intermediate nodes are considered if available. Consider a generic node k directly connected to the source node i. Thus, d_{ik} will have a finite value. The following equations must be satisfied by the shortest path from node i to node j:

$$D_{ii} = 0, \text{ for all } i$$

$$D_{ij} = \min_{i \neq k}\{d_{ik} + D_{kj}\}, \text{ for all } i \neq j$$

The pseudo-code of the Bellman-Ford algorithm is given in Algorithm 3.1. For simplicity, assume that node k, which is directly connected to node i, is sending the computed distance $D_{kj}(t)$ at time t to other directly connected nodes, such as i. Assume that the shortest path from node 1 to node 6 has to be found. Node 1 has nodes 2 and 4 as neighbors. Thus,

$D_{16}(t) = \min\{d_{12}(t) + D_{26}(t), d_{14}(t) + D_{46}(t)\}$.

At time $t = 0$: $D_{46}(0) = \infty$, $D_{26}(0) = \infty$, $D_{16}(0) = \min\{1 + \infty, 1 + \infty\} = \infty$.

At time $t = 1$: $D_{46}(1) = 15, D_{26}(1) = \infty, D_{16}(1) = \min\{1 + 15, 1 + \infty\} = 16$.

At time $t = 2$: $D_{46}(2) = 2, D_{26}(2) = 3, D_{16}(2) = \min\{1 + 2, 1 + 3\} = 3$.

Algorithm 3.1 Bellman-Ford Algorithm (Steps taken at node i) [111]

1: **Begin**
2: Initialize
3: $D_{ii}(t) = 0$
4: $D_{ij}(t) = \infty$ (for node j that node i is aware of)
5: **For** (nodes j that node i is aware of) **Do**
6: $D_{ij}(t) = \min_{k \text{ directly connected to } i}\{d_{ik}(t) + D_{kj}(t)\}$, for $j \neq i$
7: **End For**
8: **End**

3.3 Open Shortest Path First Routing Protocol

OSPF is a routing protocol developed for IP networks by the OSPF working group of the Internet engineering task force (IETF). It is the most commonly used intra-domain Internet routing protocol [10, 43, 73, 99, 118, 119]. OSPF enables each router to learn the complete network topology.

Each OSPF router monitors the cost (called the link state or link weight) of the link to each of its neighbors and then floods the link-state information to all other routers in the network. For this reason, OSPF is often referred to as a *link-state protocol*. The flooding of the link-state information allows each router to build a

link-state database (or topological database) that describes the complete network topology [53]. This link-state database will be identical across all the routers.

At a steady state, the routers will have the same link-state database, and so they will know how many routers are in the network, the interfaces and links between them, and the cost associated with each link. The information in the link-state database allows a router to build the shortest-path tree with itself as the root. The computation of the shortest paths is performed using Dijkstra's algorithm.

Some of the features of OSPF include:

1. Support for variable-length sub-netting (hierarchy of sub-nets of different sizes) by including the sub-net mask in the routing message.

2. A more flexible link cost that can range from 1 to 65,535. The cost can be based on any criteria, such as distance or delay.

3. Distribution of traffic (load balancing) over multiple paths that have equal cost to the destination. Equal-cost multipath is a simple form of traffic engineering.

4. Authentication schemes to ensure that routers are exchanging information with trusted neighbors.

5. Multicast rather than broadcast of its messages to reduce the load on systems that do not understand OSPF.

6. OSPF is used in large networks.

7. Unlike RIP, OSPF updates only when a change in a routing table occurs.

Dijkstra's Algorithm

The Dijkstra algorithm finds shortest paths from a source node to all other destination nodes in a network [111]. Consider a generic node i in a network of n_n nodes. It is intended to compute the shortest paths from node i to all other nodes in the network. The list of n_n nodes is denoted as $N = \{1, 2, 3, ..., n_n\}$. Also consider a generic destination node denoted as j ($j \neq i$). Consider that d_{ij} is the cost between nodes i and j, and also consider that D_{ij} is the minimum cost between node i and node j.

The Dijkstra algorithm maintains the list N as two separate lists: a permanent list L and a temporary list L'. Nodes present in the permanent list, L, are already considered, while nodes present in the temporary list L' are not yet considered. As the algorithm progresses, L expands (new nodes are added) and L' shrinks (nodes are deleted). The algorithm stops when L' becomes empty. Initially, $L = \{i\}$ and $L' = N \backslash \{i\}$ (all nodes in N except node i). The algorithm expands the list L at each iteration by considering a neighboring node k of node i with the least cost path from node i. At each iteration, the algorithm then considers the neighboring nodes of k which are not already in L to see if the minimum cost changes from the last iteration.

Consider Figure 3.1 and assume that all the shortest paths from node 1 to other

destination nodes are to be determined. Thus, initially $L = \{1\}$ and $L' = \{2, 3, 4, 5, 6\}$. The shortest paths to all nodes that are direct neighbors of node 1 can be readily found while for the rest of the nodes, the cost remains at ∞, i.e., $D_{12} = 1, D_{14} = 1, D_{13} = D_{15} = D_{16} = \infty$. For the next iteration, because node 1 has two directly connected neighbors 2 and 4, $d_{12} = 1$ and $d_{14} = 1$ respectively. Because nodes 2 and 4 are with the same minimum cost of 1, any one of the nodes can be selected. Assume node 2 is chosen. Now $L = \{1, 2\}$ and $L' = \{3, 4, 5, 6\}$. The algorithm then considers the direct neighbors of node 2 not already in set L. The neighbors of node 2 are nodes 3 and 4. The algorithm then computes the cost from node 1 to node 3 and to node 4 and checks if there is any improvement:

$$D_{13} = \min\{D_{13}, D_{12} + d_{23}\} = \min\{\infty, 1 + 2\} = 3$$

$$D_{14} = \min\{D_{14}, D_{12} + d_{24}\} = \min\{1, 1 + 1\} = 1$$

Since there is no improvement in the cost to node 4, the original shortest path is kept. For node 3, a new shortest path, 1-2-3, is obtained and for the rest of the nodes the cost remains as ∞. This completes the iteration and the process is continued to determine all the shortest paths from node 1. A formal description of Dijkstra's algorithm is given in Algorithm 3.2.

Algorithm 3.2 Dijkstras Algorithm [111]

1: Start with source node i in the permanent list of nodes considered, i.e., $L = \{i\}$;
2: All the rest of the nodes are put in the tentative list labelled as $L^{'}$
3: $D_{ij} = d_{ij}$, for all $j \in L^{'}$
4: Identify a neighboring node k not in the current list S with the minimum cost path from node i, i.e., find $k \in L^{'}$ such that $D_{ik} = \min_{m \in L^{'}} D_{im}$.
5: Add k to the permanent list L, i.e., $L = L \cup k$
6: Drop k from the tentative list $L^{'}$, i.e., $L^{'} = L^{'} \setminus k$
7: If $L^{'}$ is empty, stop
8: Consider the list of neighboring nodes, N_k, of the intermediary k (but do not consider nodes already in L), to check for improvement in the minimum cost path, i.e.,
9: $D_{ij} = \min\{D_{ij}, D_{ik} + d_{kj}\}$, for $j \in N_k \cap L^{'}$
10: Go to Step 4

3.4 Literature review of the OSPF weight setting problem

This section provides an overview of existing work from literature to solve the OSPFWS problem. Notable research in optimizing OSPF weights has been reported in the literature [8, 11, 12, 15, 40, 46, 48, 56, 77, 103, 107, 125, 132, 137, 142, 143, 146, 147, 156, 160, 162, 182]. The pioneering work on the OPSF weight setting problem was done by Fortz and Thorup [45, 47, 48], who used maximum utilization as the optimization objective. The term "maximum utilization" refers to the maximum of all link utilization values over all the link capacities in the network. A cost function based on utilization ranges was first formulated by Fortz and Thorup [46], who applied tabu search [59] to minimize "maximum utilization". The optimization function defined

by Fortz and Thorup is as follows:

$$\text{minimize} \quad \Phi = \sum_{a \in A} \Phi_a(l_a) \tag{3.1}$$

subject to the constraints:

$$l_a = \sum_{(s,t) \in N \times N} f_a^{(s,t)}, \quad a \in A, \tag{3.2}$$

$$f_a^{(s,t)} \geq 0 \tag{3.3}$$

where Φ is the cost function, Φ_a is the cost associated with arc a, l_a is the total traffic load on arc a, $f_a^{(s,t)}$ represents traffic flow from node s to t over arc a, N defines the set of nodes, and A represents the set of arcs. Equation (3.2) indicates that the total load (traffic) on arc a is equal to the sum of the traffic load on arc a and the traffic load on all incoming arcs to arc a. The constraint in Equation (3.3) states that the traffic flow from node s to t over arc a has to be greater than or equal to zero.

In Equation (3.1), Φ_a represents piecewise linear functions, with $\Phi_a(0) = 0$, and the derivative, $\Phi_a^{'}(l_a)$, is calculated using the below equation.

$$\Phi_a^{'}(l_a) = \begin{cases} 1 & \text{for } 0 \leq l_a/c_a < 1/3, \\ 3 & \text{for } 1/3 \leq l_a/c_a < 2/3, \\ 10 & \text{for } 2/3 \leq l_a/c_a < 9/10, \\ 70 & \text{for } 9/10 \leq l_a/c_a < 1, \\ 500 & \text{for } 1 \leq l_a/c_a < 11/10, \\ 5000 & \text{for } 11/10 \leq l_a/c_a < \infty \end{cases} \tag{3.4}$$

Figure 3.2 illustrates the corresponding behavior of Equation (3.4). The above function indicates that the utilization (which represents the load to capacity ratio) of a link is acceptable within 100% of the link's capacity. Fortz and Thorup employed a dynamic shortest path algorithm [44, 52, 140] to obtain multiple equidistant shortest paths between a source-destination pair. By this mechanism, traffic load was distributed equally across the links.

Subsequent to the work of Fortz and Thorup, many other researchers attempted to solve the OSPF weight setting problem with different algorithms and different objective functions. Ramakrishnan and Rodrigues [147] proposed a local search procedure using the same cost function as that of Fortz and Thorup. The main difference between the two approaches was that Rodrigues and Ramakrishnan's technique increases the link metric (i.e. the OSPF weight assigned to a link) for heavily used

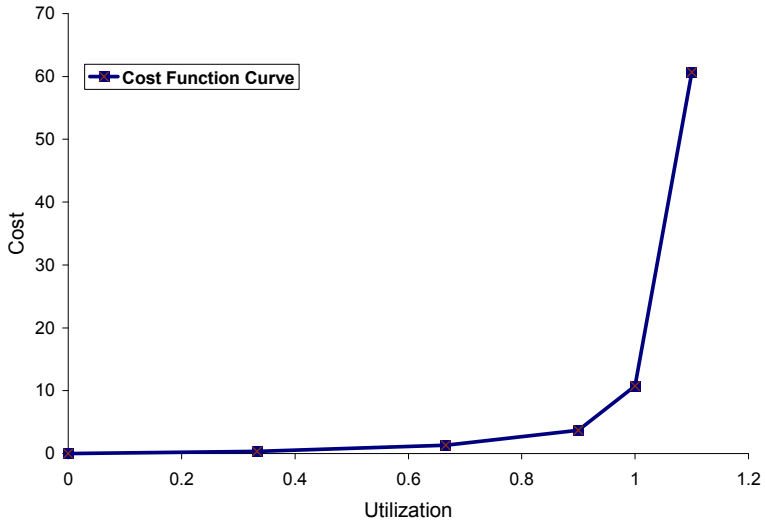


Figure 3.2: Cost function curve [46].

links. Ericsson *et al.* [40] developed a genetic algorithm [66] to solve the OSPFWS problem, also using the cost function by Fortz and Thorup. Kandula *et al.* [77] compared the performance of three OSPF weight optimizers while considering maximum link utilization as the optimization objective. Bhagat [8] proposed a hybrid genetic algorithm for the OSPF weight setting problem while using the cost model of Fortz and Thorup. Abo Ghazala *et al.* [58] performed a survey of various algorithms applied to the OSPFWS problem, and also proposed a technique based on iterative local search, while considering link utilization as the optimization objective. The underlying cost function was the same as that proposed by Fortz and Thorup. In a subsequent research article, Abo Ghazala *et al.* [56] assumed maximization of unused bandwidth as the optimization objective and employed SA and hybrid GAs for

weight optimization. Parmar *et al.* [132] formulated the OSPF weight setting problem as a mixed-integer linear programming problem and developed a branch-and-cut algorithm while assuming minimization of network congestion as the optimization objective. Pioro *et al.* [137] considered the maximum load on any link in the network as the measure of congestion and proposed two heuristic approaches for weight setting. Srivastava *et al.* [162] also considered minimization of maximum load on any link and proposed heuristic algorithms based on Lagrangian relaxation to determine feasible solutions for the weight setting problem. Buriol *et al.* [15] extended the genetic algorithm proposed in [40] to a memetic algorithm by adding a local search procedure while using the same cost function as that of Fortz and Thorup. Bley [11, 12] proposed unsplittable shortest path routing (UPSR) and claimed that the proposed approach can be applied to other routing schemes such as OSPF, while considering minimization of maximum congestion over all arcs. Zagozdzon *et al.* [182] proposed a two-phase algorithm for resolving the OSPF weight setting problem while considering the residual capacity as the optimization objective. This residual capacity resulted from setting the link weights proportional to the inverse of their capacity. Reis *et al.* [141] proposed a memetic algorithm for weight setting in OSPF and distributed exponentially-weighted flow splitting (DEFT) protocols while considering minimization of total link utilization. Lin and Gen [107] proposed a priority-based genetic algorithm for shortest path routing in OSPF. Their results indicated that the pro-

posed GA could be used for weight setting in OSPF and other routing algorithms. Retvari *et al.* [142, 143] studied the OSPF weight setting problem considering maximization of network throughput and proposed some algorithms that could efficiently optimize the link weights. Nucci *et al.* [125] proposed a tabu search heuristic for choosing link weights that takes into account both service level agreement (SLA) requirements and link failures with the objective of optimizing link utilizations. Shirmali *et al.* [156] devised an approach based on *Nash bargaining* and *decomposition* [122]. It was claimed that the proposed approach could easily be modified to yield a mechanism for setting link weights for ISPs using OSPF in a way similar to that of Fortz and Thorup. Riedl [146] presented an algorithm based on SA [91] to optimize link metrics in OSPF networks. The algorithm took into account the original routing configuration and allowed trade-off considerations between routing optimality and adaptation impact. Lee *et al.* [103] modelled the optimal link weight assignment problem as an integer linear programming problem while considering minimization of total energy consumption of all links.

It is noteworthy to mention that, generally, the aforementioned approaches considered a single objective in the optimization process. For example, the cost function proposed by Fortz and Thorup (Equation (3.1)), on which many subsequent attempts were based [8, 15, 40, 57, 58, 147, 156], considered minimization of maximum link utilization. Other researchers [77, 137, 162] also assumed minimization of maximum link

utilization. Other objectives considered in the optimization process were maximization of unused link bandwidth [56], minimization of network congestion [11, 12, 132], residual capacity of links [182], minimization of total link utilization [141], maximization of network throughput [142, 143], and minimization of total energy consumption over all links [103].

Exceptions from these single-objective optimization approaches were Nucci *et al.* [125] and Sqalli *et al.* [149, 160]. Nucci *et al.* [125] considered link failure and link no-failure states as the optimization objectives. Sqalli *et al.* [149, 160] considered minimization of maximum utilization and minimization of congested links as the optimization objectives to address OSPFWS problem.

A cost function developed by Sqalli *et al.* [149, 160] evolved from the earlier work by Fortz and Thorup. The reason for using the cost function of Fortz and Thorup was that the function was employed in many studies as mentioned above. One novel aspect of the work of Sqalli *et al.* was the addition of another optimization objective (i.e. minimization of congested links) in addition to minimization of maximum link utilization. This resulted in better distribution of traffic in the network, since one fundamental requirement of network traffic engineering is properly distributed traffic. The objective function employed by Sqalli *et al.* is defined as

$$\Phi = MU + \frac{\sum_{a \in SetCA} (l_a - c_a)}{|A|} \tag{3.5}$$

where MU is the maximum utilization of the network. $SetCA$ defines the set of congested links, $|A|$ represents the total number of links in the network, c_a refers to the capacity of link a, and l_a is the total traffic on link a. Equation (3.5) is denoted as "SqalliCF".

The second term in Equation (3.5) defines the extra load on the network. This extra load is divided by the total number of links present in the network to normalize the entire function. For an uncongested network, the second term results as zero. Thus, Equation (3.5) results in minimization of maximum utilization provided that there is no congestion in the network. If congestion exists, then the function results in the minimization of maximum utilization as well as the minimization of the number of congested links. Sqalli *et al.* concluded that the cost function in Equation (3.5) results in more efficient minimization of the number of congested links compared to the cost function of Fortz and Thorup [46]. Furthermore, Sqalli *et al.* discovered that the results for maximum utilization using Equation (3.5) were comparable to those obtained by the approach of Fortz and Thorup. Using the cost function of Equation (3.5), Sqalli *et al.* applied the SimE algorithm [93] to the OSPFWS problem and compared the results with the results of SA [149]. Tabu search using the cost function of Sqalli *et al.* [160] has also been applied to the OSPFWS problem [159].

A limitation of the cost function of Fortz and Thorup is that it minimizes "maximum utilization" only. This may lead to the existence of links which are either

congested or unused. The cost function proposed by Sqalli *et al.* (Equation (3.5)) was aimed at simultaneous optimization of maximum utilization and the number of congested links, without any consideration of unused links. It is, therefore, not guaranteed that optimizing maximum utilization and the number of congested links would implicitly optimize the number of unused links as well. This observation points to the fact that to have a more stable traffic flow, traffic from congested links should be shifted to unused links. Therefore, in order to overcome this issue, this thesis proposes a fuzzy logic based cost function that addresses the simultaneous optimization of maximum utilization, number of congested links, and number of unused links through fuzzy logic based aggregation. The details of this cost functions is present in a subsequent chapter.

3.5 Conclusion

This chapter described routing concepts, followed by a description of the RIP and OSPF routing protocols. The Bellman-Ford and Dijkstra's shortest path algorithms have also been discussed. Lastly, a brief description of existing research work pertaining to the OSPFWS problem was presented. The next chapter defines the OSPFWS as a multi-objective optimization problem.

Chapter 4

Open Shortest Path First Weight Setting Problem

This chapter provides a formal description of the OSPFWS problem. The problem is formulated as a multi-objective optimization problem using fuzzy logic. Therefore, the formation of membership functions for the design objectives is discussed along with the fuzzy logic based fitness function used to evaluate a solution.

The outline of the chapter is as follows: Section 4.1 provides the formal description of the OSPFWS problem. The description of Fuzzy logic cost function for the OSPFWS problem is given in Section 4.2. Section 4.3 provides the details of the test cases used in this thesis.

4.1 Formal description of OSPFWS Problem

The purpose of this section is to describe the OSPFWS problem. Section 4.1.1 enumerates the steps to calculate the traffic load on each link, after assigning a set of weights to the network links.

The OSPFWS problem is formulated as follows: Given a network topology and predicted traffic demands, find a set of OSPF weights that optimizes network performance. More precisely, given a directed network $G = (N, A)$, a demand matrix D, and capacity C_a for each arc $a \in A$, determine a positive integer weight $\omega_a \in [1, \omega_{max}]$ for each arc $a \in A$ such that the objective function or cost function Φ is minimized. The maximum value of this weight, ω_{max}, is a user-defined upper limit. Fortz and Thorup [45] discovered that a small range of values of weight significantly reduces the overhead of the algorithm. By experimentation, they set ω_{max} to 20. The chosen weights on arcs determine the shortest paths, which in turn completely determine the routing of traffic flow, the loads on the arcs, and the value of the cost function. The quality of OSPF routing is highly dependent on the selection of weights. Figure 4.1 shows a topology with weights assigned to each arc. These weights are in the range $[1, 20]$. A solution for this topology can be $(18, 1, 7, 15, 3, 17, 5, 14, 19, 13, 18, 4, 16, 16)$. The weights for this solution are arranged through a breadth-first traversal of the graph. For example, for node A, the weights on the outgoing links are 18 and 1. For node B, the weights on outgoing links are 7 and 15, and so on.

As discussed in the previous chapter, the existing literature considered optimizing up to two objectives, namely, maximum utilization and the number of congested links. In this thesis, a third objective, namely minimization of the number of unused links, is also considered as an optimization objective. Minimization of the number of unused links is a conflicting objective, because the less the number of unused links, the higher the maximum utilization and number of congested links. Minimizing maximum utilization will lead to a better distribution of network traffic across all links such that congestion can be avoided and the network can be utilized well as per its capacity [47]. Network administrators desire less congested links. However, if a network is highly congested, then the preference is to reduce the congestion by at least minimizing the total number of congested links. For example, assume a network with 50 congested links and 20 unused links. It would be preferred to accommodate some or all of the extra traffic of the 50 congested links on the 20 unused links. This will reduce, and even eliminate, data loss in the network since data on a link will not be exceeding the capacity of the link. Thus, such distribution will have a positive impact on the performance of the network in terms of the prevention of data loss [116]. Therefore, a new solution might create new routing paths such that traffic on congested links may be distributed on unused links.

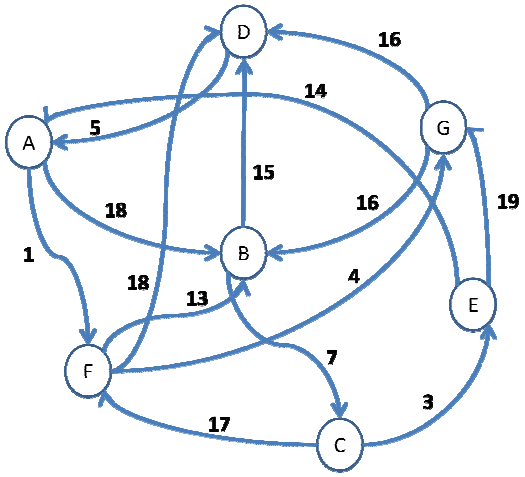


Figure 4.1: Representation of a topology with assigned weights.

4.1.1 Traffic Load Calculation

Given a weight setting, $\{\omega_a \mid a \in A, \omega_a \geq 1\}$, the arc loads l_a are calculated in five steps. For all demand pairs, $d_{st} \in D$, consider one destination t at a time and compute partial arc loads, $l_a^t \ \forall \ t \in \bar{N} \subseteq N$, where $\bar{N}$ is the set of destination nodes and N is the set of all nodes. The steps are as follows:

1. Compute the shortest distances d_u^t from each node $u \in N$ to t using Dijkstra's shortest path algorithm [30]. Dijkstra's algorithm usually computes the distances away from source s, but since it is required to compute the distance to the destination node t from all other nodes, the algorithm is applied on the graph obtained by reversing all arcs in G.

2. Compute the set A^t of arcs on shortest paths to t from all other nodes as

$$A^t = \{(u,v) \in A : \omega_{(u,v)} = d_u^t - d_v^t\}$$

3. For each node u, let δ_u^t denote its outdegree in $G^t = (N, A^t)$, i.e.,

$$\delta_u^t = \ \mid \{v \in N : (u,v) \in A^t\} \mid$$

If $\delta_u^t > 1$, then traffic flow is split at node u to balance the load.

4. The partial loads l_a^t are computed as follows:

(a) Nodes $v \in N$ are visited in order of decreasing distance d_v^t to t.

(b) When visiting a node v, for all $(v,h) \in A^t$, set

$$l_{(v,h)}^t = 1/[\delta_v^t(d_{vt} + \textstyle\sum_{(u,v)\in A^t} l_{(u,v)}^t)]$$

5. The arc load l_a is now summed from the partial loads as:

$$l_a = \textstyle\sum_{t\in\bar{N}} l_a^t$$

4.2 Fuzzy Logic Cost Function for the OSPFWS Problem

This section describes the fuzzy logic cost function for the OSPFWS problem. A solution to the OSPFWS problem is found by assigning weights to each network

link. The best solution is that one which optimizes the network resources efficiently. The design objectives of the OSPFWS problem include maximum utilization (MU), number of congested links (NOC), and number of unused links (NUL). These objectives individually on their own do not provide adequate information for deciding the quality of a solution. The conflicting nature of these objectives further amplifies the complexity of the problem. To address this complexity, a mechanism is required to find a solution that provides the best trade-off covering all the objectives. Fuzzy logic is one approach that can conveniently and efficiently handle the trade-off issues between multiple objectives.

The rest of this section discusses the use of fuzzy logic to combine the three conflicting objectives into a single overall objective function which is a scalar quantity. To formulate the overall objective function, the membership values of individual objectives need to be determined first, through membership functions. This process is described below.

To define the membership function for MU, two extreme values, the minimum and maximum, are determined first. These values are found from prior knowledge. Figure 4.2 shows the membership function of the objective to be optimized (MU in this case). Point "A" refers to minimum MU (MU_{min}) and point "B" refers to maximum MU (MU_{max}). The membership value for MU, μ_{MU}, is determined as follows:

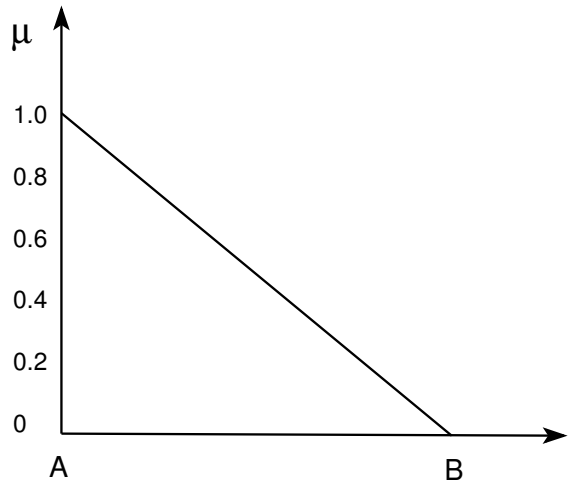


Figure 4.2: Membership function of the objective to be optimized.

$$\mu_{MU}(x) = \begin{cases} 1 & \text{if } MU \leq MU_{min} \\ \frac{MU_{max} - MU}{MU_{max} - MU_{min}} & \text{if } MU_{min} < MU \leq MU_{max} \\ 0 & \text{if } MU > MU_{max} \end{cases} \tag{4.1}$$

The membership function for NOC, μ_{NOC}, is defined in a similar way. With reference to Figure 4.2, point "A" then refers to minimum NOC (NOC_{min}) and "B" refers to maximum NOC (NOC_{max}). The membership function of NOC is defined as follows:

$$\mu_{NOC}(x) = \begin{cases} 1 & \text{if } NOC \leq NOC_{min} \\ \frac{NOC_{max} - NOC}{NOC_{max} - NOC_{min}} & \text{if } NOC_{min} < NOC \leq NOC_{max} \\ 0 & \text{if } NOC > NOC_{max} \end{cases} \tag{4.2}$$

Finally, the membership function for NUL, μ_{NUL}, is defined as

$$\mu_{NUL}(x) = \begin{cases} 1 & \text{if } NUL \leq NUL_{min} \\ \frac{NUL_{max} - NUL}{NUL_{max} - NUL_{min}} & \text{if } NUL_{min} < NUL \leq NUL_{max} \\ 0 & \text{if } NUL > NUL_{max} \end{cases} \tag{4.3}$$

where minimum (NUL_{min}) and maximum (NUL_{max}) values correspond to "A" and "B" respectively in Figure 4.2.

A good solution to the OSPFWS problem is one that is characterized by a low MU, low NOC, and low NUL. In fuzzy logic, this can be stated by the following fuzzy rule:

Rule: IF a solution X has low MU AND low NOC AND low NUL THEN it is a good solution.

The words "low MU", "low NOC" and "low NUL" are linguistic variables, each defining a fuzzy subset of solutions. Using the UAO operator [86], the above fuzzy rule reduces to the following equation:

$$\mu(x) = \frac{\mu_{MU}(x)\mu_{NOC}(x)\mu_{NUL}(x) + \nu \times max\{\mu_{MU}(x), \mu_{NOC}(x), \mu_{NUL}(x)\}}{\nu + max\{\mu_{MU}(x), \mu_{NOC}(x), \mu_{NUL}(x)\}} \quad (4.4)$$

where $\mu(x)$ is the membership value for solution x in the fuzzy set "good OSPF weight set" and ν is a constant in the range [0,1]. The solution which results in the maximum value for Equation (4.4) is reported as the best solution.

As an example, consider an arbitrary solution S_1, having $\mu_{MU} = 0.19$, $\mu_{NOC} = 0.2$, and $\mu_{NUL} = 0.17$. Also assume that $\nu = 0.5$. Then, Equation (4.4) results in a value of 0.152. Similarly, consider $\mu_{MU} = 0.22$, $\mu_{NOC} = 0.23$, and $\mu_{NUL} = 0.09$ associated with another arbitrary solution S_2. Again assume that $\nu = 0.5$. Then, Equation (4.4) evaluates to 0.164. Thus, solution S_2 is better than solution S_1 in terms of quality. Equation (4.4) is employed as a fuzzy cost function for solving the OSPFWS problem and is denoted as "FuzzyCF".

4.3 Details of Test Cases

This research work used the test cases proposed by Fortz and Thorup [46]. Table 4.1 shows the characteristics of the test cases. For each test case, the table lists its network type, the number of nodes, and the number of links. The *2-level hierarchical networks* are generated using the GT-ITM generator [183], based on the model of Calvert [17]

and Zegura [184]. In hierarchical networks, short distance arcs have capacities equal to 200, while long distance arcs have capacities equal to 1000. In *random networks* and *Waxman networks*[1], capacities are set at 1000 for all arcs. Fortz and Thorup generated the demands to force some nodes to be more active senders or receivers than others, thus modelling *hot spots*[2] on the network. The demands generated by Fortz and Thorup assign higher demands to closely located node pairs. Further details on the assignment process of generated demands can be found in Fortz and Thorup [47].

Table 4.1: Test cases for the OSPFWS problem (N = number of nodes, A = number of arcs.). Smallest test case is in italics and the biggest test case is in boldface.

Test Code	**Network type**	**N**	**A**
h100N280a	2-level hierarchical graph	100	280
h100N360a	2-level hierarchical graph	100	360
h50N148a	*2-level hierarchical graph*	*50*	*148*
h50N212a	2-level hierarchical graph	50	212
r100N403a	Random graph	100	403
r100N503a	**Random graph**	**100**	**503**
r50N228a	Random graph	50	228
r50N245a	Random graph	50	245
w100N391a	Waxman graph	100	391
w100N476a	Waxman graph	100	476
w50N169a	Waxman graph	50	169
w50N230a	Waxman graph	50	230

[1]Waxman graphs are frequently chosen in simulations as topologies resembling communications networks. Waxman graphs are named after Bernard M. Waxman.

[2]A hot spot is a network point or router having heavy incoming and outgoing traffic.

4.4 Conclusion

This chapter defined the OSPFWS problem as a multi-objective optimization problem. Three design objectives, namely, maximum utilization, number of congested links, and number of unused links are considered as the optimization objectives. The three objectives are aggregated using the unified and-or operator.

The next chapter discusses the application of a multi-objective FSA algorithm to solve the OSPFWS problem.

Chapter 5

Fuzzy Simulated Annealing Algorithm for the OSPFWS Problem

This chapter discusses the multi-objective FSA algorithm developed to solve the OSPFWS problem. The fuzzy cost function (denoted as FuzzyCF) described in Chapter 4 has been used as the underlying objective function. The chapter also provides a sensitivity analysis of the FSA control parameters. The results of FSA are compared with existing work in the literature.

The outline of the chapter is as follows: Section 5.1 provides implementation details of the FSA algorithm. Section 5.2 describes the solution archiving mechanism

used by the FSA algorithm. Section 5.3 discusses the experimental setup used for FSA simulations. This section also discusses the effect of the cooling rate parameter α on the quality of solutions with respect to FuzzyCF. Section 5.3.1 provides a comparison of the SqalliCF and FuzzyCF cost functions using the SA algorithm.

5.1 Fuzzy Simulated Annealing Algorithm

An algorithmic description of SA was given in Section 2.3.1. SA is applied as a multi-objective algorithm to solve the OSPFWS problem by maximizing the cost function, FuzzyCF, given in Equation (4.4). The OSPFWS problem requires an assignment of a set of weights to the links in the network. This set of weights is a solution to the OSPFWS problem. Thus in the context of the FSA algorithm, the weights are the movable elements. A change in the weight of a link leads to different shortest paths between source and destination pairs, resulting in a new solution, and thus providing a different cost.

The FSA algorithm executes three main steps, namely, initialization, the metropolis procedure, and evaluation. These steps are described below.

Initialization

The initialization step generates a random solution (i.e. a random set of weights). The FSA control parameters, i.e. the initial temperature T_0, the cooling rate α, the constant β, maximum time for the annealing process T_{max} (in terms of the number

of iterations), and the length of Markov chain, M, are also initialized.

The FSA control parameters have an impact on the convergence of the algorithm. For example, the initial temperature T_0 is set to an appropriate value, so that all moves are accepted initially. The appropriate value is determined by a trail and error approach. A very high initial temperature causes the algorithm to search blindly. Contrary to this, a very low initial temperature results in bad solutions to be rejected in the early stage of the search, therefore limiting exploration. A number of methods have been proposed in the literature to determine an appropriate initial temperature [21, 75, 91, 110, 135]. This thesis adopts the method proposed by Kirkpatrick [91], where the value of T_0 is chosen such that the initial acceptance ratio, $X(T_0)$, is close to unity. The acceptance ratio is calculated using

$$X(T_0) = \frac{N_{T_0}}{T_{T_0}} \tag{5.1}$$

where N_{T_0} is the number of moves accepted at temperature T_0 and T_{T_0} is the total number of moves attempted at temperature T_0.

The cooling rate, α, controls the rate of decrease in temperature. The higher the value of α, the lower the decrease in temperature, and vice versa. A typical value of α ranges from 0.8 to 0.99 [100].

Parameter M also has an impact on the convergence of FSA, and it is very important to determine an appropriate value of M. A very high value of M increases

the execution time. As an example, consider $M = 35$. This means that, at a given temperature, the algorithm will attempt 35 moves. It is quite probable that the same quality of solution is achieved with $M = 15$. Therefore, 20 moves are wasted. Similarly, when M is very low (for example 5), then the algorithm may result in poor quality solutions, simply because of the fact that the algorithm does not have sufficient time to explore the search space for a given temperature.

Metropolis procedure

The core procedure of the annealing algorithm is known as the *metropolis procedure*. This procedure is repeated for a given number of iterations. During the metropolis procedure, a solution is perturbed through a move. For the OSPFWS problem, a move involves selecting a weight from the set of given weights, and replacing the selected weight with a different weight. For example, with reference to Figure 4.1, a solution for this topology is $(18, 1, 7, 15, 3, 17, 5, 14, 19, 13, 18, 4, 16, 16)$. FSA selects a link randomly and replaces its weight with another random weight. Changing any weight from the above solution will lead to a different solution in the search space. Thus a new solution can be $(18, 1, 7, 15, 3, \mathbf{5}, 5, 14, 19, 13, 18, 4, 16, 16)$ by changing the weight 17 to 5 on the link between node C and E. As shown in Figure 5.1, in the subsequent iterations, weight 18 is changed to 9 on the link between nodes F and D, weight 18 is changed to 10 on the link between nodes A and B, and weight 7 is changed to 6 on the link between nodes B and C. Note that after each step shown, a

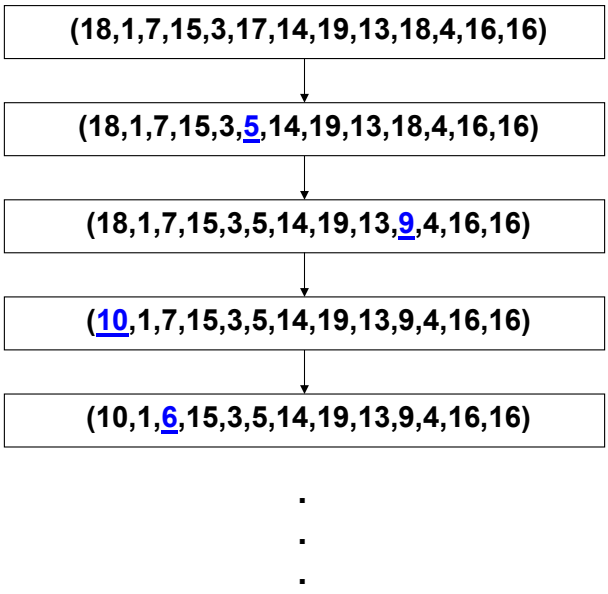


Figure 5.1: Sequence of moves in FSA.

new solution is generated. This solution is evaluated and compared with the previous solution. If the cost of a new solution is better than the cost of the immediate previous solution, then the new solution is accepted. If the cost of the new solution is less than the cost of the immediate previous solution, then the new solution is probabilistically accepted as explained in Section 2.3.1.

Evaluation of a solution

Once a new solution is generated, its cost should be computed. This is done using FuzzyCF (Equation (4.4)). For each individual objective, minimum and maximum values are a prerequisite for obtaining the membership values. These extreme values are found as follows: For objective MU, the minimum value, MU_{min}, is taken as

the minimum utilization of the initial solution given to the algorithm during the initialization phase. The maximum value for MU, MU_{max}, is taken as the maximum utilization of the initial solution. The minimum value for objective NOC, NOC_{min}, is set to zero. The maximum value for NOC, NOC_{max}, is taken as the number of congested links obtained from the initial solution. In the same way, NUL_{min} is set to zero, while NUL_{max} is taken as the number of unused links obtained from the initial solution. In the subsequent steps, membership values for each objective are calculated by using Equations (4.1), (4.2) and (4.3).

Stopping criterion

The algorithm is stopped when a maximum number of iterations has been reached. The next section describes the archiving procedure used in FSA.

5.2 Solutions Archiving in FSA

In MOO there is a set of Pareto-optimal solutions, also known as the *Pareto front*. The Pareto-optimal solutions are trade-off non-dominated solutions, and all of them are global optimum solutions [6]. Since FSA is an iterative single-solution based algorithm that generates a single solution in each iteration, the algorithm can not produce Pareto-optimal solutions by its nature. In this thesis, a simple archiving scheme is implemented during the execution of the proposed FSA. This is to determine whether FSA has the capability to yield a set of non-dominated solutions. The initialization

step places the first solution in the archive. Then, the following procedure is used to maintain the archive during subsequent iterations:

1. If the obtained new solution is non-dominating with all the members of archive, then new solution is added to the archive.

2. If all the members of archive are dominated by the new solution, then all the members are deleted from the archive and the new solution is added to the archive.

3. If the new solution is not dominating any one of the members of archive, then the new solution is not added to the archive.

5.3 Experimental setup

This section discusses the parameters used for SA with respect to FuzzyCF and SqalliCF. For each test case (refer to Section 4.3 for the details on these test cases), 30 independent runs were executed for 5000 iterations. The average of the best cost and the standard deviation over these 30 runs were calculated. For all experiments with respect to SqalliCF, the parameters $\alpha = 0.965$ and $\beta = 1.01$ were used as recommended in a previous study by Sqalli *et al* [160]. Experiments were conducted to obtain the best parameter values of α and M with respect to FuzzyCF. The first set of experiments focussed on obtaining the best cooling rate parameter, α. Initially,

the values $\beta = 1.01$ and $M = 20$ were used. The initial α value was set as 0.99 for each test case. Subsequently, the α value was decreased by 0.01. As soon as a lower α value produced a statistically significant worse result than a previously evaluated higher α value, the higher α value was considered as the best value. For example, if the α value 0.95 produced statistically significant worse result than the result of α value 0.96, then the α value 0.96 was considered as the best α value. Tables 5.1 and 5.2 provide the quality of solutions with respect to FuzzyCF obtained for all the experimented cooling rate values. The penultimate column of the table represents the percentage difference between the average fuzzy cost values obtained for two consecutive α values. Because an α value of 0.99 is used as the initial value, the percentage difference value for $\alpha = 0.99$ is NA.

Besides these tables, graphic representations of the search progress for each experimented value of α with respect to all test cases are also shown in Figures 5.2, 5.3, 5.4 and 5.5. The results show that $\alpha = 0.96$ performed well for all the test cases and hence motivated the use of $\alpha = 0.96$ for all further experiments. After setting the value of α, further experimentation was done with values of the Markov chain parameter $M = 20$ and $M = 30$ for all test cases with respect to FuzzyCF and SqalliCF. As observed from Table 5.3, in general, $M = 20$ produced better results for both SqalliCF (a lower value is desired) and FuzzyCF (a higher value is desired). There are exceptions with respect to test cases h50N148a, r100N503a and w50N169a

for the FuzzyCF cost function. However, since $M = 20$ requires less perturbations, and thus results in less execution time, $M = 20$ is preferred over $M = 30$. Therefore, all experiments were conducted using $M = 20$ for both FuzzyCF and SqalliCF.

Figures 5.6 and 5.7 show the plots of the average fuzzy cost and standard deviation for all the test cases over 30 independent runs with respect to FSA. The search accepted bad solutions in the early phases of execution, and behaved greedily towards the end, accepting only the good solutions. As the search progresses, successive temperature values decrease depending on the value of cooling rate α. The decrease in temperature decreases the probability of accepting bad moves. It is also observed from the figures that the standard deviation decreased over time. A decrease in standard deviation confirms that all the results over these 30 independent runs converge towards similar optimal solutions. Similar search progress trends and convergence towards similar optimal solutions were also found with respect to SqalliCF as shown in Figures 5.8 and 5.9. Note that FuzzyCF is a maximization function, while SqalliCF was modelled as a minimization function.

5.3.1 Comparison of SqalliCF and FuzzyCF using SA

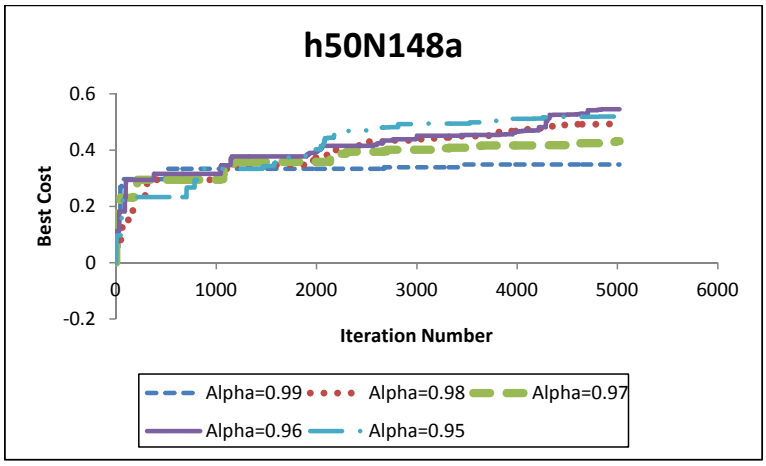
Table 5.4 summarizes the results obtained for FSA using SqalliCF and FuzzyCF. Furthermore, the average runtime for each test case with respect to the two cost functions is also provided. Table 5.5 shows the percentage improvement achieved by

Table 5.1: Best average FuzzyCF values for the test cases h50N148a, h50N212a, h100N280a, h100N360a, r50N228a and r50N245a with respect to experimented cooling rate values. Statistically significant differences are in italics. NA = Not Applicable (% Difference is calculated between the average fuzzy cost values obtained for two consecutive cooling rate values).

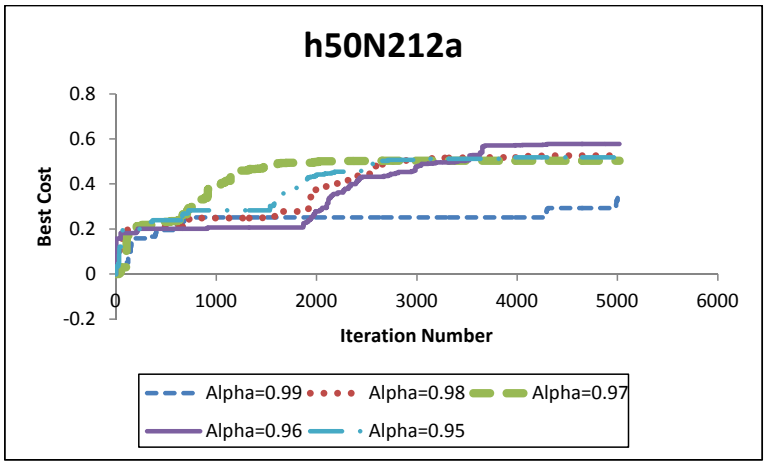
Test Case	Cooling rate α	Best Fuzzy Cost	% Difference	p-values
h50N148a	0.99	0.348 ± 0.102	NA	NA
	0.98	0.391 ± 0.021	*11*	0.043
	0.97	0.431 ± 0.095	*9.28*	0.037
	0.96	0.547 ± 0.072	*21.21*	0.005
	0.95	0.522 ± 0.099	-4.79	0.187
h50N212a	0.99	0.361 ± 0.068	NA	NA
	0.98	0.417 ± 0.083	*13.43*	0.016
	0.97	0.473 ± 0.072	*11.84*	0.038
	0.96	0.575 ± 0.075	*17.74*	0.03
	0.95	0.517 ± 0.008	-11.22	0.216
h100N280a	0.99	0.3 ± 0.041	NA	NA
	0.98	0.328 ± 0.041	*8.54*	0.011
	0.97	0.356 ± 0.116	*7.87*	0.014
	0.96	0.585 ± 0.154	*39.15*	0.024
	0.95	0.554 ± 0.078	-5.6	0.126
h100N360a	0.99	0.287 ± 0.114	NA	NA
	0.98	0.312 ± 0.096	*8.01*	0.012
	0.97	0.333 ± 0.036	*6.31*	0.009
	0.96	0.6 ± 0.055	*44.5*	0.033
	0.95	0.57 ± 0.048	-5.26	0.119
r50N228a	0.99	0.351 ± 0.006	NA	NA
	0.98	0.426 ± 0.116	*17.61*	0.017
	0.97	0.502 ± 0.047	*15.14*	0.014
	0.96	0.581 ± 0.025	*13.6*	0.035
	0.95	0.542 ± 0.114	-7.2	0.167
r50N245a	0.99	0.307 ± 0.002	NA	NA
	0.98	0.413 ± 0.062	*25.67*	0.03
	0.97	0.52 ± 0.014	*20.58*	0.035
	0.96	0.613 ± 0.028	*15.17*	0.04
	0.95	0.571 ± 0.114	-7.36	0.245

Table 5.2: Best average FuzzyCF values for the test cases r100N403a, r100N503a, w50N169a, w50N230a, w100N391a and w100N476a with respect to experimented cooling rate values. Statistically significant differences are in italics. NA = Not Applicable (% Difference is calculated between the average fuzzy cost values obtained for two consecutive cooling rate values).

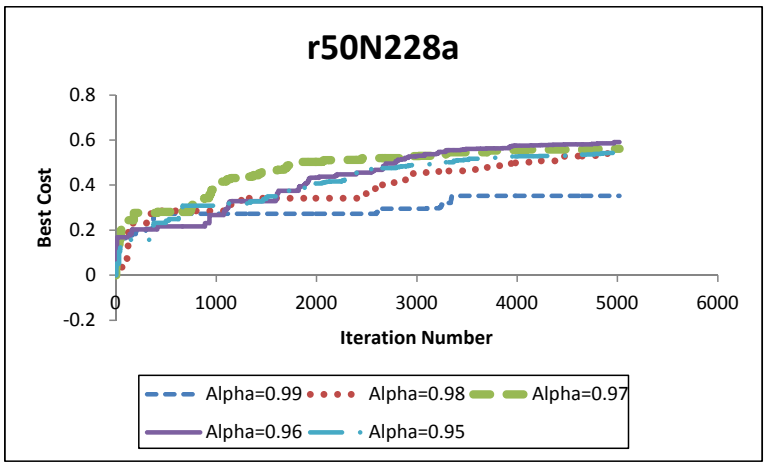
Test Case	Cooling rate α	Best Fuzzy Cost	% Difference	p-values
r100N403a	0.99	0.309 ± 0.056	NA	NA
	0.98	0.321 ± 0.084	*3.74*	0.017
	0.97	0.333 ± 0.111	*3.6*	0.021
	0.96	0.487 ± 0.185	*31.62*	0.014
	0.95	0.482 ± 0.125	-1.04	0.186
r100N503a	0.99	0.265 ± 0.023	NA	NA
	0.98	0.274 ± 0.072	*3.28*	0.029
	0.97	0.283 ± 0.107	*3.18*	0.037
	0.96	0.507 ± 0.32	*44.18*	0.032
	0.95	0.479 ± 0.008	-5.85	0.174
w50N169a	0.99	0.432 ± 0.023	NA	NA
	0.98	0.509 ± 0.053	*15.13*	0.008
	0.97	0.587 ± 0.055	*13.29*	0.032
	0.96	0.692 ± 0.018	*15.17*	0.029
	0.95	0.668 ± 0.125	-3.59	0.152
w50N230a	0.99	0.302 ± 0.006	NA	NA
	0.98	0.355 ± 0.053	*14.93*	0.003
	0.97	0.502 ± 0.076	*29.28*	0.041
	0.96	0.705 ± 0.138	*28.79*	0.022
	0.95	0.667 ± 0.078	-5.7	0.234
w100N391a	0.99	0.376 ± 0.129	NA	NA
	0.98	0.416 ± 0.068	*9.62*	0.017
	0.97	0.456 ± 0.115	*8.77*	0.024
	0.96	0.628 ± 0.065	*27.39*	0.014
	0.95	0.602 ± 0.109	-4.32	0.218
w100N476a	0.99	0.359 ± 0.053	NA	NA
	0.98	0.445 ± 0.075	*19.33*	0.005
	0.97	0.53 ± 0.061	*16.04*	0.036
	0.96	0.606 ± 0.032	*12.54*	0.021
	0.95	0.542 ± 0.032	-11.81	0.157



(a)

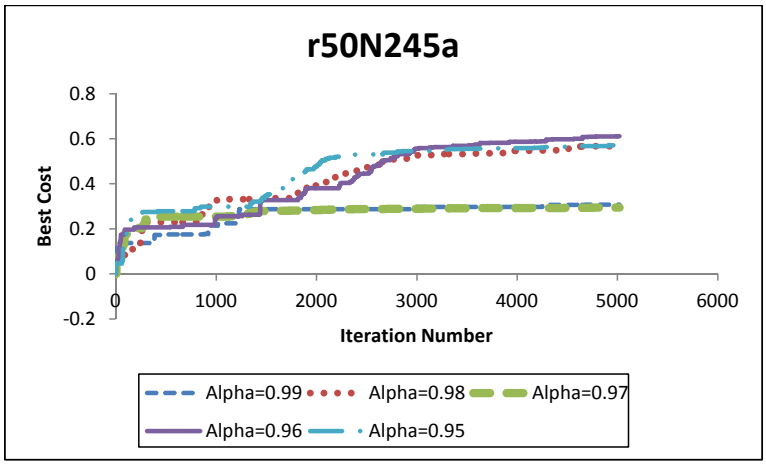


(b)

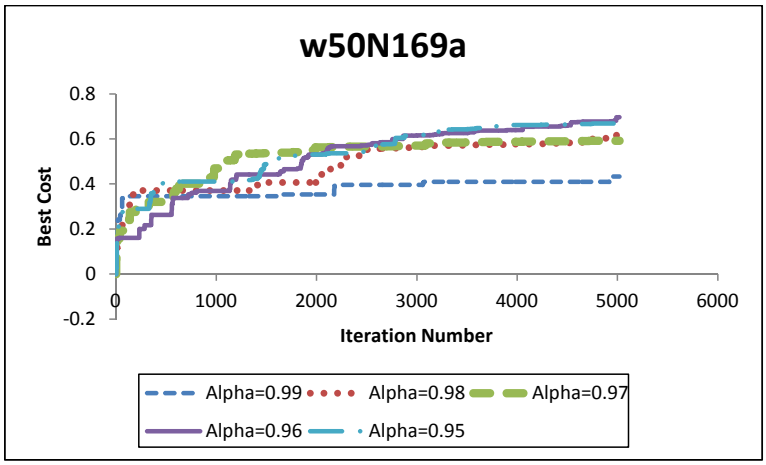


(C)

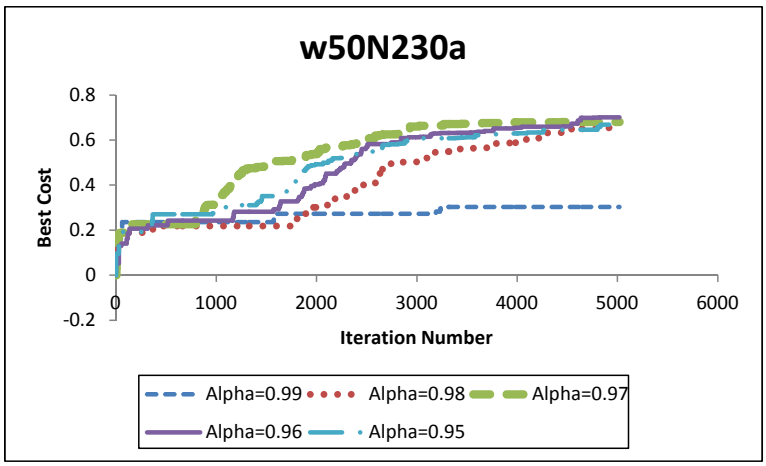
Figure 5.2: Plots of FSA average best cost for all the values of α with respect to the test cases (a) h50N148a (b) h50N212a and (c) r50N228a. Search with value $\alpha = 0.96$ produced better results.



(a)

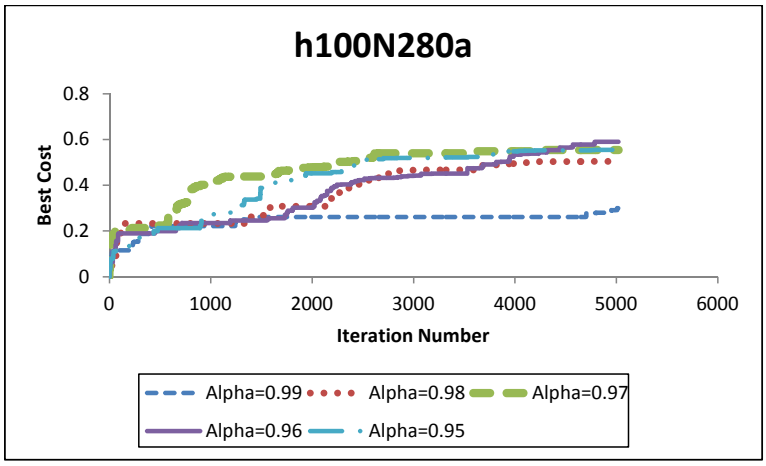


(b)

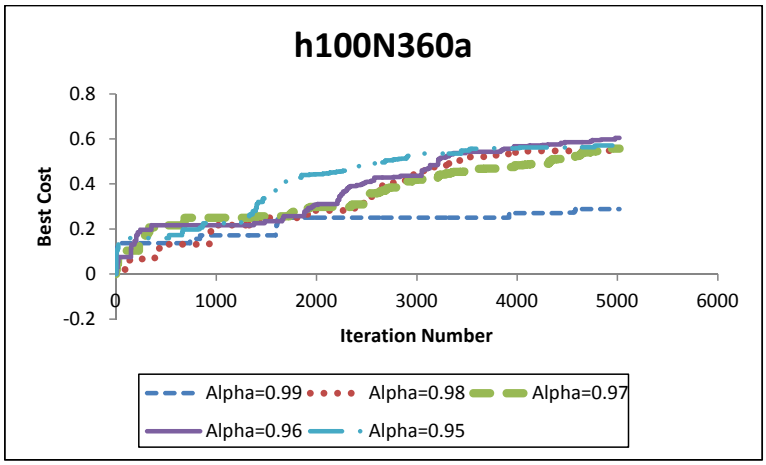


(c)

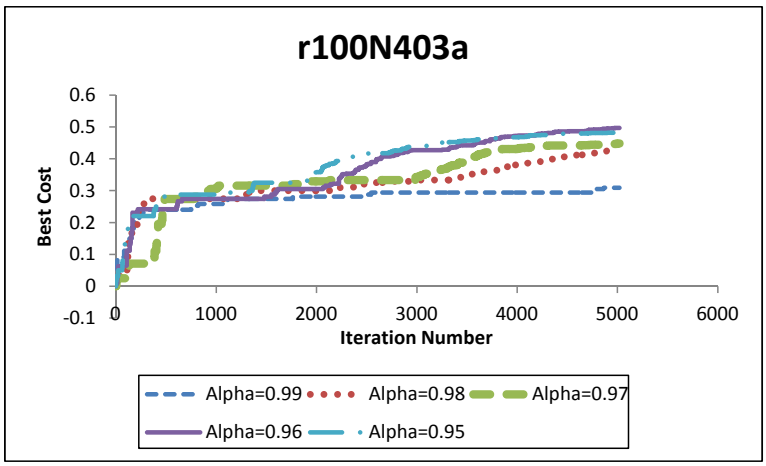
Figure 5.3: Plots of FSA average best cost for all the values of α with respect to the test cases (a) r50N245a (b) w50N169a and (c) w50N230a. Search with value $\alpha = 0.96$ produced better results.



(a)

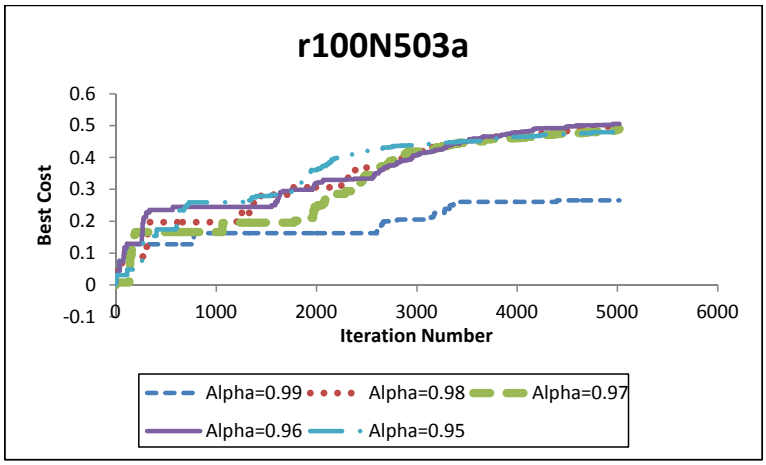


(b)

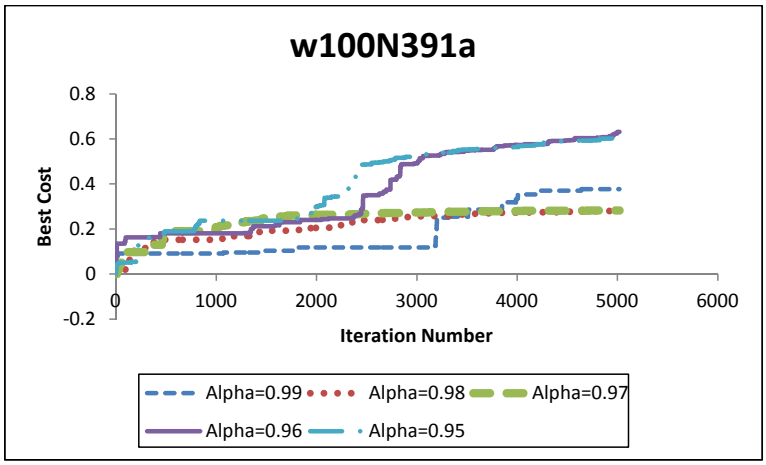


(c)

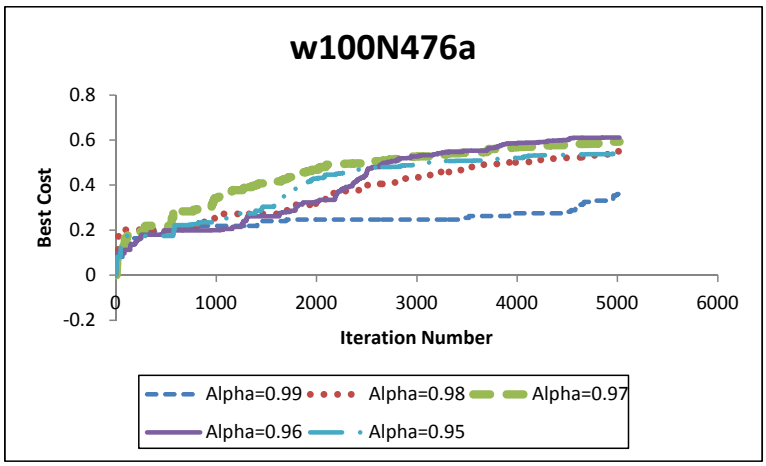
Figure 5.4: Plots of FSA average best cost for all the values of α with respect to the test cases (a) h100N280a (b) h100N360a and (c) r100N403a. Search with value $\alpha = 0.96$ produced better results.



(a)



(b)



(c)

Figure 5.5: Plots of FSA average best cost for all the values of α with respect to the test cases (a) r100N503a (b) w100N391a and (c) w100N476a. Search with value $\alpha = 0.96$ produced better results.

Table 5.3: Results of average cost for different values of Markov chain parameter (M) for SqalliCF and FuzzyCF. Imp. shows % improvement achieved with $M = 20$ relative to $M = 30$. Significant % improvement is in boldface. Inferior performance with $M = 20$ for FuzzyCF is in italics.

Test Case	SqalliCF			FuzzyCF		
	$M = 20$	$M = 30$	% Imp.	$M = 20$	$M = 30$	% Imp.
h50N148a	2.451±0.04	3.223±0.12	23.95	0.428±0.27	0.518±0.03	*-21.03*
h50N212a	2.498±0.37	3.395±0.03	26.42	0.461±0.10	0.261±0.01	43.38
h100N280a	2.654±0.10	2.688±0.02	1.27	0.543±0.04	0.519±0.02	4.42
h100N360a	3.003±0.03	4.994±0.03	39.87	0.308±0.02	0.217±0.01	29.55
r50N228a	14.618±1.23	30.453±2.03	52.00	0.588±0.09	0.561±0.01	4.59
r50N245a	36.437±6.12	55.253±5.01	34.05	0.56±0.04	0.552±0.03	1.43
r100N403a	26.811±2.30	49.309±5.01	45.63	0.457±0.02	0.375±0.09	17.94
r100N503a	29.145±2.05	59.54±4.03	51.05	0.291±0.02	0.44±0.05	*-51.20*
w50N169a	3.6±0.01	13.461±1.12	**73.26**	0.558±0.03	0.591±0.01	*-5.91*
w50N230a	3.436±0.19	10.84±0.31	68.30	0.586±0.03	0.264±0.10	**54.95**
w100N391a	2.666±0.06	8.065±0.04	66.94	0.455±0.02	0.257±0.01	43.52
w100N476a	6.145±0.07	18.94±2.03	67.56	0.592±0.06	0.561±0.09	5.24

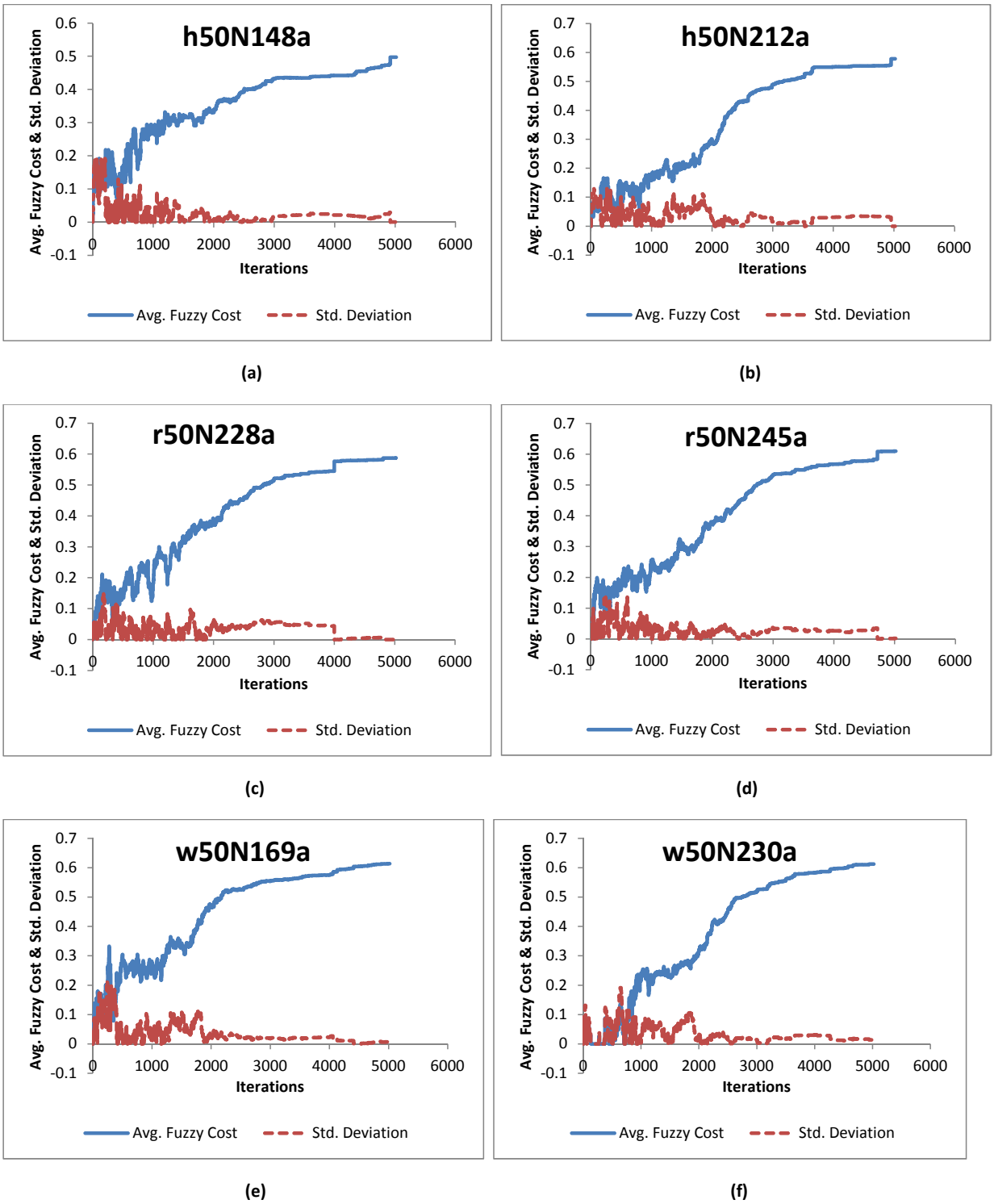


Figure 5.6: Average cost and associated standard deviation curves of FSA for all the test cases with 50 nodes with respect to FuzzyCF (maximization function); $\alpha = 0.96, \beta = 1.01$ and $M = 20$.

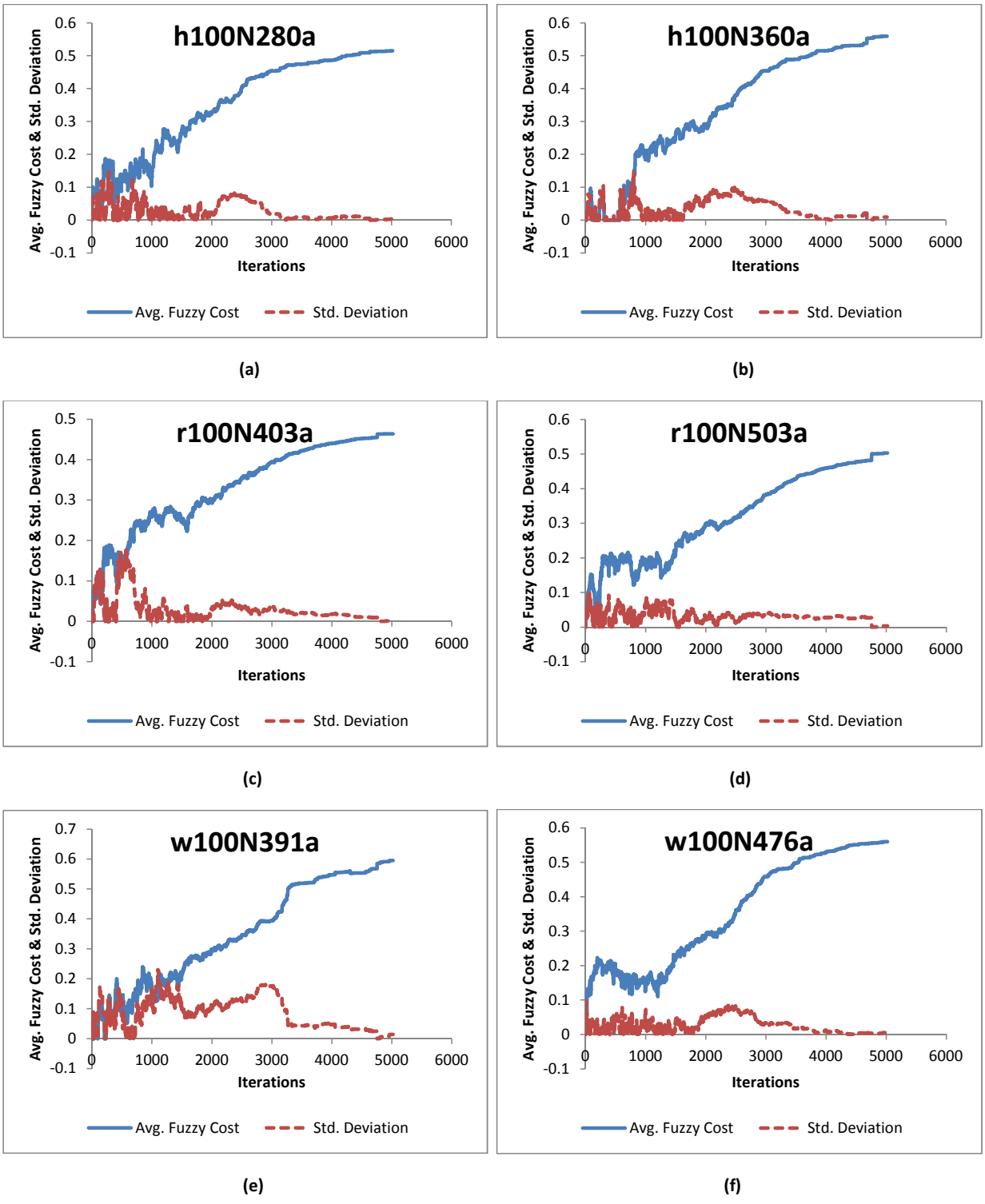


Figure 5.7: Average cost and associated standard deviation curves of FSA for all the test cases with 100 nodes with respect to FuzzyCF (maximization function); $\alpha = 0.96, \beta = 1.01$ and $M = 20$.

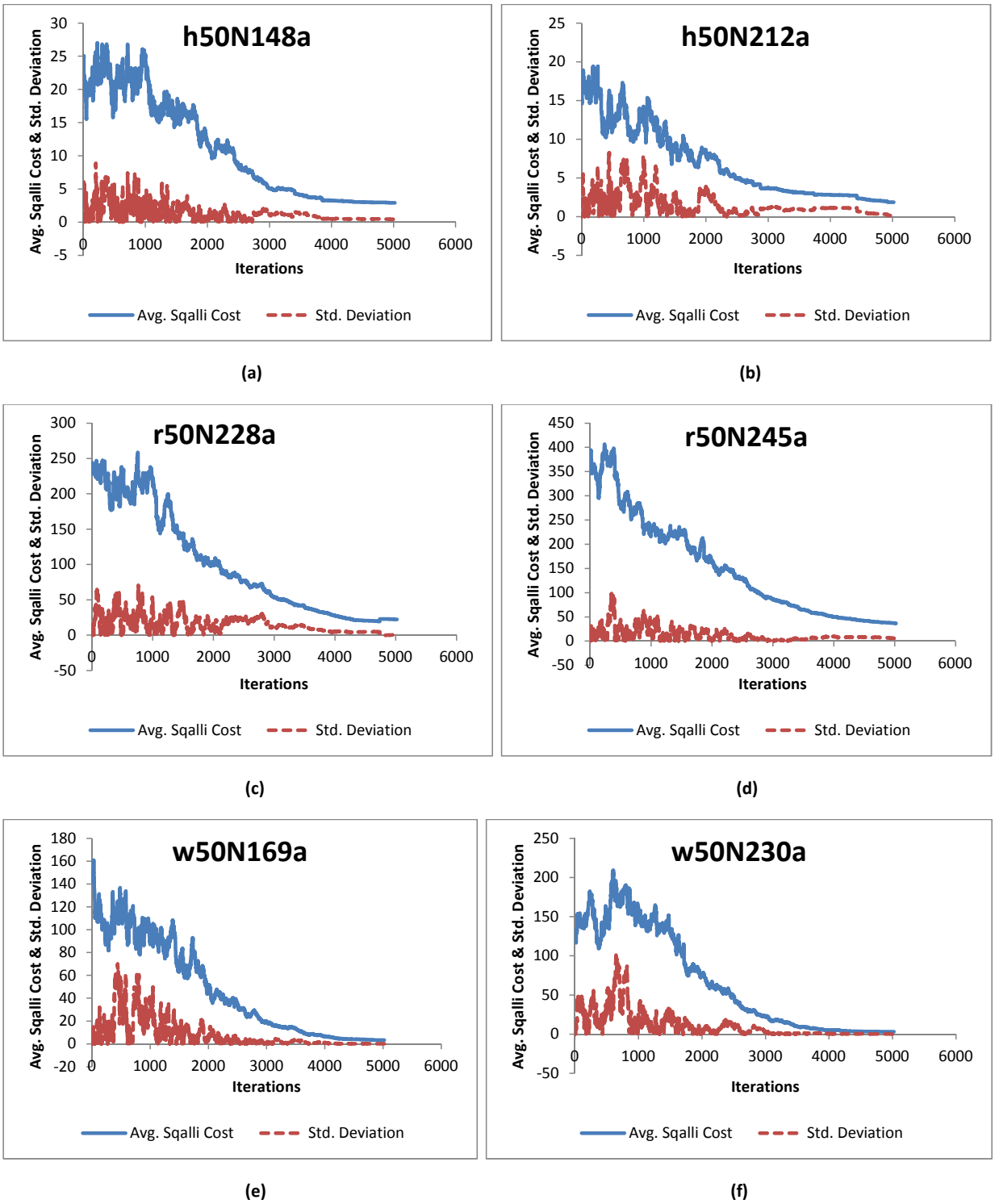


Figure 5.8: Average cost and associated standard deviation curves of SA for all the test cases with 50 nodes with respect to SqalliCF (minimization function); $\alpha = 0.965, \beta = 1.01$ and $M = 20$.

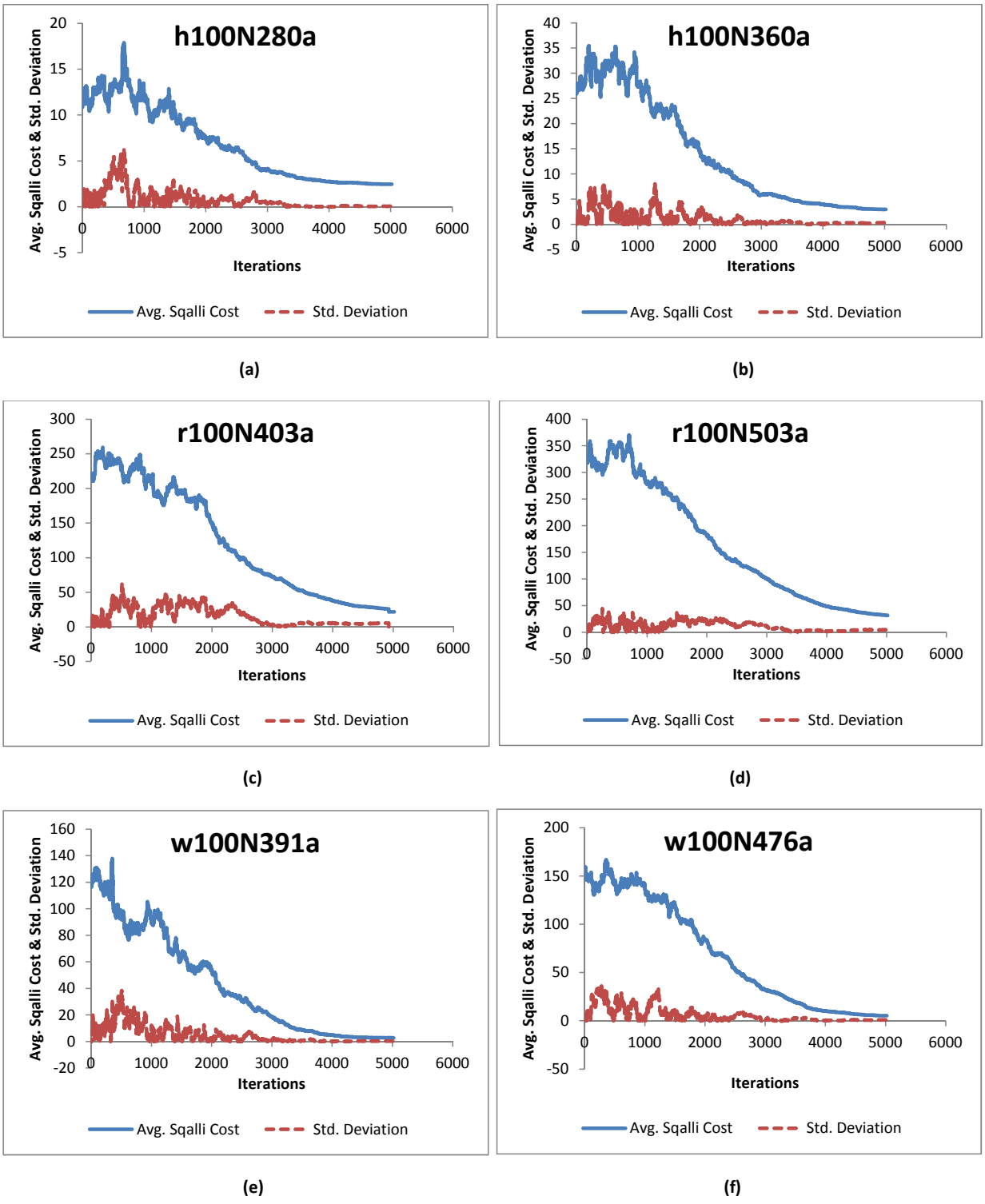


Figure 5.9: Average cost and associated standard deviation curves of SA for all the test cases with 100 nodes with respect to SqalliCF (minimization funtion); $\alpha = 0.965, \beta = 1.01$ and $M = 20$.

Table 5.4: MU, NOC, NUL, and average execution time corresponding to SqalliCF and FuzzyCF using FSA.

Test case	Traffic Demand (bytes)	SqalliCF				FuzzyCF			
		MU	NOC	NUL	Time (sec)	MU	NOC	NUL	Time (sec)
h100N280a	4605	1.36± 0.017	10.17± 2.422	13.13± 3.181	1177± 12.084	1.50± 0.352	8.93± 1.388	0.70± 3.313	859.1± 8.658
h100N360a	12407	1.66± 0.157	16.20± 3.067	16.93± 4.401	1091.1± 8.016	2.01± 0.531	21.03± 6.300	0.20± 0.484	796.5± 5.138
h50N148a	4928	1.40± 0.058	9.63± 2.251	3.07± 1.460	135.2± 1.095	1.51± 0.100	8.40± 1.670	0.00± 0.000	98.2± 3.097
h50N212a	3363	1.47± 0.277	5.63± 1.670	43.70± 5.977	145.7± 1.23	1.68± 0.080	5.17± 0.791	0.07± 0.254	102.1± 1.592
r100N403a	70000	1.93± 0.184	63.50± 5.124	3.00± 1.681	1335.5± 13.798	2.72± 0.576	62.60± 6.579	0.03± 0.183	1017.2± 13.792
r100N503a	100594	1.86± 0.143	83.60± 6.184	5.87± 2.713	1390.4± 14.209	3.58± 0.416	82.33± 26.212	6.57± 16.079	1069.8± 18.918
r50N228a	42281	1.58± 0.154	26.97± 3.347	5.80± 2.441	160.3± 1.466	2.02± 0.134	22.03± 2.251	0.03± 0.183	116.1± 1.408
r50N245a	53562	2.24± 0.276	41.83± 4.913	5.10± 2.056	159.1± 1.788	2.83± 0.239	28.77± 2.622	0.23± 0.504	117.2± 2.329
w100N391a	48474	1.44± 0.046	2.33± 1.647	4.80± 1.954	1355.5± 13.423	1.75± 0.708	42.10± 19.921	7.23± 6.185	1029.2± 22.928
w100N476a	63493	1.42± 0.043	23.83± 6.923	14.73± 3.433	1383.3± 13.392	2.24± 0.219	41.70± 15.647	4.73± 11.694	1048.2± 21.955
w50N169a	25411	1.27± 0.021	8.60± 1.792	2.93± 1.552	153.7± 1.552	1.41± 0.086	8.80± 1.808	0.00± 0.000	110.7± 1.78
w50N230a	39447	1.23± 0.017	4.90± 1.493	6.13± 2.674	158.6± 1.938	1.55± 0.151	14.40± 19.08	1.37± 4.537	113.8± 2.497

Table 5.5: Comparison of FuzzyCF and SqalliCF in percentage using SA. Superior performance of FuzzyCF is in boldface and inferior performance of FuzzyCF is in italics.

Test Case	MU % diff	p-value	NOC % diff	p-value	NUL % diff	p-value
h100N280a	*-10.29*	0.022	**12.19**	0.019	**94.67**	0.013
h100N360a	*-21.08*	0.001	*-29.81*	0.027	**98.82**	0.011
h50N148a	*-7.86*	0.031	**12.77**	0.019	**100.00**	0.010
h50N212a	*-14.29*	0.022	**8.17**	0.172	**99.84**	0.010
r100N403a	*-40.93*	0.003	**1.42**	0.557	**99.00**	0.027
r100N503a	*-92.47*	0.001	**1.52**	0.798	*-11.93*	0.815
r50N228a	*-27.85*	0.021	**18.32**	0.041	**99.48**	0.009
r50N245a	*-26.34*	0.034	**31.22**	0.035	**95.49**	0.020
w100N391a	*-21.53*	0.021	*-1706.87*	0.000	*-50.63*	0.044
w100N476a	*-57.75*	0.009	*-74.99*	0.008	**67.89**	0.032
w50N169a	*-11.02*	0.033	*-2.33*	0.669	**100.00**	0.010
w50N230a	*-26.02*	0.017	*-193.88*	0.001	**77.65**	0.000

FuzzyCF when compared to SqalliCF. The results in both tables are displayed with respect to the three design objectives, i.e. MU, NOC, and NUL.

From Table 5.4, it is observed that SqalliCF was able to generate lower levels of MU as compared to FuzzyCF for all test cases. More specifically, MU for SqalliCF was generally in the range 1.23 to 1.93, with the exception of r50N245a (having an MU level of 2.24). For FuzzyCF, MU was generally in the range 1.41 to 3.58. The difference between the MU levels of the two approaches were statistically evaluated for significance in terms of the percentage improvements as shown in Table 5.5. The results in the second column of table suggest that, for the objective MU, the results obtained by FuzzyCF were of inferior quality than those of SqalliCF.

With respect to the second objective (NOC), the results were somewhat mixed as shown in Table 5.4. For all the Waxman graphs as well as test case h100N360a, SqalliCF resulted in a lower number of congested links, while for the remaining test cases, the FuzzyCF cost function showed better performance. This was confirmed by the values in Table 5.5, which showed statistically better results for FuzzyCF than SqalliCF for four test cases (h100N280a, h50N148a, r50N228a, and r50N245a), while SqalliCF performed statistically significantly better than FuzzyCF for the other four cases (h100N360a, w100N391a, w100N476a, and w50N230a). For the remaining four cases, both FuzzyCF and SqalliCF had the same quality of results.

The third objective (i.e. NUL) is reflected in Table 5.4, highlighting the dominant trend that FuzzyCF performed significantly better than SqalliCF for over 80% (i.e. 10 out of 12) of the test cases. As seen in Table 5.5, the results obtained by FuzzyCF were significantly better than that of SqalliCF, except for test case w100N391a while for r100N503a, the results were of the same quality.

With regard to the execution time, it is observed from Table 5.4 that the average execution time per run for FuzzyCF was less than that of SqalliCF.

Based on the above observations and analysis, it can be suggested that overall, FuzzyCF performed better than SqalliCF.

5.4 Conclusion

This chapter presented the FSA algorithm for the OSPFWS problem. The fuzzy logic based objective function, FuzzyCF, was incorporated in the FSA algorithm. A sensitivity analysis of the parameters of FSA was performed, which indicated that $\alpha = 0.965$ and $M = 20$ produced the best results for FSA. FSA was compared with another SA algorithm based on the SqalliCF cost function. The results indicated that:

- for the MU objective, FuzzyCF produced results of inferior quality as compared to that of SqalliCF.
- for the NOC objective, FuzzyCF performed statistically significantly better than SqalliCF for 4 out of 12 test cases. FuzzyCF performance was comparable to SqalliCF for 4 out of 12 test cases.
- for the NUL objective, FuzzyCF performed better than SqalliCF for 10 out of 12 test cases.

Furthermore, archiving of solutions is implemented in FSA to obtain a set of non-dominated solutions. This archive of solutions will be used in Chapter 8 to compare the performance of FSA with other population-based algorithms.

Chapter 6

Fuzzy Simulated Evolution Algorithm for the OSPFWS Problem

This chapter presents the multi-objective FSimE algorithm for the OSPFWS problem. The fuzzy cost function, FuzzyCF, described in Chapter 4 has been used as the underlying objective function. The results are compared with a SimE algorithm employing the SqalliCF cost function.

The outline of the chapter is as follows: Section 6.1 provides implementation details of the FSimE algorithm. Section 6.2 provides the experimental setup used for the FSimE simulations. Section 6.3 discusses the effect of the bias parameter,

B, on the quality of solutions with respect to FuzzyCF. Section 6.3 also provides a comparison of the SqalliCF and FuzzyCF cost functions using the SimE algorithm.

6.1 Fuzzy Simulated Evolution Algorithm

A detailed algorithmic description of SimE was given in Section 2.3.2. The SimE algorithm updates a single solution, but unlike SA, navigates the search space by employing compound moves (takes more than one move per iteration) as shown in Figure 6.1, which corresponds to the topology given in Figure 4.1. The underlined numbers are the new weights on their respective links of the network. The algorithm repetitively iterates between the evaluation, selection, and allocation steps before a pre-defined stopping criterion is reached. Furthermore, the concept of *archiving* (as explained in Section 5.2) is also applied to FSimE. Similar to FSA, FSimE also yielded an archive of non-dominated solutions. The implementation of the main steps of FSimE are described below:

Evaluation

For the OSPFWS problem, each element is a weight on a link, for which goodness needs to be evaluated. The goodness function is one of the key factors that affects the performance of the FSimE algorithm, and should therefore be carefully designed. This thesis uses the goodness function defined by Sqalli *et al.* [149] for both cost functions (SqalliCF and FuzzyCF). The goodness function is defined as

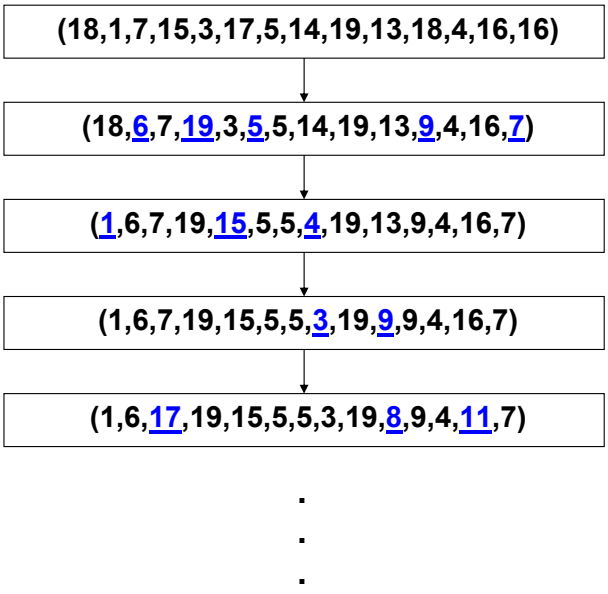


Figure 6.1: Sequence of moves in FSimE within the allocate function.

$$g_i = \begin{cases} 1 - u_i & \text{for } MU \leq 1 \\ 1 - u_i/MU + u_i/MU^2 & \text{for } MU > 1 \end{cases} \tag{6.1}$$

where u_i represents the utilization on link i and MU refers to the maximum utilization. To illustrate how the above goodness function works, examine the following example: consider two links e_1 and e_2 of a particular solution. Let the maximum utilization be 0.9. Assume that the utilizations on links e_1 and e_2 are 0.6 and 0.1, respectively. By substituting the above values in the goodness function in Equation (6.1), $g_{e_1} = 0.4$ and $g_{e_2} = 0.9$ are obtained. In the next iteration, e_1 is more probable to be selected for replacement than e_2. This is because the goodness of e_1 is worse than that of e_2.

Now, consider the maximum utilization to be 1.8, and let the utilization on links e_1 and e_2 be 0.7 and 1.4, respectively. From Equation (6.1), $g_{e_1} = 0.827$ and $g_{e_2} = 0.655$. In the next iteration, e_2 is more probable to be selected for replacement than e_1.

Selection

In the selection phase, for each link, i, a random number is sampled from a uniform distribution in the range [0,1]. If this random number is larger than $g_i + B$, the corresponding weight is selected for allocation.

The bias B is used to control the size of the set of selected weights. A low value of B increases the number of elements selected in each iteration, thus allowing the algorithm to explore more. This may lead to high quality solutions, but at the expense of higher computational effort. A high value of B inflates the goodness of each element. This may result in a reduced number of elements selected for re-allocation. Consequently, the execution time of the algorithm is reduced, but at the risk of premature convergence to a sub-optimal (or local optimal) solution.

Since it is computationally expensive to find the best bias value by a process of trial-and-error, different approaches have been proposed in the literature [84, 148] to use a dynamic bias, instead of a user-defined static bias. The approach by Sait *et al.* [148] calculates the bias based on the quality of the current solution and changes in every iteration. The corresponding dynamic bias is given by

$$B(t) = 1 - G(t) \tag{6.2}$$

where $B(t)$ is the bias in iteration t and $G(t)$ is the average goodness of all the elements at the start of that iteration. The average goodness of elements is a measure of how many "good" elements are present in the solution. For a detailed analysis of the dynamic bias, refer to Sait *et al.* [148].

Allocation

During the allocation step of the algorithm, the selected weights are removed from the solution one at a time. For each removed weight, new weights are tried in such a way that they result in an overall better solution. The following allocation scheme was adopted: For all iterations, a weight window of size 4 is kept. Therefore, if weight 6 was selected for replacement, then weight values 4, 5, 7, and 8 are tried. This results in a beam search and is done to have less disturbance in the solutions in each iteration.

Figures 6.2 and 6.3 illustrate a typical behavior of the proposed FSimE algorithm. These figures respectively plot the average goodness of weights and the cardinality of the selection set for the test case r100N503a as an example. It is observed in Figure 6.2 that the average goodness increases with time. In the initial stages of the search, the goodness increases significantly. However, as the search progresses, the average goodness increases slowly. Figure 6.3 illustrates the cardinality of the selection set. It is observed that the number of selected elements (i.e., weights) decreases as the number of iterations increases. This suggests that, on average, the weights have been

assigned to their optimal values, and the algorithm reached a level of convergence. Therefore, fewer elements (i.e., weights) are selected for perturbation. If the two trends in Figures 6.2 and 6.3 are considered together, the general observation is that the convergence of the FSimE algorithm towards a better solution is correlated with the average goodness of links. The increase in average goodness with a decrease in the selection set size indicates that most of the links are in their optimal or near optimal positions and therefore less moves are made. This relationship between average goodness and selection set size is also illustrated in Figure 6.4 for the test case r100N503a. When the average goodness of weights is low, the selection set size is big and vice versa. Thus bad elements (i.e. weights) were differentiated from good elements and were subjected to be changed with better weights.

6.2 Experimental setup

For each test case, 30 independent runs were executed for 100 iterations and the average of the best solutions found in each run was recorded, together with the standard deviation. Static bias values of -0.02, -0.1, 0, 0.1, 0.2, 0.3 and a dynamic bias (see Equation (6.2)) were used for analysis with respect to FuzzyCF. For the SqalliCF cost function, a bias value of -0.02 was used as reported by Sait *et al.* [149]. The Wilcoxon rank-sum test [65] was used to validate the significance of the results. A confidence level of 95% was used.

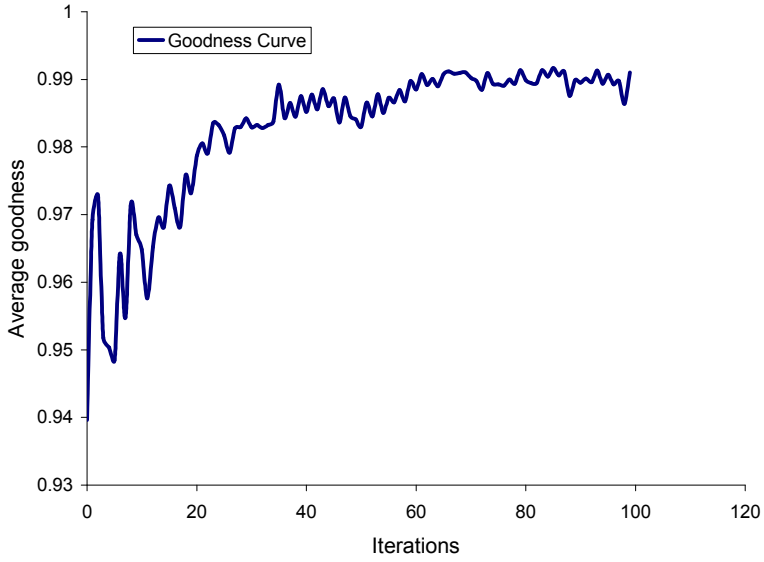


Figure 6.2: Average goodness of weights for test case r100N503a.

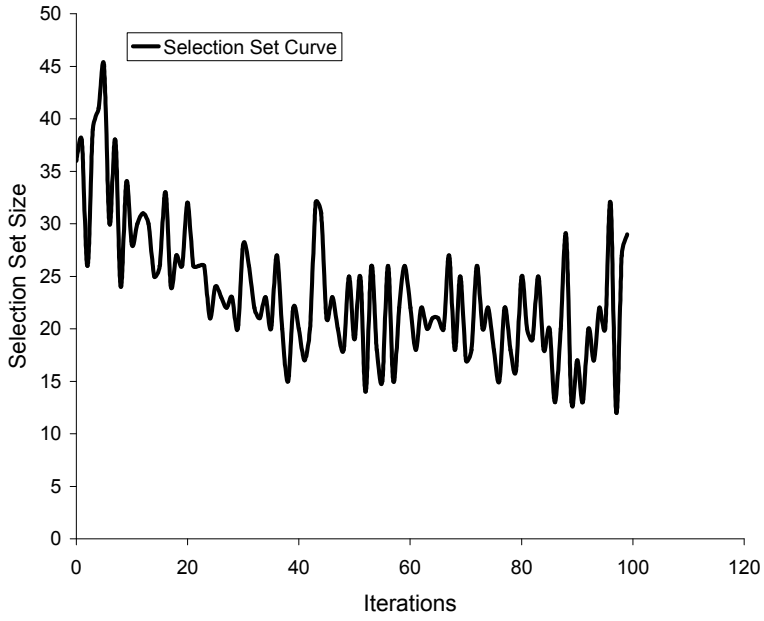


Figure 6.3: Selection set size for test case r100N503a.

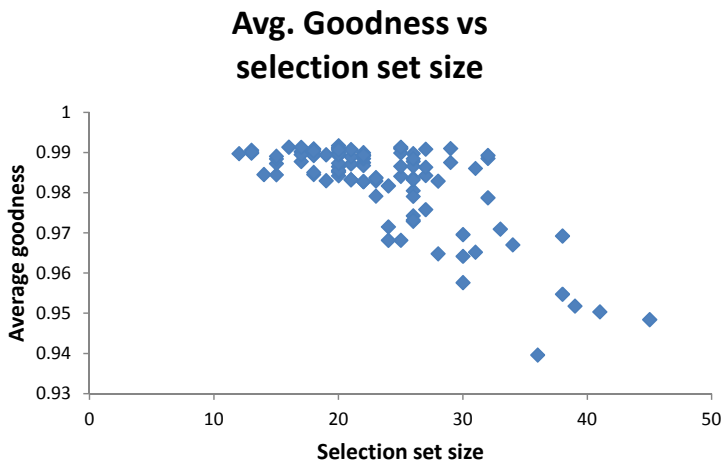


Figure 6.4: Average goodness vs selection set size for test case r100N503a.

6.3 Results and Discussion

The purpose of this section is to discuss the effect of the bias parameter, B, on the quality of solutions with respect to FuzzyCF. This section also provides a comparison of the SqalliCF and FuzzyCF cost functions using the SimE algorithm.

The average cost over 30 independent runs for each bias value is shown in Tables 6.1, 6.2 and 6.3 for the 12 test cases (details of the test cases are presented in Section 4.3). Graphic illustrations of the search progress for each experimented value of the bias with respect to all test cases are shown in Figures 6.5, 6.6, 6.7 and 6.8. It is observed that a bias value of -0.1 produced the best results (i.e. having the highest value for FuzzyCF) for all the test cases. Furthermore, a dynamic bias was not able to produce high quality results. Therefore, for all experiments involving FSimE with FuzzyCF, a bias value of -0.1 was used.

The average value of MU for all test cases was lower for SqalliCF than that of

Table 6.1: Effect of bias value on the quality of solution for the test cases h50N148a, h50N212a, h100N280a and h100N360a. Statistically significant improvement achieved by value -0.1 over other values is in boldface. NA = Not Applicable (The value -0.1 is used to calculate % improvement over other values).

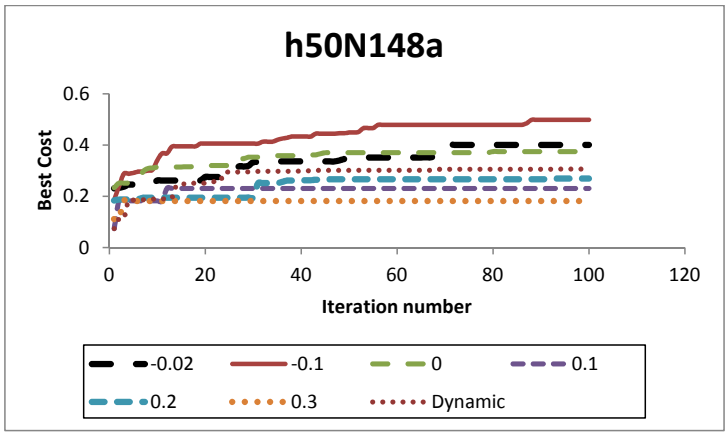
Test Case	Bias Value	Avg. Cost	% Impr.
h50N148a	-0.02	0.4±0.151	19.679
	-0.1	0.498±0.078	NA
	0	0.374±0.038	24.9
	0.1	0.23±0.076	53.815
	0.2	0.268±0.062	46.185
	0.3	0.181±0.089	**63.655**
	Dynamic	0.306±0.027	38.554
h50N212a	-0.02	0.415±0.017	10.944
	-0.1	0.466±0.144	NA
	0	0.395±0.015	15.236
	0.1	0.287±0.045	38.412
	0.2	0.266±0.093	42.918
	0.3	0.22±0.045	**52.79**
	Dynamic	0.293±0.096	37.124
h100N280a	-0.02	0.2±0.003	64.789
	-0.1	0.568±0.117	NA
	0	0.345±0.028	39.261
	0.1	0.369±0.001	35.035
	0.2	0.262±0.088	53.873
	0.3	0.203±0.076	64.261
	Dynamic	0.186±0.04	**67.254**
h100N360a	-0.02	0.37±0.064	27.875
	-0.1	0.513±0.162	NA
	0	0.429±0.037	16.374
	0.1	0.348±0.044	32.164
	0.2	0.293±0.031	42.885
	0.3	0.166±0.061	**67.641**
	Dynamic	0.357±0.074	30.409

Table 6.2: Effect of bias value on the quality of solution for the test cases r50N228a, r50N245a, r100N403a and r100N503a. Statistically significant improvement achieved by value -0.1 over other values is in boldface. NA = Not Applicable (The value -0.1 is used to calculate % improvement over other values).

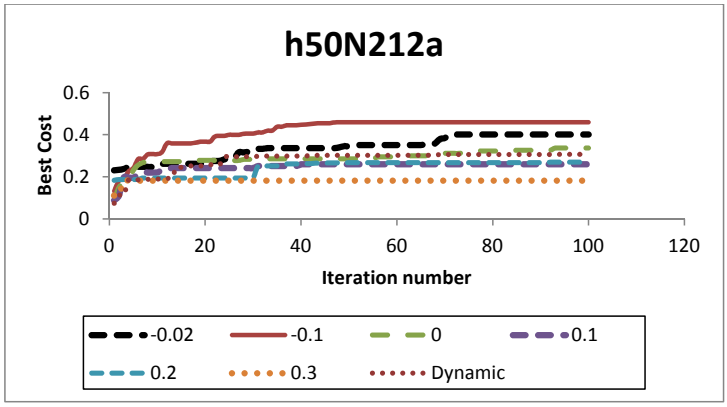
Test Case	Bias Value	Avg. Cost	% Impr.
r50N228a	-0.02	0.538±0.118	11.658
	-0.1	0.609±0.134	NA
	0	0.276±0.058	**54.68**
	0.1	0.493±0.015	19.048
	0.2	0.45±0.091	26.108
	0.3	0.382±0.099	37.274
	Dynamic	0.279±0.029	54.187
r50N245a	-0.02	0.298±0.098	51.702
	-0.1	0.617±0.132	NA
	0	0.572±0.08	7.293
	0.1	0.29±0.092	**52.998**
	0.2	0.474±0.098	23.177
	0.3	0.336±0.085	45.543
	Dynamic	0.508±0.198	17.666
r100N403a	-0.02	0.3±0.037	47.826
	-0.1	0.575±0.03	NA
	0	0.206±0.053	64.174
	0.1	0.258±0.081	55.13
	0.2	0.24±0.019	58.261
	0.3	0.1666±0.028	**71.026**
	Dynamic	0.24±0.033	58.261
r100N503a	-0.02	0.3±0.082	52.607
	-0.1	0.633±0.101	NA
	0	0.3±0.058	52.607
	0.1	0.494±0.058	21.959
	0.2	0.471±0.103	25.592
	0.3	0.293±0.074	**53.712**
	Dynamic	0.492±0.111	22.275

Table 6.3: Effect of bias value on the quality of solution for the test cases w50N169a, w50N230a, w100N391a and w100N476a. Statistically significant improvement achieved by value -0.1 over other values is in boldface. NA = Not Applicable (The value -0.1 is used to calculate % improvement over other values).

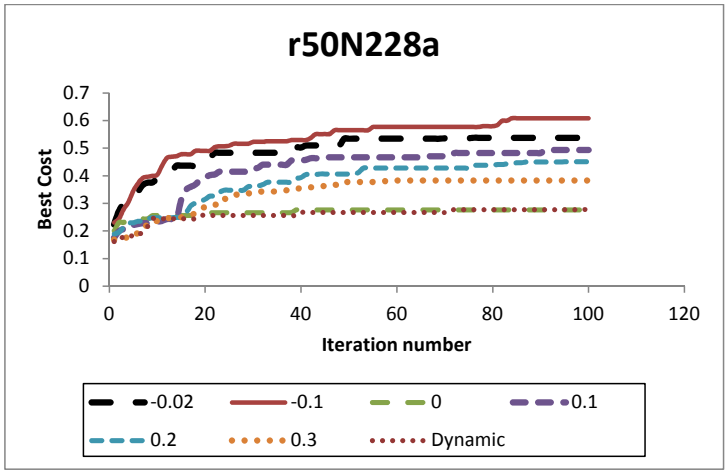
Test Case	Bias Value	Avg. Cost	% Impr.
w50N169a	-0.02	0.61±0.078	7.855
	-0.1	0.662±0.093	NA
	0	0.547±0.065	17.372
	0.1	0.478±0.037	27.795
	0.2	0.453±0.051	31.571
	0.3	0.352±0.051	**46.828**
	Dynamic	0.491±0.065	25.831
w50N230a	-0.02	0.302±0.089	57.284
	-0.1	0.707±0.031	NA
	0	0.276±0.098	**60.962**
	0.1	0.457±0.061	35.361
	0.2	0.39±0.089	44.837
	0.3	0.335±0.076	52.617
	Dynamic	0.493±0.098	30.269
w100N391a	-0.02	0.333±0.076	54.57
	-0.1	0.733±0.068	NA
	0	0.638±0.005	12.96
	0.1	0.61±0.095	16.78
	0.2	0.55±0.012	24.966
	0.3	0.25±0.071	**65.894**
	Dynamic	0.562±0.082	23.329
w100N476a	-0.02	0.652±0.093	14.548
	-0.1	0.763±0.091	NA
	0	0.611±0.026	19.921
	0.1	0.507±0.052	33.552
	0.2	0.504±0.073	33.945
	0.3	0.348±0.034	**54.391**
	Dynamic	0.542±0.096	28.965



(a)

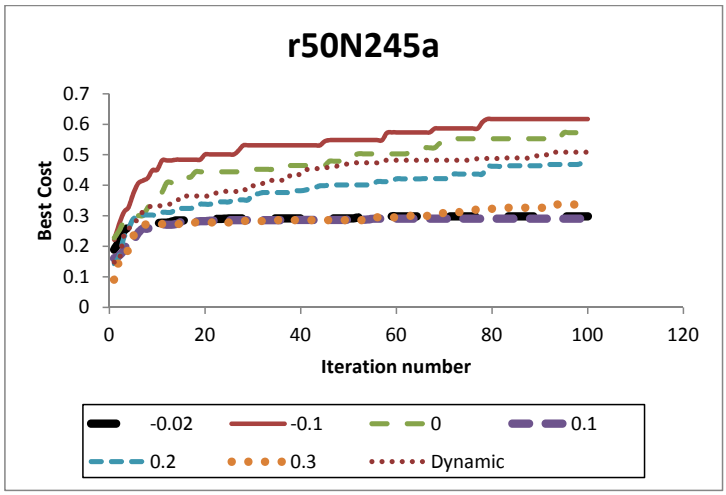


(b)

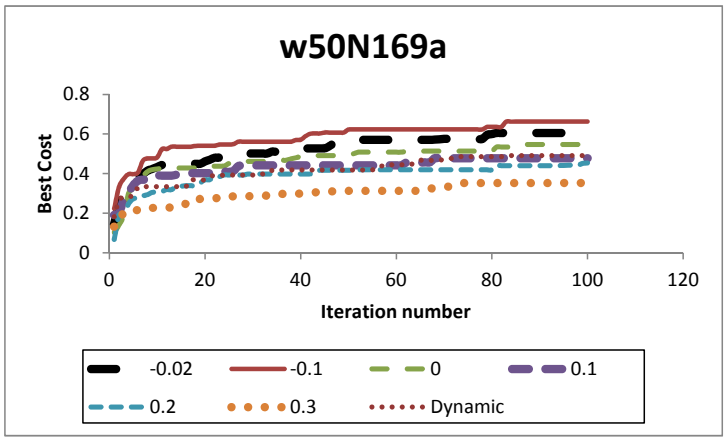


(c)

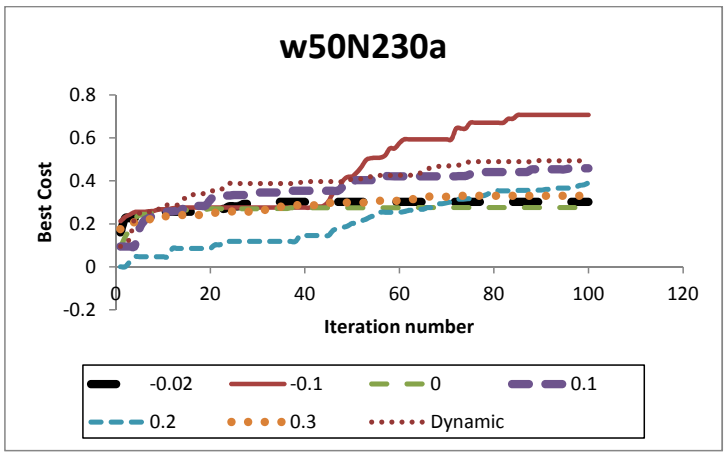
Figure 6.5: Plots of FSimE average best cost for all the bias values with respect to the test cases (a) h50N148a (b) h50N212a and (c) r50N228a. Search with bias value -0.1 produced better results.



(a)

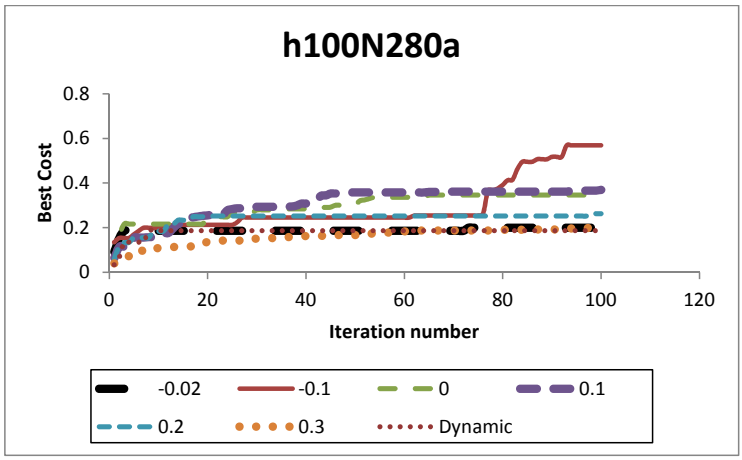


(b)

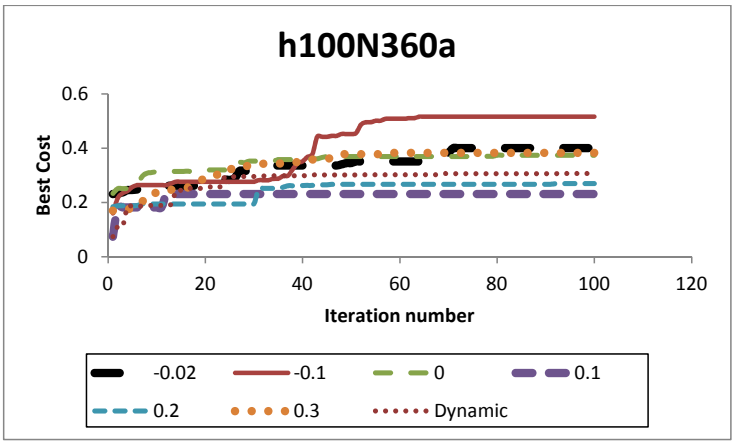


(c)

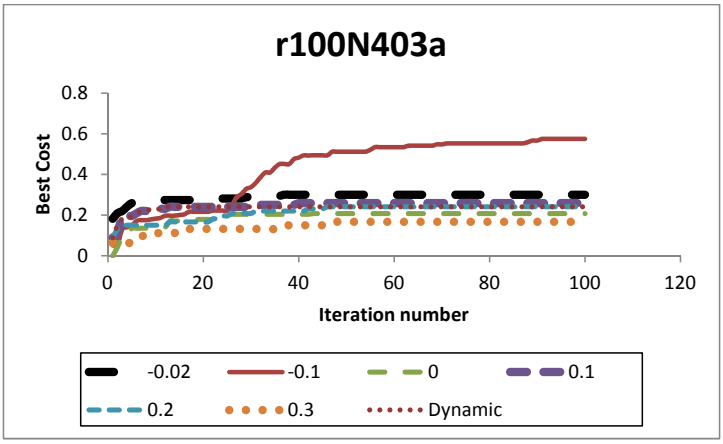
Figure 6.6: Plots of FSimE average best cost for all the bias values with respect to the test cases (a) r50N245a (b) w50N169a and (c) w50N230a. Search with bias value -0.1 produced better results.



(a)

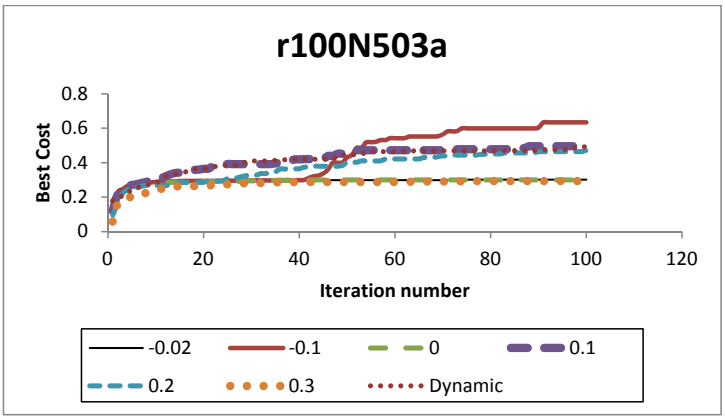


(b)

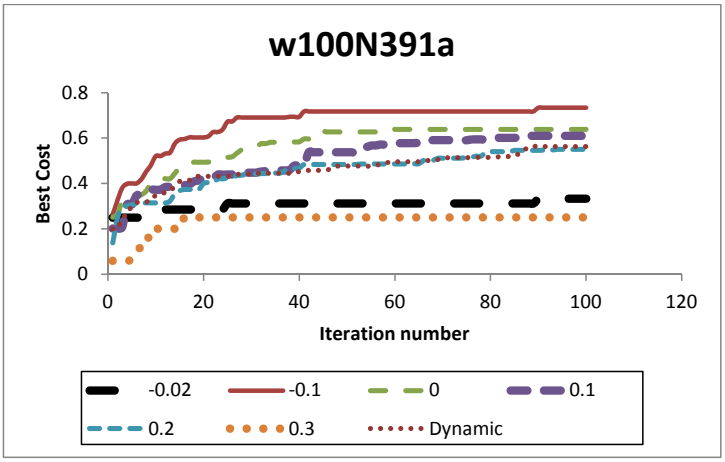


(c)

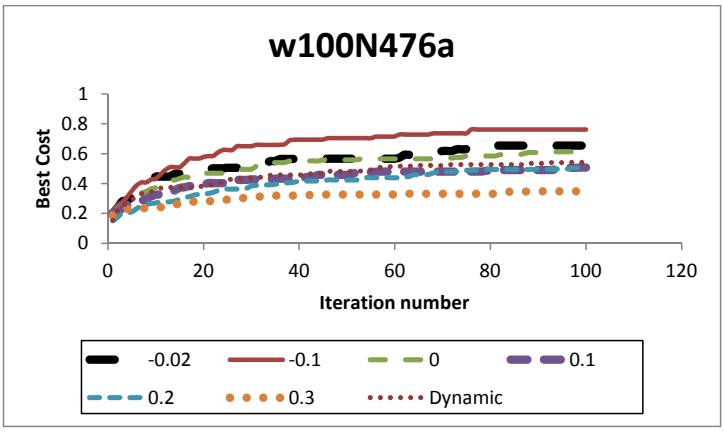
Figure 6.7: Plots of FSimE average best cost for all the bias values with respect to the test cases (a) h100N280a (b) h100N360a and (c) r100N403a. Search with bias value -0.1 produced better results.



(a)



(b)



(c)

Figure 6.8: Plots of FSimE average best cost for all the bias values with respect to the test cases (a) r100N503a (b) w100N391a and (c) w100N476a. Search with bias value -0.1 produced better results.

Table 6.4: MU, NOC, NUL, and average execution time corresponding to two cost functions using SimE.

Test case	Traffic Demand (bytes)	SqalliCF				FuzzyCF			
		MU	NOC	NUL	Time (sec)	MU	NOC	NUL	Time (sec)
h100N280a	4605	1.34± 0.000	10.73± 2.449	21.33± 3.078	1220.4	1.41± 0.058	8.50± 1.432	1.23± 0.898	875.9
h100N360a	12407	1.42± 0.088	15.63± 2.189	17.67± 3.708	1189.6	1.72± 0.092	16.13± 2.240	0.93± 1.015	826.6
h50N148a	4928	1.42± 0.051	10.93± 2.518	3.67± 1.668	148.1	1.62± 0.119	10.37± 2.025	0.07± 0.254	128.6
h50N212a	3363	1.32± 0.096	7.60± 1.714	46.07± 5.705	151	1.67± 0.102	4.93± 0.740	2.77± 1.924	137.5
r100N403a	70000	1.44± 0.064	53.13± 4.562	0.83± 0.648	1597.9	1.86± 0.074	44.73± 2.490	0.13± 0.346	1588.5
r100N503a	100594	1.41± 0.059	37.77± 5.283	1.63± 0.890	1479.2	2.19± 0.166	52.80± 2.809	0.17± 0.379	1334.1
r50N228a	42281	1.36± 0.054	24.83± 3.130	2.20± 1.808	160	1.80± 0.121	19.77± 1.305	0.10± 0.305	152.2
r50N245a	53562	2.14± 0.182	39.20± 3.872	2.03± 0.999	167.2	2.60± 0.193	26.20± 1.972	0.80± 0.887	114.4
w100N391a	48474	1.41± 0.006	1.10± 0.305	5.03± 1.829	1371	1.42± 0.033	7.17± 2.692	0.03± 0.183	1132.3
w100N476a	63493	1.32± 0.003	7.07± 1.143	12.63± 1.712	1374.6	1.46± 0.070	17.07± 2.363	0.60± 0.814	1174.9
w50N169a	25411	1.26± 0.016	9.37± 2.205	2.80± 1.375	178.4	1.44± 0.078	8.37± 1.189	0.03± 0.183	134.8
w50N230a	39447	1.23± 0.007	3.67± 0.959	5.27± 1.741	189.1	1.44± 0.088	9.13± 1.570	0.27± 0.583	112.8

Table 6.5: Comparison of FuzzyCF and SqalliCF in percentage using SimE. Superior performance of FuzzyCF is in boldface and inferior performance of FuzzyCF is in italics.

Test Case	MU % diff	p-value	NOC % diff	p-value	NUL % diff	p-value
h100N280a	*-5.22*	0.032	**20.78**	0.040	**94.23**	0.019
h100N360a	*-21.13*	0.027	*-3.20*	0.385	**94.74**	0.018
h50N148a	*-14.08*	0.020	**5.12**	0.341	**98.09**	0.020
h50N212a	*-26.52*	0.022	**35.13**	0.033	**93.99**	0.011
r100N403a	*-29.17*	0.020	**15.81**	0.031	**84.34**	0.010
r100N503a	*-55.32*	0.016	*-39.79*	0.027	**89.57**	0.012
r50N228a	*-32.35*	0.019	**20.38**	0.019	**95.45**	0.034
r50N245a	*-21.50*	0.011	**33.16**	0.020	**60.59**	0.023
w100N391a	*-0.71*	0.045	*-551.82*	0.000	**99.40**	0.011
w100N476a	*-10.61*	0.022	*-141.44*	0.001	**95.25**	0.025
w50N169a	*-14.29*	0.020	**10.67**	0.033	**98.93**	0.019
w50N230a	*-17.07*	0.015	*-148.77*	0.001	**94.88**	0.021

FuzzyCF. From the third column of Table 6.4, note that MU using SqalliCF was in the range 1.23 to 1.44, with the exception of r50N245a (with MU level of 2.14). For FuzzyCF, the MU was generally in the range of 1.41 to 1.72; with the exception of r100N503a and r50N245a, having an MU of 2.19 and 2.60, respectively. The outcome of a statistical test on the percentage improvement (second column of Table 6.5) suggests that FuzzyCF produced inferior results compared to SqalliCF for all test cases.

For the NOC objective, the results in Table 6.4 show superior performance by FuzzyCF for seven cases out of which six are statistically significant as shown in Table 6.5. For two test cases (h100N360a and h50N148a), the results were of the same

quality. Furthermore, SqalliCF showed statistically significantly better performance for the remaining cases (r100N503a, w100N391a, w100N476a, and w50N230a).

For the NUL objective, FuzzyCF had a lower number of unutilized links (which is desired) than SqalliCF for all test cases, as depicted in Table 6.4. As suggested by the values in Table 6.5, all results obtained by FuzzyCF were significantly better than SqalliCF.

As far as execution time is concerned, the results depict more or less the same trend as observed for FSA. Table 6.4 shows that the average execution time per run for FuzzyCF was less than that of SqalliCF.

In view of the above results and analysis, the overall assessment is that FuzzyCF performed better than SqalliCF.

6.4 Conclusion

This chapter proposed and investigated a fuzzy multi-objective algorithm based on the SimE algorithm developed to solve the OSPFWS problem. The proposed algorithm utilizes the fuzzy logic based objective function, namely, FuzzyCF. The FSimE algorithm was compared with a SimE algorithm that uses the SqalliCF cost function. An empirical analysis showed that the performance of FuzzyCF was:

- worse than SqalliCF with respect to MU for all test cases;

- better than SqalliCF for the majority of test cases with respect to NOC; and
- better than SqalliCF for all test cases with respect to NUL.

Furthermore, archiving of solutions is implemented in FSimE to get a set of non-dominated solutions. This archive set of solutions will be used in Chapter 8 to compare the performance of FSimE with other population-based algorithms.

Chapter 7

Particle Swarm Optimization Algorithm Variants and NSGA-II for the OSPFWS Problem

This chapter presents the PSO algorithm variants to solve the OSPFWS problem. These algorithm variants are FPSO, FEPSO, WAPSO, and PDPSO. The fuzzy cost function (denoted as FuzzyCF) described in Chapter 4 is used as the underlying objective function for FPSO and FEPSO. The weighted sum (or aggregation) method described in Section 2.1.1 is used as the cost function for WAPSO, while the PDPSO algorithm uses the non-dominance criterion to solve the OSPFWS problem. The NSGA-II algorithm is then discussed. Control parameters are optimized for all the

algorithms presented in this chapter.

The outline of the chapter is as follows: Section 7.1 provides the implementation details of the FPSO algorithm. FEPSO is discussed in Section 7.2. The chapter is then followed by a discussion of WAPSO in Section 7.3. Section 7.4 describes the PDPSO algorithm for the OSPFWS problem. Lastly, NSGA-II is discussed in Section 7.5.

7.1 Fuzzy Particle Swarm Optimization Algorithm

The purpose of this section is to describe the implementation details of the FPSO algorithm. Section 7.1.1 discusses the particle position and velocity representation for the OSPFWS problem. The velocity update step of FPSO is discussed in Section 7.1.2. Lastly, the particle position update of the FPSO implementation is given in Section 7.1.3, followed by control parameter tuning in Section 7.1.4.

The original PSO algorithm was discussed in Chapter 2. Just like standard PSO, FPSO navigates the search space by maintaining a swarm of candidate solutions, with each candidate solution referred to as a *particle*. Each particle explores new positions in the search space through its own history of traversing the search space, and from the experience of other particles in the particle's neighborhood. With respect to the OSPFWS problem, each particle reaches a new candidate solution by changing a few weights on the links of the network. As with the basic PSO, the guidance in changing

these weights is provided by the particle's current position, its own best position so far, and the global best position obtained so far by the entire algorithm. Each step of the proposed FPSO algorithm is discussed in the following sub-sections in detail with the aid of examples.

7.1.1 Particle Position and Velocity Representation

The standard PSO uses floating-point vectors to represent positions and velocities. For the OSPFWS problem the solution representation is a set of weights on the network links. Therefore, this study uses a fixed size set representation for particles. The size of each set is equal to the number of links in the network. Therefore, for an arbitrary network, each particle position is defined as a set

$$\mathbf{X}_i(t) = \{\omega_{ab} : (a, b) \in E\}$$

where ω_{ab} is the weight assigned to the link between nodes a and b in the network, and E is a set of edges. $W = |\mathbf{X}_i(t)|$ is the number of weights in the solution, which is also equal to the number of links in the network. The velocity of particle i is represented as

$$\mathbf{V}_i(t) = \{(\omega \Rightarrow \omega')_{ab}\}$$

which represents a sequence of replacement operators, where the weight ω of link (a, b) is replaced with a new value, ω', and $|\mathbf{V}_i(t)|$ gives the total number of changes

to particle i subject to the constraint, $0 \leq |\mathbf{V}_i(t)| \leq W$.

Example 1: Consider the topology given in Figure 4.1. Note that the total number of links is 14. The assigned weights in this figure represents a possible configuration at time t, whereas the configuration represents a solution (i.e. a particle). A solution for this topology can be $(18, 1, 7, 15, 3, 17,\ 5, 14, 19, 13, 18, 4, 16, 16)$. This current solution is represented as

$\mathbf{X}_i(t) =$

$\{18_{AB}, 1_{AF}, 7_{BC}, 15_{BD}, 3_{CE}, 17_{CF}, 5_{DA}, 14_{EA}, 19_{EG}, 13_{FB}, 18_{FD}, 4_{FG}, 16_{GB}, 16_{GD}\}$

Also assume that, at time t, $\mathbf{V}_i(t) = \{(19 \Rightarrow 18)_{AB}, (2 \Rightarrow 1)_{AF}, (4 \Rightarrow 7)_{BC}, (12 \Rightarrow 15)_{BD}, (4 \Rightarrow 3)_{CE}, (15 \Rightarrow 17)_{CF}, (6 \Rightarrow 5)_{DA}, (12 \Rightarrow 14)_{EA}, (13 \Rightarrow 19)_{EG}, (10 \Rightarrow 13)_{FB}, (11 \Rightarrow 18)_{FD}, (9 \Rightarrow 4)_{FG}, (17 \Rightarrow 16)_{GB}, (17 \Rightarrow 16)_{GD}\}$. That is, the above solution, $\mathbf{X}_i(t)$, was obtained by replacing weight 19 on link AB with a weight of 18, weight 2 on link AF was replaced with a weight of 1, and so on. The particle $\mathbf{X}_i(t)$ is then updated in subsequent steps as discussed in the following sub-sections.

7.1.2 Velocity Update

The velocity of particle i is updated using

$$\mathbf{V}_i(t+1) = w \otimes \mathbf{V}_i(t) \oplus c_1 r_1(t) \otimes [\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)] \oplus c_2 r_2(t) \otimes [\mathbf{P}_g(t) \oslash \mathbf{X}_i(t)] \quad (7.1)$$

where $\mathbf{P}_i(t)$ represents the particle's own best position, and $\mathbf{P}_g(t)$ represents the global best position. It is noted that the sets $\mathbf{X}_i(t)$, $\mathbf{P}_i(t)$ and $\mathbf{P}_g(t)$ have the same set of links for a given network topology.

Equation (7.1) uses the operator $\otimes$ in three terms namely in the calculation of the inertia component, the cognitive component, and the social component. For these three terms, the same constant is multiplied with a set, generally expressed as $c \otimes \mathbf{X}$, where c is a constant. The operator is implemented by randomly selecting $\lfloor c \times |\mathbf{X}| \rfloor$ elements from $\mathbf{X}$ as the result of the operator. For the different terms of Equation (7.1), this means that:

1. Inertia component: $\lfloor w \times |\mathbf{V}_i(t)| \rfloor$ elements are randomly selected from $\mathbf{V}_i(t)$. If $\lfloor w \times |\mathbf{V}_i(t)| \rfloor > |\mathbf{V}_i(t)|$, then all the elements from $\mathbf{V}_i(t)$ are selected.

2. Cognitive component: $\lfloor c_1 r_1(t) \times |\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)| \rfloor$ elements are randomly selected from the set $\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)$. $r_1(t)$ is a random value in the range $[0, 1]$ sampled from a uniform distribution. The operator $\oslash$ is a "replacement" operator, which replaces weights on the links in $\mathbf{X}_i(t)$ by the weights on the corresponding links in $\mathbf{P}_i(t)$. If $\lfloor c_1 r_1(t) \times |\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)| \rfloor > |\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)|$, then all the elements from the set $\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)$ are selected.

3. Social component: $\lfloor c_2 r_2(t) \times |\mathbf{P}_g(t) \oslash \mathbf{X}_i(t)| \rfloor$ elements are randomly selected from the set $\mathbf{P}_g(t) \oslash \mathbf{X}_i(t)$. $r_2(t)$ is a random value in the range $[0, 1]$ sampled

from a uniform distribution. The operator $\oslash$ replaces weights on the links in $\mathbf{X}_i(t)$ by the weights on the corresponding links in $\mathbf{P}_g(t)$. If $\lfloor c_2 r_2(t) \times |\mathbf{P}_g(t) \oslash \mathbf{X}_i(t)| \rfloor > |\mathbf{P}_g(t) \oslash \mathbf{X}_i(t)|$, then all the elements from the set $\mathbf{P}_g(t) \oslash \mathbf{X}_i(t)$ are selected.

The operator $\oplus$ implements the set addition (union) operator.

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). Further assume that the best fitness so far for particle i was generated by the position:

$\mathbf{P}_i(t) =$

$\{18_{AB}, 12_{AF}, 7_{BC}, 15_{BD}, 3_{CE}, 16_{CF}, 5_{DA}, 13_{EA}, 19_{EG}, 13_{FB}, 8_{FD}, 4_{FG}, 12_{GB}, 16_{GD}\}$

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

$\mathbf{P}_g(t) =$

$\{18_{AB}, 2_{AF}, 7_{BC}, 15_{BD}, 3_{CE}, 15_{CF}, 5_{DA}, 13_{EA}, 19_{EG}, 13_{FB}, 9_{FD}, 4_{FG}, 1_{GB}, 16_{GD}\}$

The inertia weight, w, determines the number of replacements that will be randomly selected from $\mathbf{V}_i(t)$ (mentioned in Example 1 above). Since $w = 0.5$, and $|\mathbf{V}_i(t)| = 14$, the number of randomly selected replacements is $\lfloor 0.5 \times |\mathbf{V}_i(t)| \rfloor = 7$. Thus, any seven replacements from the set $\mathbf{V}_i(t)$ can be taken randomly. Consider

that those replacements are $\{(2 \Rightarrow 1)_{AF}, (4 \Rightarrow 7)_{BC}, (4 \Rightarrow 3)_{CE}, (6 \Rightarrow 5)_{DA}, (12 \Rightarrow 14)_{EA}, (13 \Rightarrow 19)_{EG}, (10 \Rightarrow 13)_{FB}\}$.

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 the weight on each link in $\mathbf{X}_i(t)$ with the weight of the corresponding link in $\mathbf{P}_i(t)$ as:

$\mathbf{P}_i(t) \oslash \mathbf{X}_i(t) = \{(18 \Rightarrow 18)_{AB}, (1 \Rightarrow 12)_{AF}, (7 \Rightarrow 7)_{BC}, (15 \Rightarrow 15)_{BD}, (3 \Rightarrow 3)_{CE}, (17 \Rightarrow 16)_{CF}, (5 \Rightarrow 5)_{DA}, (14 \Rightarrow 13)_{EA}, (19 \Rightarrow 19)_{EG}, (13 \Rightarrow 13)_{FB}, (18 \Rightarrow 8)_{FD}, (4 \Rightarrow 4)_{FG}, (16 \Rightarrow 12)_{GB}, (16 \Rightarrow 16)_{GD}\}$.

Now, $c_1 \times r_1 \otimes (\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)) = \lfloor 0.5 \times 0.52 \times |\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)| \rfloor$. Replacements involving new and old weights having the same value are ignored. Thus, the cardinality of $\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)$ is 5. This implies that $0.5 \times 0.52 \otimes |\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)| = 1.3 = 1$. This means that any one of the five elements in $\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)$ is randomly chosen. Assume that $c_1 \times r_1 \otimes (\mathbf{P}_i(t) \oslash \mathbf{X}_i(t)) = \{(18 \Rightarrow 8)_{FD}\}$.

Similarly,

$\mathbf{P}_g(t) \oslash \mathbf{X}_i(t) = \{(18 \Rightarrow 18)_{AB}, (1 \Rightarrow 2)_{AF}, (7 \Rightarrow 7)_{BC}, (15 \Rightarrow 15)_{BD}, (3 \Rightarrow 3)_{CE}, (17 \Rightarrow 15)_{CF}, (5 \Rightarrow 5)_{DA}, (14 \Rightarrow 13)_{EA}, (19 \Rightarrow 19)_{EG}, (13 \Rightarrow 13)_{FB}, (18 \Rightarrow 9)_{FD}, (4 \Rightarrow 4)_{FG}, (16 \Rightarrow 1)_{GB}, (16 \Rightarrow 16)_{GD}\}$.

The cardinality of the above set is 5. Therefore, $\lfloor 0.5 \times 0.75 \times |\mathbf{P}_g(t) \oslash \mathbf{X}_i(t)| \rfloor = 0.5 \times 0.75 \times 5 = 1.3 = 1$ replacement. Assume $\{(17 \Rightarrow 15)_{CF}\}$ is randomly chosen.

Substitution of the results above in Equation (7.1) gives $\mathbf{V}_i(t+1)$ containing 9

elements, i.e.

$\mathbf{V}_i(t+1) = \{(2 \Rightarrow 1)_{AF}, (4 \Rightarrow 7)_{BC}, (4 \Rightarrow 3)_{CE}, (6 \Rightarrow 5)_{DA}, (12 \Rightarrow 14)_{EA}, (13 \Rightarrow 19)_{EG}, (10 \Rightarrow 13)_{FB}, (18 \Rightarrow 8)_{FD}, (17 \Rightarrow 15)_{CF}\}$.

Since $V_{max} = 2$, only two replacements from $\mathbf{V}_i(t+1)$ are randomly chosen. Assume that $(18 \Rightarrow 8)_{FD}$ and $(17 \Rightarrow 15)_{CF}$ are chosen. Then,

$\mathbf{V}_i(t+1) = \{(18 \Rightarrow 8)_{FD}, (17 \Rightarrow 15)_{CF}\}$.

7.1.3 Particle Position Update

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

$$\mathbf{X}_i(t+1) = \mathbf{X}_i(t) \biguplus \mathbf{V}_i(t+1) \tag{7.2}$$

where $\biguplus$ is a special operator that updates the links in $\mathbf{X}_i(t)$ on the basis of weight replacements in $\mathbf{V}_i(t+1)$, to obtain the new position $\mathbf{X}_i(t+1)$.

Example 3: Consider Example 2, for which

$\mathbf{X}_i(t+1) = \mathbf{X}_i(t) \biguplus \mathbf{V}_i(t+1) = \{18_{AB}, 1_{AF}, 7_{BC}, 15_{BD}, 3_{CE}, 17_{CF}, 5_{DA}, 14_{EA}, 19_{EG}, 13_{FB}, 18_{FD}, 4_{FG}, 16_{GB}, 16_{GD}\} \biguplus \{(17 \Rightarrow 15)_{CF}, (18 \Rightarrow 8)_{FD}\} = \{18_{AB}, 1_{AF}, 7_{BC}, 15_{BD}, 3_{CE}, \mathbf{15_{CF}}, 5_{DA}, 14_{EA}, 19_{EG}, 13_{FB}, \mathbf{8_{FD}}, 4_{FG}, 16_{GB}, 16_{GD}\}$.

Thus, in the new solution, weight 18 on link FD is replaced by 8 and weight 17 on link CF is replaced by 15. Subsequently, the cost of the new solution is calculated

as explained in Chapter 4 and the algorithm continues until the stopping criterion is satisfied. It is noteworthy to discuss the similarities and differences between the proposed set based approach and earlier research work. For the OSPFWS problem, a fixed size set representation for particles is used in this thesis, which is similar to the approach taken by Chen *et al.* [19], and Khan and Engelbrecht [89] in their respective work. The differences are as follows:

1. The particle position is defined as a set of links between network nodes in the work of Khan and Engelbrecht [89]. For OSPFWS the particle position is defined as a set of OSPF weights on the network links.

2. Khan and Engelbrecht [89] defined particle velocity as a set of link exchange operations. For the approach proposed here, the particle velocity is defined as a set of OSPF weight replacements on the network links.

3. Because the set-based PSO approach proposed in this thesis incorporates the OSPFWS problem specific elements, for example fixed set sizes, it is not a generic true set-based PSO as proposed by Langeveld and Engelbrecht [101].

7.1.4 Control Parameter Tuning for Fuzzy Particle Swarm Optimization

This section discusses the process of obtaining best parameter combination for the FPSO algorithm with respect to each test case. The parameter combination that resulted in the best quality solutions with respect to FuzzyCF is considered as the best parameter combination. For each test, 30 independent runs were executed. The average of the best fuzzy cost and the standard deviation over 30 runs were calculated. The Wilcoxon rank-sum test [65] was used to validate the significance of the results. A confidence level of 95% was used. The first set of experiments focussed on obtaining the best swarm size. The values $c_1 = c_2 = 1.49$, $w = 0.72$ and $V_{max} = 5$ were used. The initial swarm size was set to 20 for each test case. Subsequently, the swarm size was increased by 20. As soon as a higher swarm size produced a result that is statistically significantly worse than a previously evaluated lower swarm size, the lower swarm size was considered as the best swarm size.

Tables 7.1 and 7.2 provide the quality of solutions with respect to FuzzyCF obtained for all the evaluated swarm sizes. Column 1 represents the test case. Column 2 represents the swarm size. Column 3 represents the average cost of the best solutions obtained over the 30 independent runs using FuzzyCF. Column 4 represents the percentage difference between the average fuzzy cost values obtained for two consecutive swarm sizes. Because swarm size 20 is used as the initial value, the percentage dif-

ference value for swarm size 20 is NA (not applicable). A swarm size of 40 produced the best quality solutions with respect to FuzzyCF for the five test cases h100N280a, h50N148a, r50N245a, w50N169a, and w50N230a. For the rest of the test cases, a swarm size of 100 produced the best quality solutions.

After setting the value of the swarm size for each test case, further experimentation was done to obtain the best value for V_{max}. As discussed in Section 2.3.5, V_{max} confines the step size of particles. In other words V_{max} confines the level of perturbation. With respect to the OSPF protocol, changes on very few weights of the links may establish a completely different route. To avoid such undesirable large changes or perturbations, values of 5, 10, 15, and 20 for V_{max} were evaluated. These values correspond to perturbation rates of 3% to 5%.

Table 7.3 shows the obtained best average fuzzy cost values with respect to FPSO over 30 independent runs. It is observed from the table that $V_{max} = 5$ produced the best results for test cases h100N280a, h50N148a, r50N245a, w50N169a, and w50N230a. For the rest of the test cases, $V_{max} = 15$ produced the best results. The results produced by $V_{max} = 5$ was compared with the results produced by $V_{max} = 10$, $V_{max} = 15$ and $V_{max} = 20$ using Wilcoxon rank-sum test [65].

Table 7.4 shows that all the results were statistically significant except the test case h100N280a, where the result produced by $V_{max} = 5$ is statistically insignificant when compared with the result produced by $V_{max} = 10$. Similarly, the results produced

Table 7.1: Best average FuzzyCF values for the test cases h50N148a, h50N212a, r50N228a, r50N245a, w50N169a and w50N230a for the experimented swarm sizes. Statistically significant differences are in italics. NA = Not Applicable (% Difference is calculated between the average fuzzy cost values obtained for two consecutive swarm sizes).

Test Case	Swarm Size	Best Fuzzy Cost	% Difference	p-values
h50N148a	20	0.353±0.014	NA	NA
	40	0.45±0.017	*21.55*	0.024
	60	0.456±0.026	1.31	0.167
h50N212a	20	0.389±0.062	NA	NA
	40	0.47±0.025	*17.23*	0.021
	60	0.482±0.026	*2.4*	0.013
	80	0.5±0.02	*3.6*	0.026
	100	0.504±0.018	*0.79*	0.014
	120	0.508±0.016	0.78	0.115
r50N228a	20	0.443±0.013	NA	NA
	40	0.482±0.026	*8.09*	0.037
	60	0.514±0.015	*6.22*	0.045
	80	0.532±0.024	*3.38*	0.024
	100	0.543±0.013	*2.02*	0.037
	120	0.544±0.019	0.18	0.227
r50N245a	20	0.444±0.027	NA	NA
	40	0.499±0.025	*11.02*	0.043
	60	0.54±0.028	7.59	0.132
w50N169a	20	0.506±0.032	NA	NA
	40	0.541±0.019	*6.46*	0.012
	60	0.543±0.026	0.36	0.146
w50N230a	20	0.4±0.083	NA	NA
	40	0.504±0.035	*20.63*	0.038
	60	0.506±0.034	0.39	0.187

Table 7.2: Best average FuzzyCF values for the test cases h100N280a, h100N360a, r100N403a, r100N503a, w100N391a and w100N476a for the experimented swarm sizes. Highest average cost values are in boldface. Statistically significant differences are in italics. NA = Not Applicable (% Difference is calculated between the average fuzzy cost values obtained for two consecutive swarm sizes).

Test Case	Swarm Size	Best Fuzzy Cost	% Difference	p-values
h100N280a	20	0.433±0.033	NA	NA
	40	0.471±0.025	*8.06*	0.015
	60	0.473±0.025	0.42	0.103
h100N360a	20	0.434±0.039	NA	NA
	40	0.48±0.035	*9.58*	0.006
	60	0.514±0.03	*6.61*	0.025
	80	0.529±0.024	*2.83*	0.027
	100	0.543±0.033	*2.57*	0.001
	120	0.545±0.037	0.36	0.154
r100N403a	20	0.392±0.011	NA	NA
	40	0.425±0.037	*7.76*	0.005
	60	0.447±0.024	*4.92*	0.023
	80	0.47±0.024	*4.89*	0.022
	100	0.481±0.027	*2.28*	0.04
	120	0.483±0.017	0.41	0.115
r100N503a	20	0.41±0.012	NA	NA
	40	0.474±0.03	*13.5*	0.014
	60	0.506±0.026	*6.32*	0.019
	80	0.52±0.023	*2.69*	0.016
	100	0.543±0.043	*4.23*	0.004
	120	0.547±0.013	0.73	0.2
w100N391a	20	0.409±0.078	NA	NA
	40	0.49±0.062	*16.53*	0.027
	60	0.563±0.037	*12.96*	0.028
	80	0.582±0.042	*3.26*	0.039
	100	0.609±0.032	*4.43*	0.047
	120	0.61±0.039	0.16	0.143
w100N476a	20	0.489±0.024	NA	NA
	40	0.573±0.022	*14.65*	0.031
	60	0.614±0.052	*6.67*	0.042
	80	0.630±0.072	*2.53*	0.03
	100	0.657 ±0.095	*4.1*	0.034
	120	0.66±0.025	0.45	0.116

by $V_{max} = 15$ was compared with the results produced by $V_{max} = 5$, $V_{max} = 10$ and $V_{max} = 20$. Table 7.5 shows that all the results were statistically significant except for the test cases h50N212a, r50N228a, and w100N476a when compared with the results produced by $V_{max} = 20$.

Table 7.3: Average FuzzyCF values for all the test cases with respect to FPSO. Experimented V_{max} values were 5, 10, 15 and 20. Best FuzzyCF values are highlighted in boldface.

Test Case	$V_{max} = 5$ Set (1)	$V_{max} = 10$ Set (2)	$V_{max} = 15$ Set (3)	$V_{max} = 20$ Set (4)
h100N280a	**0.471**±0.033	0.462±0.028	0.455±0.018	0.45±0.021
h100N360a	0.475±0.033	0.482±0.051	**0.543**±0.034	0.503±0.039
h50N148a	**0.45**±0.017	0.414±0.026	0.4±0.019	0.387±0.023
h50N212a	0.465±0.018	0.479±0.019	**0.504**±0.022	0.485±0.038
r100N403a	0.435±0.027	0.457±0.014	**0.481**±0.031	0.46±0.017
r100N503a	0.485±0.025	0.52±0.023	**0.543**±0.029	0.512±0.02
r50N228a	0.487±0.053	0.517±0.021	**0.543**±0.032	0.52±0.023
r50N245a	**0.499**±0.025	0.48±0.024	0.465±0.03	0.458±0.093
w100N391a	0.498±0.022	0.528±0.05	**0.609**±0.063	0.572±0.083
w100N476a	0.587±0.025	0.625±0.023	**0.657**±0.022	0.632±0.013
w50N169a	**0.541**±0.041	0.523±0.033	0.51±0.019	0.482±0.026
w50N230a	**0.504**±0.035	0.482±0.08	0.475±0.038	0.478±0.019

After setting the value of V_{max}, the best values for the parameters inertia weight, w, and acceleration coefficients, c_1 and c_2, were determined. The experiments were conducted using all the values of $w \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ with all the values of $c_1, c_2 \in \{0.0, 0.5, 1.0, 1.5, 2.0\}$. The values of w, c_1 and c_2 that resulted in the best quality solutions with respect to FuzzyCF for FPSO are tabulated in Table 7.6 for all the test cases. An archive of non-dominated solutions was maintained

Table 7.4: Percentage difference of FuzzyCF values for the test cases where $V_{max} = 5$ performed better. Significant differences are highlighted in boldface.

Test Case	5 vs 10	p-values	5 vs 15	p-values	5 vs 20	p-values
h100N280a	1.91	0.164	**3.4**	0.008	**4.46**	0.016
h50N148a	**8**	0.021	**11.11**	0.026	**14**	0.025
r50N245a	**3.81**	0.03	**6.81**	0.018	**8.22**	0.019
w50N169a	**3.33**	0.004	**5.73**	0.015	**10.91**	0.011
w50N230a	**4.37**	0.011	**5.75**	0.006	**5.16**	0.021

Table 7.5: Percentage difference of FuzzyCF values for the test cases where $V_{max} = 15$ performed better. Significant differences are highlighted in boldface.

Test Case	15 vs 5	p-values	15 vs 10	p-values	15 vs 20	p-values
h100N360a	**12.52**	0.003	**11.23**	0.013	**7.37**	0.015
h50N212a	**7.74**	0.011	**4.96**	0.019	3.77	0.142
r100N403a	**9.56**	0.042	**4.99**	0.029	**4.37**	0.002
r100N503a	**10.68**	0.012	**4.24**	0.013	**5.71**	0.019
r50N228a	**10.31**	0.032	**4.79**	0.037	4.24	0.135
w100N391a	**18.23**	0.009	**13.3**	0.02	**6.08**	0.025
w100N476a	**10.65**	0.037	**4.87**	0.017	3.81	0.128

for FPSO as described in Section 5.2.

Table 7.6: Best parameter combinations obtained for FPSO for each test case.

Test Case	Swarm Size	c_1	c_2	w	V_{max}	FPSO fuzzy cost
h100N280a	40	1.5	1.5	0.7	5	0.531±0.018
h100N360a	100	1.5	1.5	0.7	15	0.543±0.036
h50N148a	40	1.5	1.5	0.7	5	0.437±0.021
h50N212a	100	1.5	1.5	0.7	15	0.504±0.015
r100N403a	100	1.5	1.5	0.7	15	0.481±0.017
r100N503a	100	1.5	1.5	0.7	15	0.543±0.013
r50N228a	100	1.5	1.5	0.7	15	0.543±0.019
r50N245a	40	1.5	1.5	0.7	5	0.557±0.019
w100N391a	100	1.5	1.5	0.7	15	0.609±0.029
w100N476a	100	1.5	1.5	0.7	15	0.657±0.025
w50N169a	40	1.5	1.5	0.7	5	0.595±0.021
w50N230a	40	1.5	1.5	0.7	5	0.591±0.029

To study the convergence behavior of FPSO, the average fuzzy cost of all the particles over 30 independent runs and the associated standard deviation is plotted against the iterations in Figures 7.1 and 7.2 for all the test cases. These experiments were conducted by using the optimized parameter combinations given in Table 7.6. Observe from the figures that the average fuzzy cost of all the particles increased over time. Contrary to this, the standard deviation decreased over time and became almost zero. This shows good convergence because all the particles reached the optimal region by having a zero or negligible difference between their cost values.

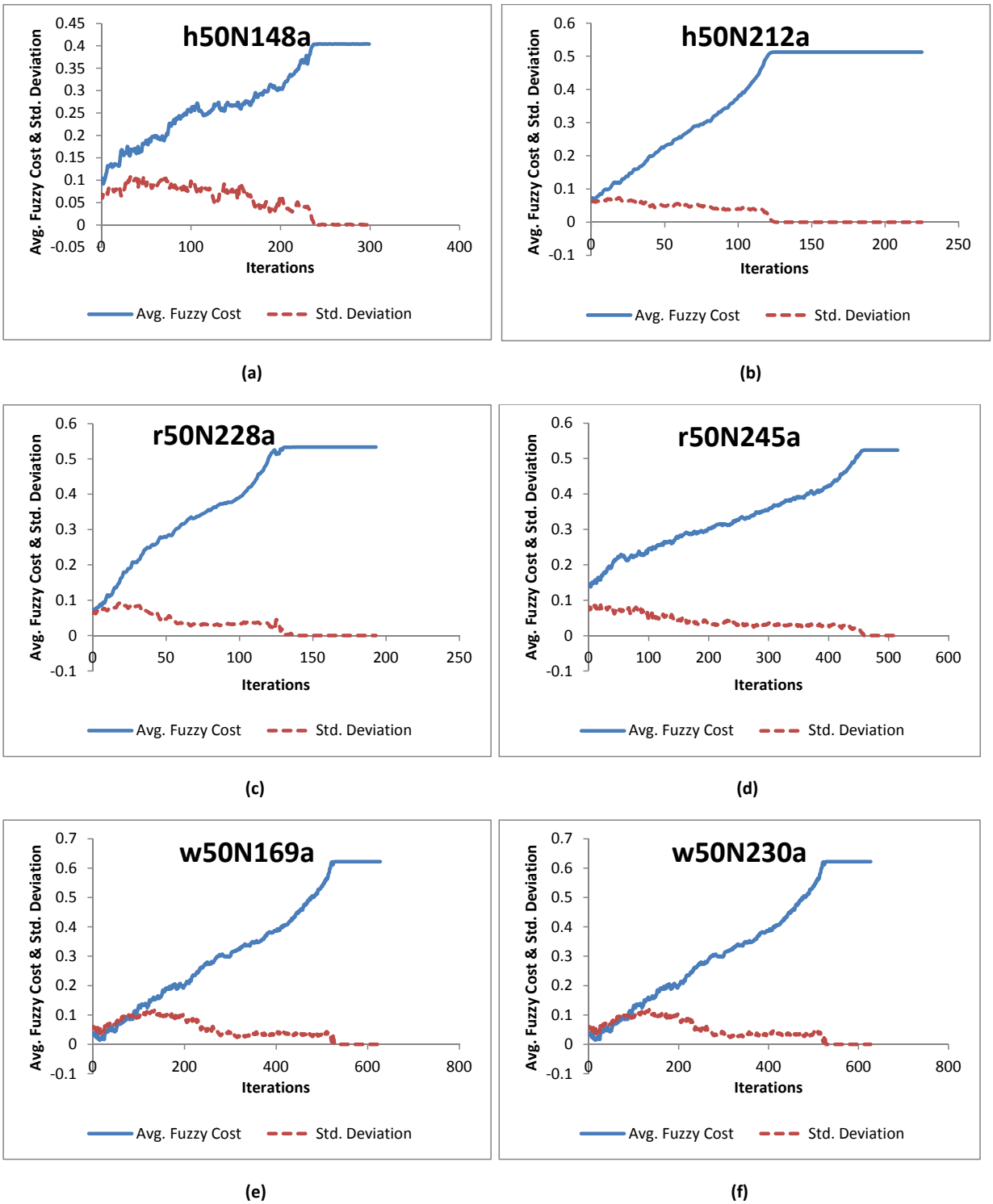


Figure 7.1: Average Fuzzy cost and associated standard deviation curves of FPSO for all the test cases with 50 nodes.

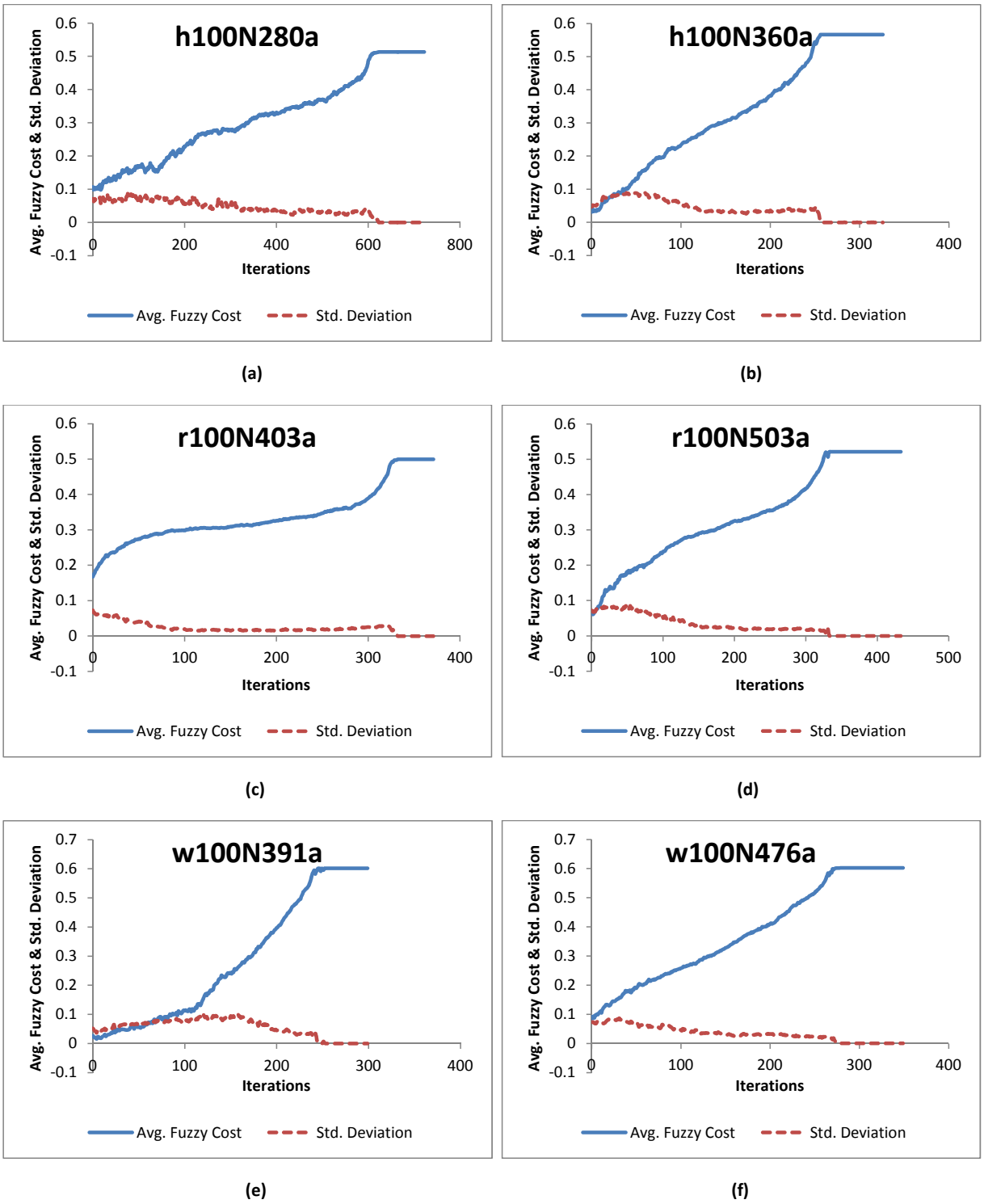


Figure 7.2: Average Fuzzy cost and associated standard deviation curves of FPSO for all the test cases with 100 nodes.

7.2 Fuzzy Evolutionary Particle Swarm Optimization Algorithm

This section discusses the implementation of FEPSO for the OSPFWS problem. Control parameter tuning for FEPSO is described in Section 7.2.1. In addition to the FPSO algorithm for the OSPFWS problem, a hybrid variant of the FPSO using the SimE algorithm was also developed to avoid replacement of high quality weights. This hybrid variant is referred to as the FEPSO. The weight replacement for the OSPFWS problem involves replacing old weights with new weights on the links. Furthermore, the total number of performed replacements is limited by the parameter V_{max} in PSO. It is possible that, for a link i, a replacement may remove a weight (to be replaced with another weight), which might already be an optimum (or near-optimum) weight for that link. Note that this replacement is done "blindly", that is, the value of the new weight is chosen randomly. If these blind replacements continue for other links having optimum weights, then it will take a significant amount of time for the algorithm to converge. Rather than having a blind replacement, it would be more appropriate to replace a weight based on its quality. A weight with low quality will have a high probability of being removed from its current position, and vice versa.

The question is how to measure the quality of a weight. This can be answered by incorporating the evaluation and selection phases of the SimE algorithm into the

FPSO algorithm, as discussed in Section 6.1. The evaluation is performed for all current weights which are part of the set $\mathbf{V}_i(t+1)$ as defined by Equation (7.1).

Once the goodness of each existing weight in $\mathbf{V}_i(t+1)$ is evaluated, the selection phase chooses the weights that would be replaced with new weights. This selection is done probabilistically based on the quality of existing weights in $\mathbf{V}_i(t+1)$. A random number, r, in the range [0,1] is generated. If $r \leq 1 - g_{ij} + B$, then the existing weight is selected for replacement, otherwise no replacement is done. In the above expression, g_{ij} refers to the goodness of the current weight on the link connecting nodes i and j, and B is the selection bias. Algorithm 7.1 provides pseudo-code of the selection function for FEPSO. The selection process is illustrated by the following example.

Example 4: In Example 2, $\mathbf{V}_i(t+1)$ was found as follows:

$\mathbf{V}_i(t+1) = \{(2 \Rightarrow 1)_{AF}, (4 \Rightarrow 7)_{BC}, (4 \Rightarrow 3)_{CE}, (6 \Rightarrow 5)_{DA}, (12 \Rightarrow 14)_{EA}, (13 \Rightarrow 19)_{EG}, (10 \Rightarrow 13)_{FB}, (18 \Rightarrow 8)_{FD}, (17 \Rightarrow 15)_{CF}\}$

In FEPSO, the replacements will be done based on the goodness of weights. Assume that weight 2 on link AF, weight 4 on link CE, weight 10 on link FB, weight 18 on link FD and weight 17 on link CF are of low goodness. Out of these 5 links assume that the replacements $(4 \Rightarrow 3)_{CE}$, $(10 \Rightarrow 13)_{FB}$ and $(17 \Rightarrow 15)_{CF}$ were probabilistically selected based on the selection procedure. However, since $V_{max} = 2$, only two replacements were selected randomly out of the three. So, a possible result could

be

$$\mathbf{V}_i(t+1) = \{(10 \Rightarrow 13)_{FB},\ (17 \Rightarrow 15)_{CF}\}$$

Algorithm 7.1 Selection(B); Weight replacement function of FEPSO.

```
/* B: Selection Bias; */
Get the set of possible replacements V_i(t+1) from Equation (7.1)
For all the current weights in the set V_i(t+1) Do
    Calculate the goodness g_ij of the weight on link between nodes i and j using
    Equation (6.1)
    If r ≤ 1 − g_ij + B Then
        Allow the replacement of the old weight with the new weight;
    Else
        Do not allow the replacement of the old weight with the new weight;
    End If
End For
End Selection;
```

7.2.1 Control Parameter Tuning for Fuzzy Evolutionary Particle Swarm Optimization

The best swarm size and V_{max} values found for the FPSO were used for the FEPSO. Further experiments were conducted to determine the best values of the parameters inertia weight, w, and acceleration coefficients, c_1 and c_2. For each test, 30 independent runs were executed. The average of the best fuzzy cost and the standard deviation over 30 runs were calculated. Similar to FPSO, the experiments were conducted using each of the values of $w \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ with each of the values of $c_1, c_2 \in \{0.0, 0.5, 1.0, 1.5, 2.0\}$. The values of w, c_1 and c_2 that re-

sulted in the best quality solutions with respect to FuzzyCF for FEPSO are tabulated in Table 7.7 for all the test cases. During the search, an archive of non-dominated solutions was maintained as described in Section 5.2.

Table 7.7: Best parameter combinations obtained for FEPSO for each test case.

Test Case	Swarm Size	c_1	c_2	w	V_{max}	FEPSO fuzzy cost
h100N280a	40	1.5	1.5	0.7	5	0.526±0.015
h100N360a	100	1.5	1.5	0.7	15	0.605±0.012
h50N148a	40	1.5	1.5	0.7	5	0.469±0.019
h50N212a	100	1.5	1.5	0.7	15	0.528±0.013
r100N403a	100	1.5	1.5	0.7	15	0.595±0.011
r100N503a	100	1.5	1.5	0.7	15	0.710±0.012
r50N228a	100	1.5	1.5	0.7	15	0.610±0.016
r50N245a	40	1.5	1.5	0.7	5	0.644±0.014
w100N391a	100	1.5	1.5	0.7	15	0.725±0.010
w100N476a	100	1.5	1.5	0.7	15	0.757±0.011
w50N169a	40	1.5	1.5	0.7	5	0.640±0.012
w50N230a	40	1.5	1.5	0.7	5	0.711±0.022

After determining the c_1, c_2 and w values, further experimentation was done to determine the parameter bias, B. A bias value in the range $[-0.2, 0.2]$ was recommended by Sait and Youssef [150]. For the FEPSO algorithm, a best value for the bias parameter was determined from the set of values $\{-0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3\}$. For each test, 30 independent runs were executed with these bias values. Figure 7.3 illustrates the best average results of FuzzyCF for all the test cases. A bias value of -0.2 was found to generate statistically significantly better quality solutions than other bias values. Hence, a bias value of -0.2 was used for the FEPSO algorithm runs. Figures 7.4 and 7.5 plot the average fuzzy cost versus standard deviation for all the

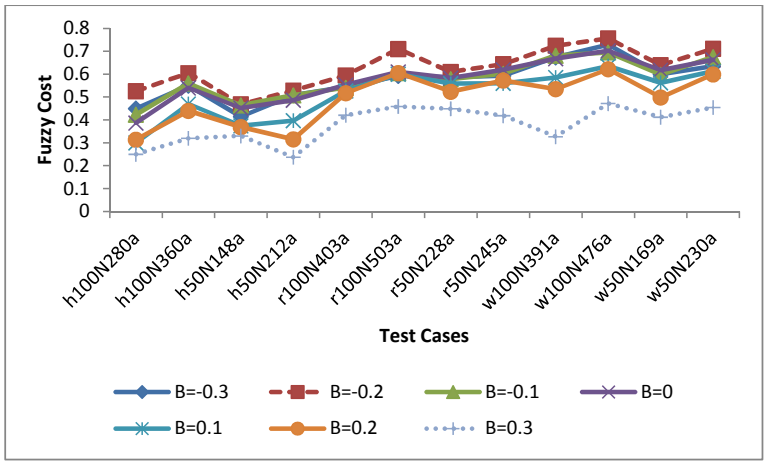


Figure 7.3: Results obtained for different bias values for FEPSO.

test cases over 30 independent runs. The figures illustrate that the average fuzzy cost increased and that the standard deviation decreased over time for all the test cases. Thus, all the particles converged to the optimal region.

7.3 Weighted Aggregation Particle Swarm Optimization Algorithm

The purpose of this section is to describe the WAPSO algorithm for the OSPFWS problem. Section 7.3.1 discusses the control parameter tuning for WAPSO. The FPSO and FEPSO algorithms were based on the fuzzy cost function (FuzzyCF). WAPSO is based on the concept of a weighted sum of objectives as discussed in Section 2.1.1. Both the fuzzy and the weighted-aggregation approaches convert the

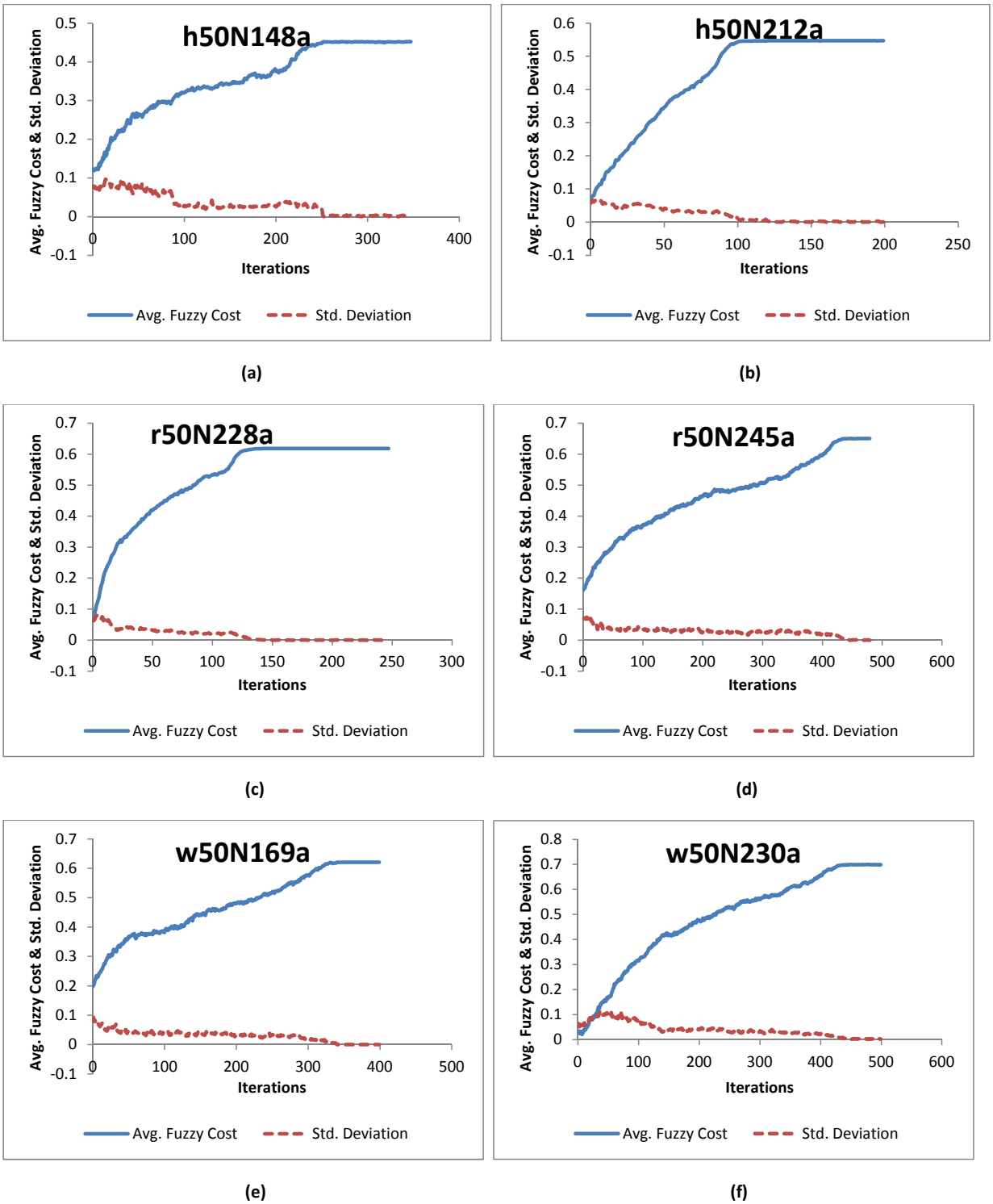


Figure 7.4: Average Fuzzy cost and associated standard deviation curves of FEPSO for all the test cases with 50 nodes.

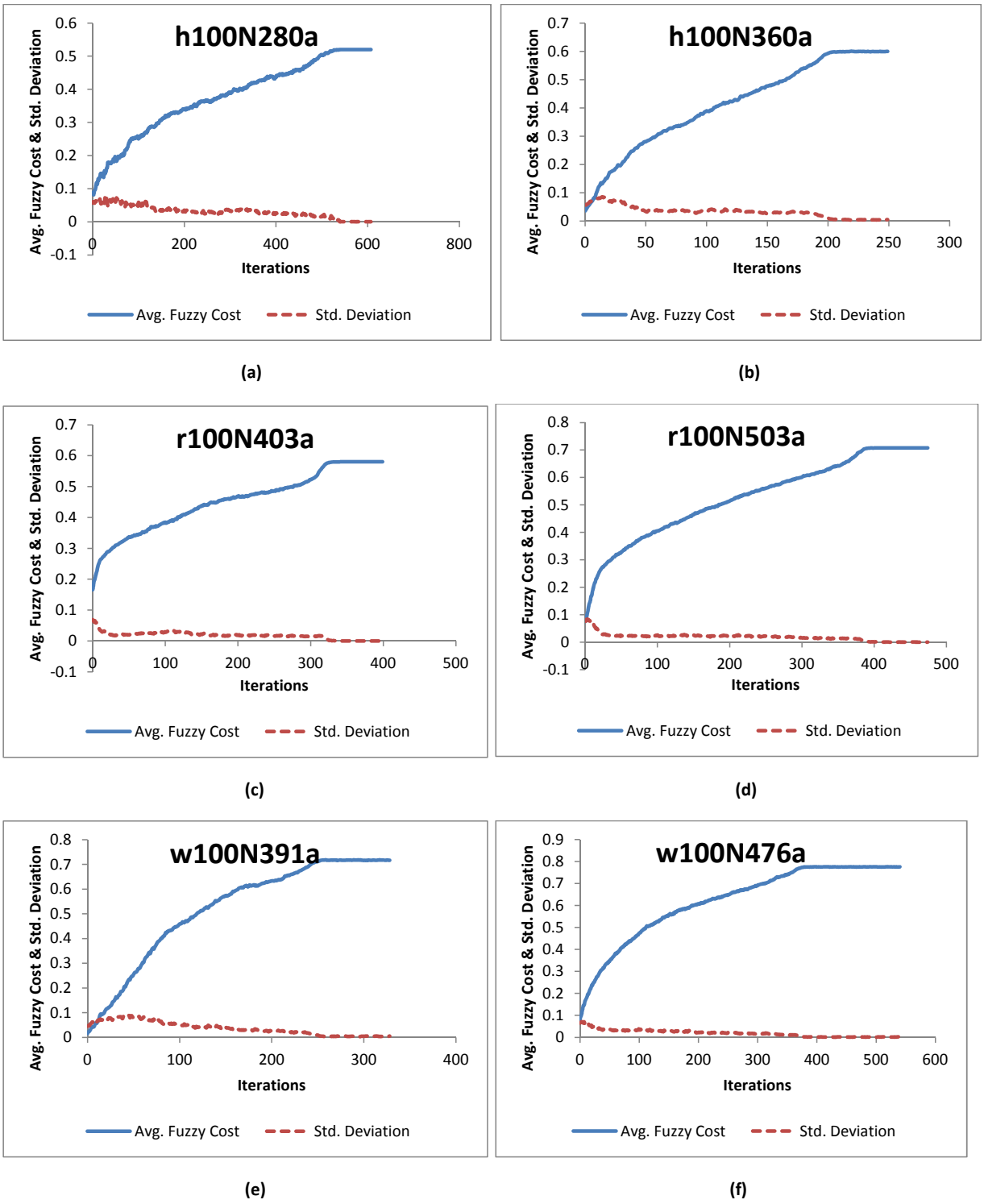


Figure 7.5: Average Fuzzy cost and associated standard deviation curves of FEPSO for all the test cases with 100 nodes.

multi-objective optimization problem into a single-objective optimization problem. The fuzzy approach evaluates a candidate solution based on the membership values obtained for each objective. Contrary to this, the weighted aggregation approach associates weights to each objective. The values of these weights are decided based on three aspects: (1) the importance of each objective in the optimization process, (2) to balance conflicting objectives, and (3) to normalize the objectives into the same range. For the OSPFWS problem, the objective MU is usually a fraction ranging between 0 to 10, while NOC and NUL are integers. In order to normalize these objectives, each objective is divided by its corresponding maximum value. Thus, the single-objective problem is formulated as:

$$f(x) = \left(\lambda_1 * \frac{MU(x)}{MU_{max}}\right) + \left(\lambda_2 * \frac{NOC(x)}{NOC_{max}}\right) + \left(\lambda_3 * \frac{NUL(x)}{NUL_{max}}\right) \tag{7.3}$$

where λ_1, λ_2 and λ_3 are the weights associated with the objectives MU, NOC, and NUL respectively (the sum of all these weights should be equal to 1). Because the objectives of the OSPFWS problem are to be minimized, the above function is minimized.

7.3.1 Control Parameter Tuning for Weighted Aggregation Particle Swarm Optimization

The best swarm size and V_{max} values found for the FPSO were used for the WAPSO. Further experiments were conducted to determine the best values for the inertia

weight, w, and the acceleration coefficients, c_1 and c_2. For each control parameter value configuration, 30 independent runs were executed. The average of the best weighted aggregation cost and the standard deviation over 30 runs were calculated. Similar to FPSO, the experiments were conducted using each of the values of $w \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ with each of the values of $c_1, c_2 \in \{0.0, 0.5, 1.0, 1.5, 2.0\}$. The values of w, c_1 and c_2 that resulted in the best quality solutions with respect to Equation (7.3) for WAPSO are tabulated in Table 7.8 for all the test cases. As for PSO and FEPSO, an archive of non-dominated solutions was maintained for WAPSO.

Table 7.8: Best parameter combinations obtained for WAPSO for each test case.

Test Case	Swarm Size	c_1	c_2	w	V_{max}	Weighted aggregation cost
h100N280a	40	1.5	1.5	0.7	5	0.301 ± 0.063
h100N360a	100	1.5	1.5	0.7	15	0.266 ± 0.077
h50N148a	40	1.5	1.5	0.7	5	0.366 ± 0.192
h50N212a	100	1.5	1.5	0.7	15	0.325 ± 0.091
r100N403a	100	1.5	1.5	0.7	15	0.361 ± 0.052
r100N503a	100	1.5	1.5	0.7	15	0.305 ± 0.043
r50N228a	100	1.5	1.5	0.7	15	0.306 ± 0.167
r50N245a	40	1.5	1.5	0.7	5	0.263 ± 0.018
w100N391a	100	1.5	1.5	0.7	15	0.218 ± 0.045
w100N476a	100	1.5	1.5	0.7	15	0.209 ± 0.114
w50N169a	40	1.5	1.5	0.7	5	0.227 ± 0.032
w50N230a	40	1.5	1.5	0.7	5	0.235 ± 0.086

In order to find the best objective function weight values, ten weight combinations were evaluated as given in Table 7.9. A motivation for each weight combination is also provided in Table 7.9. Figure 7.6 illustrates the cost values for all the test cases with

respect to these six weight combinations. It is evident from the figure that weight combination $\lambda_1 = 0.333, \lambda_2 = 0.333$ and $\lambda_3 = 0.333$ performed better than the other combinations for all the test cases. Thus, in this thesis equal importance is given to all the three objectives.

Table 7.9: Experimented weight combinations for WAPSO.

Serial No	λ_1	λ_2	λ_3	Description
1	0.333	0.333	0.333	Equal importance is given to all
2	0.5	0.25	0.25	Importance is given to MU
3	0.25	0.5	0.25	Importance is given to NOC
4	0.25	0.25	0.5	Importance is given to NUL
5	0.4	0.4	0.2	Less importance is given to NUL than other two objectives
6	0.4	0.2	0.4	Less importance is given to NOC than other two objectives
7	0.2	0.4	0.4	Less importance is given to MU than other two objectives
8	0.6	0.3	0.1	Importance is given in the decreasing order: MU, NOC and NUL
9	0.3	0.6	0.1	Importance is given in the decreasing order: NOC, MU and NUL
10	0.3	0.1	0.6	Importance is given in the decreasing order: NUL, MU and NOC

Figures 7.7 and 7.8 show the average weighted cost versus the standard deviation for all the test cases over 30 independent runs. The figures illustrate that the average weighted cost decreased and that the corresponding standard deviation decreased over time for all the test cases. This shows the convergence behaviour of all the particles towards an optimal region.

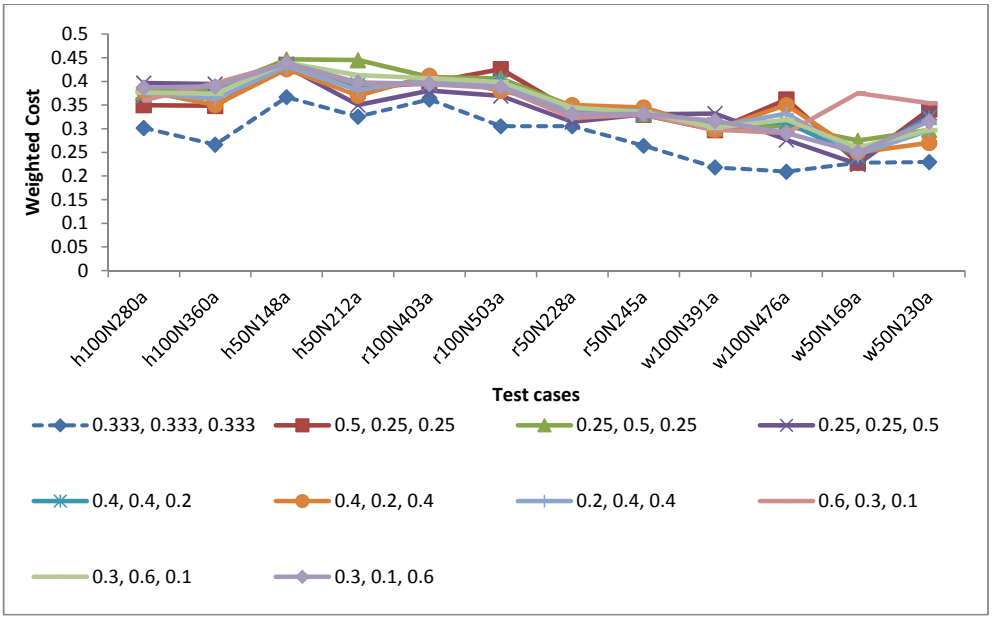


Figure 7.6: Weighted cost for ten weight combinations (λ_1, λ_2 and λ_3) for WAPSO for all the test cases.

7.4 Pareto-dominance Particle Swarm Optimization Algorithm

This section discusses the PDPSO algorithm for the OSPFWS problem. It is followed by a description of the control parameter tuning for PDPSO in Section 7.4.1. PSO achieves a balance between exploration and exploitation by progressing towards both the global best solution and particle best solutions. These global best and particle best solutions are referred to as the *global* and *local guides*, respectively. PDPSO is a dominance-based PSO. In this method an archive of non-dominated solutions is maintained to track all the obtained non-dominated solutions so far and to select global and local guides for each particle. Julio *et al.* [2] found that such selection of guides

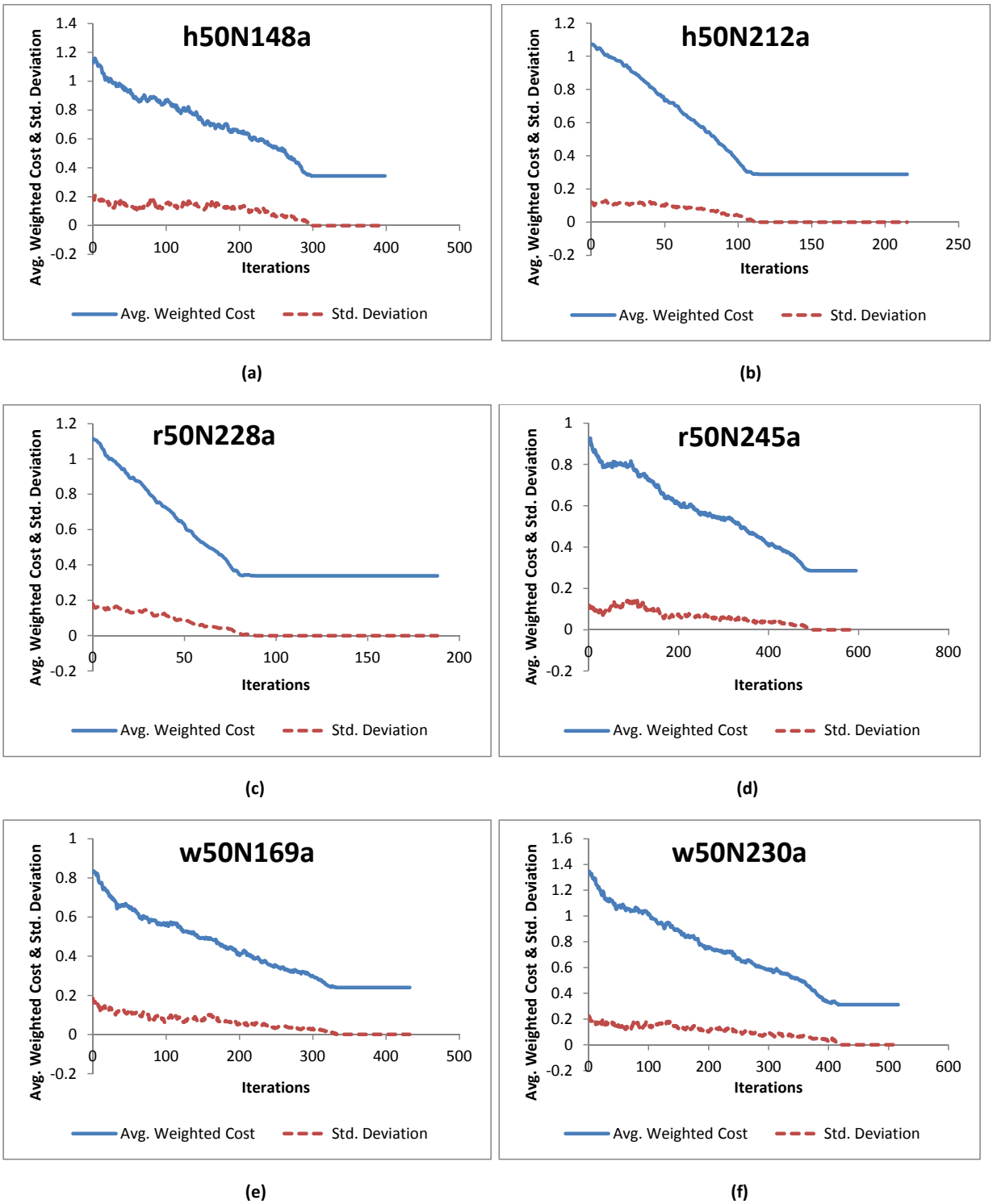


Figure 7.7: Average weighted cost and associated standard deviation curves of WAPSO for all the test cases with 50 nodes.

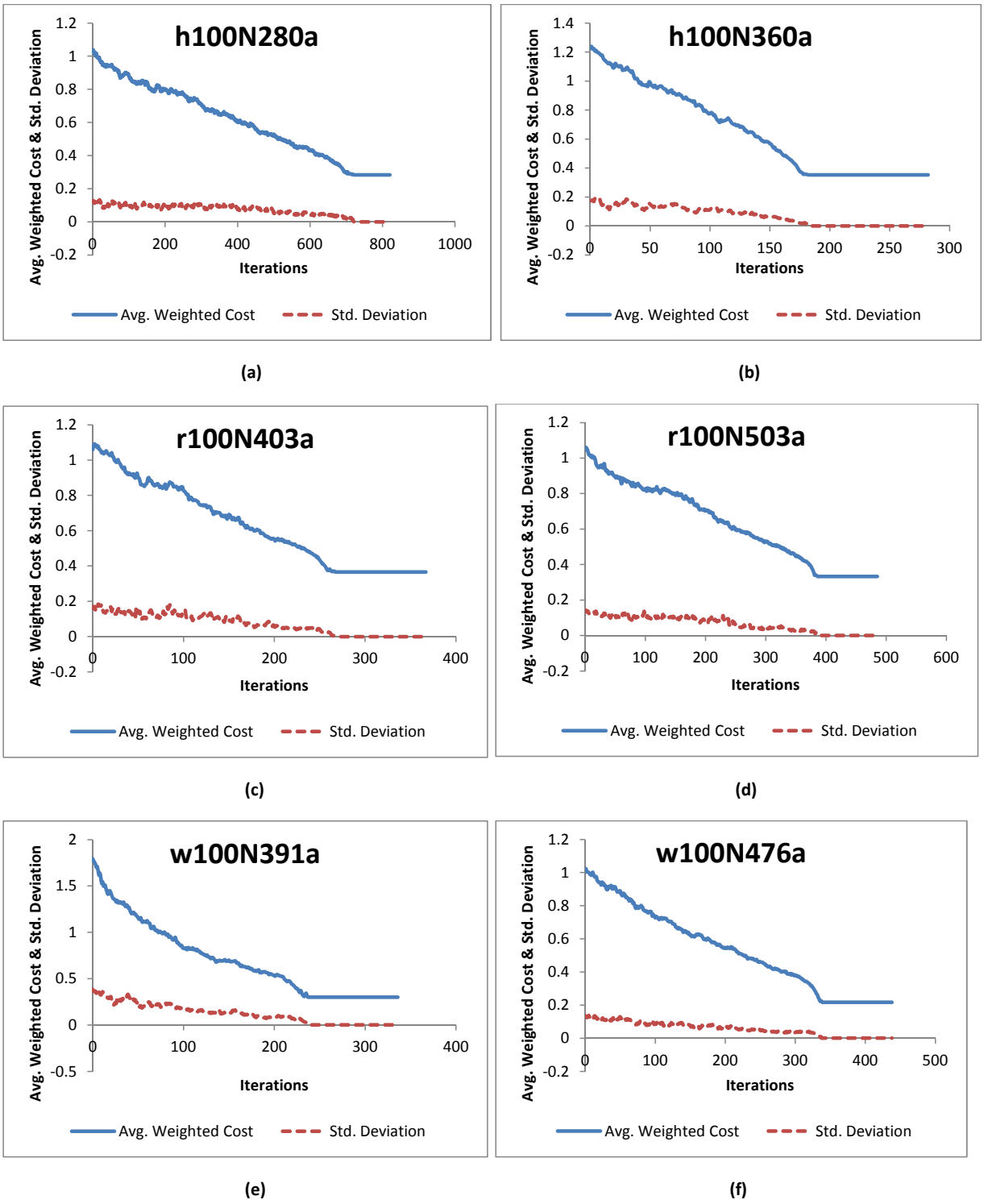


Figure 7.8: Average weighted cost and associated standard deviation curves of WAPSO for all the test cases with 100 nodes.

from an archive provides both good convergence and widespread coverage. In this thesis, PDPSO is implemented to solve the OSPFWS problem. The implementation details are given in subsequent paragraphs.

During initialization of the PDPSO, non-dominating particle positions are added to the archive. In subsequent iterations, if members of archive are dominated by the new position of any particle, then these members are deleted from the archive. Contrary to this, if the new position of any particle in the current swarm is not dominated by any member of the archive, then this new position is added to the archive. This process ensures a non-dominated set of solutions in the archive.

Local guides (personal best positions) are initialized to the particle's initial position (or solution). In subsequent iterations, if the new position of a particle dominates the particle's current local guide or both the new position and the particle's current local guide are mutually non-dominating each other, then the particle's current local guide is set to the new position.

The global guide is initialized to a randomly chosen member of the archive. In subsequent iterations, if the new position of a particle is not already part of the archive, then the global guide of this particle is chosen randomly from the members of the archive that dominates the new position. Contrary to this, if the new position of a particle is already present in the archive, then the global guide of this particle is chosen randomly from the entire archive.

7.4.1 Control Parameter Tuning for Pareto-dominance Particle Swarm Optimization

Because PDPSO is a dominance-based approach, no scalarized cost function is available to evaluate the current solution in each iteration. Therefore, the MOO hypervolume performance measure (discussed in Section 2.4) is used to evaluate the performance of PDPSO in this thesis.

The best swarm size and V_{max} values found for the FPSO were used for the PDPSO. For each test, 30 independent runs were executed. Further experimentations were done to determine the best values for the inertia weight, w, and the acceleration coefficients, c_1 and c_2. Similar to FPSO, the experiments were conducted using the values of $w \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ and the values of $c_1, c_2 \in \{0.0, 0.5, 1.0, 1.5, 2.0\}$. The values of w, c_1 and c_2 that resulted in the best quality solutions with respect to hypervolume metric for PDPSO are tabulated in Table 7.10 for all the test cases over 30 independent runs.

7.5 Non-dominating Sorting Genetic Algorithm II

The implementation details of NSGA-II for the OSPFWS problem is presented in this section. Section 7.5.1 describes the control parameter tuning for NSGA-II. An algorithmic description of NSGA-II was given in Section 2.3.1. Implementation of

Table 7.10: Best parameter combinations obtained for PDPSO for each test case.

Test Case	Swarm Size	c_1	c_2	w	V_{max}	hypervolume
h100N280a	40	1.5	1.5	0.7	5	3.924± 0.109
h100N360a	100	1.5	1.5	0.8	15	2.212± 0.226
h50N148a	40	1.5	1.5	0.6	5	3.592± 0.047
h50N212a	100	1.5	1.5	0.7	15	4.018± 0.363
r100N403a	100	1.5	1.5	0.7	15	4.865± 0.175
r100N503a	100	1.5	1.5	0.7	15	5.131± 0.139
r50N228a	100	1.5	1.5	0.7	15	4.185± 0.117
r50N245a	40	1.5	1.5	0.6	5	2.702± 0.185
w100N391a	100	1.5	1.5	0.7	15	6.401± 0.187
w100N476a	100	1.5	1.5	0.5	15	4.929± 0.058
w50N169a	40	1.5	1.5	0.7	5	4.202± 0.172
w50N230a	40	1.5	1.5	0.8	5	6.382± 0.036

NSGA-II for the OSPFWS problem is described below with an example using a population size of 10 and a crossover rate of 0.5.

1. Assume that the population has the following solutions in a particular iteration, given in the format of (MU, NOC, NUL) values:

   S_1 $(4.8213, 21, 9)$
   S_2 $(2.84269, 20, 8)$
   S_3 $(3.24488, 21, 10)$
   S_4 $(3.59829, 21, 7)$
   S_5 $(2.67229, 17, 7)$
   S_6 $(2.32255, 18, 8)$
   S_7 $(2.35034, 27, 9)$
   S_8 $(2.35034, 23, 10)$
   S_9 $(2.42047, 18, 3)$
   S_{10} $(2.67229, 21, 3)$

2. Assume that the following five offspring are generated:

   S_{11} $(2.94400, 18, 4)$

S_{12} $(4.20384, 16, 2)$
S_{13} $(2.03553, 22, 11)$
S_{14} $(5.04003, 15, 2)$
S_{15} $(3.24220, 10, 4)$

3. NSGA-II employs the elitist approach in selecting the population. The initial population and offspring solutions are combined and the best 10 solutions are selected from this combined population. The next step ranks the combined population based on non-dominance. Let n_D be the number of non-dominated solutions and S_D be the set of dominated solutions. Solution S_1 is dominated by nine solutions, $S_2, S_4, S_5, S_6, S_9, S_{10}, S_{11}, S_{12}$, and S_{15}. Upon further observation, solution S_1 does not dominate any other solution. It is worth noting that solutions S_3, S_7, S_8, S_{13}, and S_{14} are neither dominating nor dominated by S_1. The same task is repeated for each solution and the following matrix is obtained in the format (solution, n_D, S_D):

$(S_1, 9, \{\})$
$(S_2, 3, \{S_1, S_3\})$
$(S_3, 7, \{\})$
$(S_4, 5, \{S_1\})$
$(S_5, 0, \{S_1, S_2, S_3, S_4\})$
$(S_6, 0, \{S_1, S_2, S_3, S_7, S_8\})$
$(S_7, 1, \{\})$
$(S_8, 1, \{\})$
$(S_9, 0, \{S_1, S_2, S_3, S_4, S_{10}, S_{11}\})$
$(S_{10}, 1, \{S_1, S_3, S_4\})$
$(S_{11}, 1, \{S_1, S_3, S_4\})$
$(S_{12}, 0, \{S_1\})$
$(S_{13}, 0, \{\})$
$(S_{14}, 0, \{\})$

$(S_{15}, 0, \{S_1, S_3, S_4\})$

4. The next step is to categorize the solutions into non-dominating fronts ($F_1, F_2, ..., F_{n_F}$). Every solution with an n_D value of 0 is assigned to front F_1. Thus, $F_1 = \{S_5, S_6, S_9, S_{12}, S_{13}, S_{14}, S_{15}\}$.

5. Now the next step is to determine the second front, F_2. Mark all the solutions present in F_1. Take the first solution from F_1 i.e, S_5. S_5 has S_D as $\{S_1, S_2, S_3, S_4\}$. Reduce n_D values from solutions S_1, S_2, S_3 and S_4 by 1. Thus the following matrix is obtained:

$(S_1, 8, \{\})$
$(S_2, 2, \{S_1, S_3\})$
$(S_3, 6, \{\})$
$(S_4, 4, \{S_1\})$
$(S_5^{(1)}, 0, \{S_1, S_2, S_3, S_4\})$
$(S_6^{(1)}, 0, \{S_1, S_2, S_3, S_7, S_8\})$
$(S_7, 1, \{\})$
$(S_8, 1, \{\})$
$(S_9^{(1)}, 0, \{S_1, S_2, S_3, S_4, S_{10}, S_{11}\})$
$(S_{10}, 1, \{S_1, S_3, S_4\})$
$(S_{11}, 1, \{S_1, S_3, S_4\})$
$(S_{12}^{(1)}, 0, \{S_1\})$
$(S_{13}^{(1)}, 0, \{\})$
$(S_{14}^{(1)}, 0, \{\})$
$(S_{15}^{(1)}, 0, \{S_1, S_3, S_4\})$

6. Repeat the above step for other solutions of F_1 i.e. $S_6, S_9, S_{12}, S_{13}, S_{14}$ and S_{15}. During the process, if n_D of any solution becomes 0 then it is assigned to the current front F_2. The following matrix is obtained:

$(S_1, 4, \{\})$
$(S_2^{(2)}, 0, \{S_1, S_3\})$
$(S_3, 3, \{\})$
$(S_4, 2, \{S_1\})$
$(S_5^{(1)}, 0, \{S_1, S_2, S_3, S_4\})$
$(S_6^{(1)}, 0, \{S_1, S_2, S_3, S_7, S_8\})$
$(S_7^{(2)}, 0, \{\})$
$(S_8^{(2)}, 0, \{\})$
$(S_9^{(1)}, 0, \{S_1, S_2, S_3, S_4, S_{10}, S_{11}\})$
$(S_{10}^{(2)}, 0, \{S_1, S_3, S_4\})$
$(S_{11}^{(2)}, 0, \{S_1, S_3, S_4\})$
$(S_{12}^{(1)}, 0, \{S_1\})$
$(S_{13}^{(1)}, 0, \{\})$
$(S_{14}^{(1)}, 0, \{\})$
$(S_{15}^{(1)}, 0, \{S_1, S_3, S_4\})$

Thus, $F_2 = \{S_2, S_7, S_8, S_{10}, S_{11}\}$.

7. The above process is repeated to produce four fronts. These fronts are then ranked based on non-dominance. These fronts are $F_1 = \{S_5, S_6, S_9, S_{12}, S_{13}, S_{14}, S_{15}\}, F_2 = \{S_2, S_7, S_8, S_{10}, S_{11}\}, F_3 = \{S_3, S_4\}$ and $F_4 = \{S_1\}$.

8. The 10 best solutions are selected from these fronts. The seven solutions from F_1 are selected, and the remaining three solutions are selected from F_2. The crowding distance, c_d, is calculated on front F_2 to choose the three solutions. The calculation of c_d is described in the subsequent steps.

9. The solutions of front F_2 are sorted in ascending order with respect to first objective MU:

S_7 $(2.35034, 27, 9)$
S_8 $(2.35034, 23, 10)$
S_{10} $(2.67229, 21, 3)$
S_2 $(2.84269, 20, 8)$
S_{11} $(2.944, 18, 4)$

10. Assign c_d as ∞ to the first and last solution. The crowding distance for the rest of the solutions in F_2 is calculated using $(MU_{previous} - MU_{next})/(MU_{last} - MU_{first})$ with respect to MU. The crowding distance of solution S_8 is $(2.67229 - 2.35034)/(2.944 - 2.35034) = 0.54231$. The following matrix in the format (solution, c_d) is obtained:

    S_7, ∞
    $S_8, 0.54231$
    $S_{10}, 0.82419$
    $S_2, 0.45768$
    S_{11}, ∞

11. As for the above step, the calculated crowding distance for front F_2 with respect to the second objective, NOC, after sorting are as follows:

    S_{11}, ∞
    $S_2, 0.33333$
    $S_{10}, 0.33333$
    $S_8, 0.66666$
    S_7, ∞

12. Similarly, the calculated crowding distance for front F_2 with respect to the third objective, NUL, after sorting are as follows:

    S_{10}, ∞

$S_{11}, 0.71428$
$S_2, 0.71428$
$S_7, 0.28571$
S_8, ∞

13. All the crowding distances are added as follows:

$S_2, 0.45768 + 0.33333 + 0.71428 = 1.50529$
$S_7, \infty + \infty + 0.28571 = \infty$
$S_8, 0.54231 + 0.66666 + \infty = \infty$
$S_{10}, 0.82419 + 0.33333 + \infty = \infty$
$S_{11}, \infty + \infty + 0.71428 = \infty$

14. The solutions with largest crowding distances are picked. Because the four solutions S_7, S_8, S_{10}, and S_{11} have ∞ any three solutions are randomly picked. Assume that S_7, S_8, and S_{11} are selected. Thus, the 10 solutions picked from the combined population (old population and new offspring) are $\{S_5, S_6, S_7, S_8, S_9, S_{11}, S_{12}, S_{13}, S_{14}, S_{15}\}$. The parent population is replaced by these solutions. Parent solutions are selected from this new population and the simple crossover [61] operator is applied to generate offspring. A mutation operator is then applied to the offspring. Likewise, the algorithm continues to proceed for the next iteration.

7.5.1 Control Parameter Tuning for Non-dominating Sorting Genetic Algorithm II

Similar to PDPSO, NSGA-II is also a dominance-based approach and generates a set of non-dominated solutions. No scalarized cost function is therefore available to evaluate the current solution in each iteration. The MOO hypervolume performance measure (discussed in Section 2.4) is used to evaluate the performance of NSGA-II in this thesis.

The best swarm size found for the FPSO was set as the initial population size for NSGA-II with respect to each test case. The best V_{max} values found for the FPSO were taken as the mutation rate in NSGA-II to match the level of perturbations between FPSO and NSGA-II. Further experiments were done to determine the best values for the crossover rate. The values of $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ were evaluated for the crossover rate. The crossover rate that resulted in the best quality solutions with respect to the hypervolume metric for NSGA-II are tabulated in Table 7.11 for all the test cases.

7.6 Conclusion

This chapter presented five population based algorithms for the OSPFWS problem. These algorithms are FPSO, FEPSO, WAPSO, PDPSO and NSGA-II. The fuzzy logic based objective function, FuzzyCF, was incorporated into the FPSO and FEPSO

Table 7.11: Best parameter combinations obtained for NSGA-II for each test case.

Test Case	Population size	V_{max}	crossover rate	hypervolume
h100N280a	40	5	0.7	6.004± 0.184
h100N360a	100	15	0.7	2.517± 0.152
h50N148a	40	5	0.6	2.812± 0.145
h50N212a	100	15	0.8	3.629± 0.112
r100N403a	100	15	0.8	3.181± 0.114
r100N503a	100	15	0.8	3.03± 0.145
r50N228a	100	15	0.7	4.611± 0.254
r50N245a	40	5	0.5	3.101± 0.212
w100N391a	100	15	0.8	4.893± 0.211
w100N476a	100	15	0.6	1.79± 0.124
w50N169a	40	5	0.8	2.765± 0.085
w50N230a	40	5	0.8	4.857± 0.272

algorithms. A weighted sum (or aggregation) cost function is used for WAPSO. The PDPSO and NSGA-II algorithms are pareto-dominant based algorithms. The control parameters were tuned for all the algorithms presented in this chapter. The next chapter evaluates the performance of all the algorithms presented in this thesis by using the obtained results from this chapter.

Chapter 8

Empirical Comparison of the Algorithms

Chapters 5 to 7 presented a number of optimization algorithms as applied to the OSPFWS problem. This chapter presents an overall comparison of the results obtained for each of the proposed algorithms. The proposed algorithms are grouped into two categories: single-solution based (FSA and FSimE) and population based (fuzzy PSO, variants of PSO, and NSGA-II). The chapter first presents a comparison of the SA and SimE algorithms with respect to the FuzzyCF and SqalliCF cost functions. The next comparison is done between FPSO and FEPSO. This is followed by a comprehensive comparison of all algorithms, including a comparison with a state-of-the-art multi-objective genetic algorithm, NSGA-II.

The outline of the chapter is as follows: Section 8.1 specifies the experimental setup used for this chapter. A comparison of SA and SimE is done in Section 8.2. Section 8.3 discusses the performance between FPSO and FEPSO. All the implemented algorithms are compared with respect to diversity in Section 8.4 and with respect to the MOO performance measures in Section 8.5.

8.1 Experimental Setup

The results of FSA and FSimE presented in Chapters 5 and 6 are used in this chapter, along with the results of FPSO, FEPSO, WAPSO, PDPSO and NSGA-II presented in Chapter 7. Thirty independent runs were performed. The Wilcoxon rank-sum test [65] was used to validate the significance of the results. A confidence level of 95% was used.

8.2 Comparison of Simulated Annealing and Simulated Evolution Algorithms

A comparative study of SA and SimE with respect to the three design objectives using the SqalliCF and FuzzyCF functions was performed. Figure 8.1 shows different plots with respect to MU, NOC, and NUL. Figures 8.1(a) and 8.1(b) compare SimE and

SA with respect to both cost functions (FuzzyCF and SqalliCF) for the objective MU. SimE performed significantly better than SA for eight test cases using the FuzzyCF. For the remaining four test cases (h100N280a, h50N148a, h50N212a and w50N169a), there was no statistically significant difference in performance. In the case of SqalliCF, SimE performed significantly better than SA for seven test cases. For the other five cases (h100N280a, h50N148a, w100N391a, w50N169a and w50N230a), there was no statistically significant difference in performance. Thus, with reference to MU, SimE outperformed SA in the majority of test cases with respect to both cost functions. Results for the statistical tests are given in Table 8.1.

With reference to NOC, Figure 8.1(c) shows that SimE performed statistically significantly better than SA for eight test cases (h100N360a, r100N403a, r100N503a, r50N228a, r50N245a, w100N391a, w100N476a and w50N230a) with respect to FuzzyCF. For the other four cases (h50N148a, h100N280a, h50N212a and w50N169a), the performance of SimE and SA was comparable. Figure 8.1(d) shows that for seven test cases (h50N212a, r50N228a, r50N245a, r100N403a, r100N503a, w50N230a and w100N476a), SimE performed statistically significantly better than SA with respect to SqalliCF. For the other five test cases (h100N280a, h50N148a, h100N360a, w50N169a and w100N391a), there was no statistically significant difference in performance. Thus, SimE outperformed SA for the majority of the test cases for both the cost functions with respect to NOC. Results of the statistical tests are provided in Ta-

ble 8.2.

Finally, for the third objective, NUL, SimE performed significantly better than SA for four test cases (r100N503a, w100N391a, w100N476a and w50N230a) with respect to FuzzyCF as shown in Figure 8.1(e). SA performed significantly better than SimE for two test cases (h100N280a and h50N212a). For the rest of the test cases, there was no significant difference in performance. Figure 8.1(f) shows that SimE performed better than SA for six test cases (r100N403a, r100N503a, r50N228a, r50N245a, w100N476a, and w50N230a) with respect to SqalliCF. SA performed significantly better than SimE in one test case (h100N280a). For the rest of the test cases, the performance was comparable. Results of the statistical tests are given in Table 8.3.

Table 8.1: Statistical results of SimE vs SA with respect to FuzzyCF and SqalliCF cost functions for the objective MU.

Test case	FuzzyCF (for MU)		SqalliCF (for MU)	
	SimE vs SA (% diff.)	p-value	SimE vs SA (% diff.)	p-value
h100N280a	6.38	0.329	1.49	0.342
h100N360a	16.86	0.031	16.9	0.011
h50N148a	-6.79	0.234	-1.41	0.154
h50N212a	0.6	0.112	11.36	0.017
r100N403a	46.24	0.035	34.03	0.043
r100N503a	63.47	0.013	31.91	0.018
r50N228a	12.22	0.008	16.18	0.028
r50N245a	8.85	0.004	4.67	0.025
w100N391a	23.24	0.044	2.13	0.132
w100N476a	53.42	0.028	7.58	0.014
w50N169a	-2.08	0.255	0.79	0.285
w50N230a	7.64	0.014	0	0

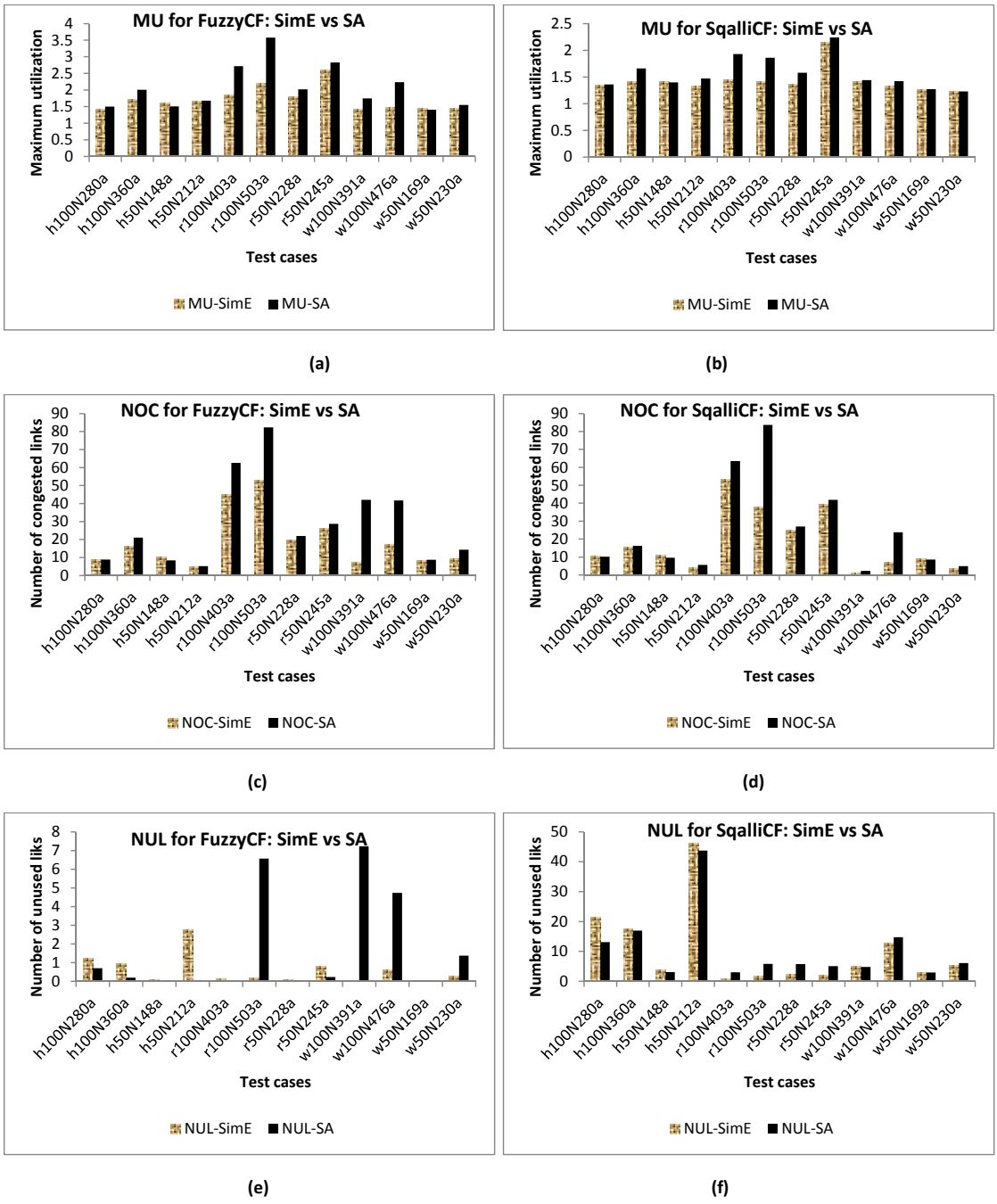


Figure 8.1: Comparison of SimE and SA using SqalliCF and FuzzyCF functions. (a) MU for FuzzyCF: SimE vs SA (b)MU for SqalliCF: SimE vs SA (c) NOC for FuzzyCF: SimE vs SA (d) NOC for SqalliCF: SimE vs SA (e) NUL for FuzzyCF: SimE vs SA (f) NUL for SqalliCF: SimE vs SA.

Table 8.2: Statistical results of SimE vs SA with respect to FuzzyCF and SqalliCF cost functions for the objective NOC.

Test case	FuzzyCF (for NOC)		SqalliCF (for NOC)	
	SimE vs SA (% diff.)	p-value	SimE vs SA (% diff.)	p-value
h100N280a	5.06	0.234	-5.22	0.322
h100N360a	30.38	0.036	3.65	0.314
h50N148a	-19	0.129	-11.89	0.136
h50N212a	4.87	0.143	40.75	0.003
r100N403a	39.95	0.027	19.52	0.021
r100N503a	55.93	0.042	121.34	0.035
r50N228a	11.43	0.012	8.62	0.012
r50N245a	9.81	0.039	6.71	0.042
w100N391a	487.17	0.028	111.82	0.176
w100N476a	144.29	0.007	237.06	0.029
w50N169a	5.14	0.072	-8.22	0.232
w50N230a	57.72	0.02	33.51	0.021

The overall better performance of SimE compared to SA is attributed to the fact that the search in SA is carried out blindly, and moves are done randomly. This may result in replacing an optimally placed weight on a link with a weight resulting in a lower quality solution. However, in SimE, perturbations to a solution are done on the basis of goodness, i.e. moves are done intelligently, rather than blindly. Figures 8.2, 8.3, 8.4 and 8.5 show the plots of average goodness of weights for all the test cases with respect to FSA and FSimE. For all the plots, FSA shows a random trend for the average goodness of weights. The corresponding FSimE plots show an increase in average goodness of weights after a few initial iterations. Once the weights settled to optimum or near-optimum values (evaluated by goodness),

Table 8.3: Statistical results of SimE vs SA with respect to FuzzyCF and SqalliCF cost functions for the objective NUL.

Test case	FuzzyCF (for NUL)		SqalliCF (for NUL)	
	SimE vs SA (% diff.)	p-value	SimE vs SA (% diff.)	p-value
h100N280a	-43.09	0.038	-38.44	0.037
h100N360a	-78.49	0.139	-4.19	0.113
h50N148a	-100	0.223	-16.35	0.075
h50N212a	-97.47	0.035	-5.14	0.141
r100N403a	-76.92	0.122	261.45	0.033
r100N503a	3764.71	0.043	260.12	0.013
r50N228a	-70	0.252	163.64	0.025
r50N245a	-71.25	0.307	151.23	0.009
w100N391a	24000	0.023	-4.57	0.126
w100N476a	688.33	0.028	16.63	0.046
w50N169a	-100	0.225	4.64	0.174
w50N230a	407.41	0.022	16.32	0.012

FSimE did not replace these settled weights. This is not the case with FSA, where weights are replaced blindly irrespective of their goodness. Thus, weights with high goodness have a lower probability of being removed, while weights with low goodness are more prone to being replaced. Hence, FSimE performed more intelligently than FSA, resulting in higher quality solutions.

As far as time complexity is concerned, the time complexity of FSA is $O(|A|) + O(I)$, where $|A|$ is total number of arcs and I is number of iterations. The first term, $O(|A|)$, represents the time required to initialize a random single solution and the second term, $O(I)$, represents the time required for performing the search. Contrary to this, FSimE has an extra step of goodness evaluation of the weights on all the

edges. Because of this, the time complexity of FSimE is $O(|A|) + O(|A| * I)$. Hence, a single iteration of FSimE would take more time than a single iteration of FSA.

8.3 Comparison of Fuzzy Particle Swarm Optimization and Fuzzy Evolutionary Particle Swarm Optimization

The purpose of this section is to compare the performance of the FPSO and FEPSO algorithms. Experiments were conducted using the best parameter combinations found for both the algorithms (refer to Tables 7.6 and 7.7). Table 8.4 summarizes the results of the comparison with respect to fuzzy cost function. It is clearly observed from the table that the improvements achieved by FEPSO were statistically significant for all test cases, with the exception of h100N280a (for which FEPSO had a slightly inferior performance with a degradation of 0.91% in the average fitness). However, this difference is not significant. Therefore, it can be confidently claimed that FEPSO outperformed FPSO in terms of the quality of solutions with respect to fuzzy cost function.

Although the proposed hybridization between FPSO and SimE seems promising in generating high quality results, the approach has some limitations. One major issue is

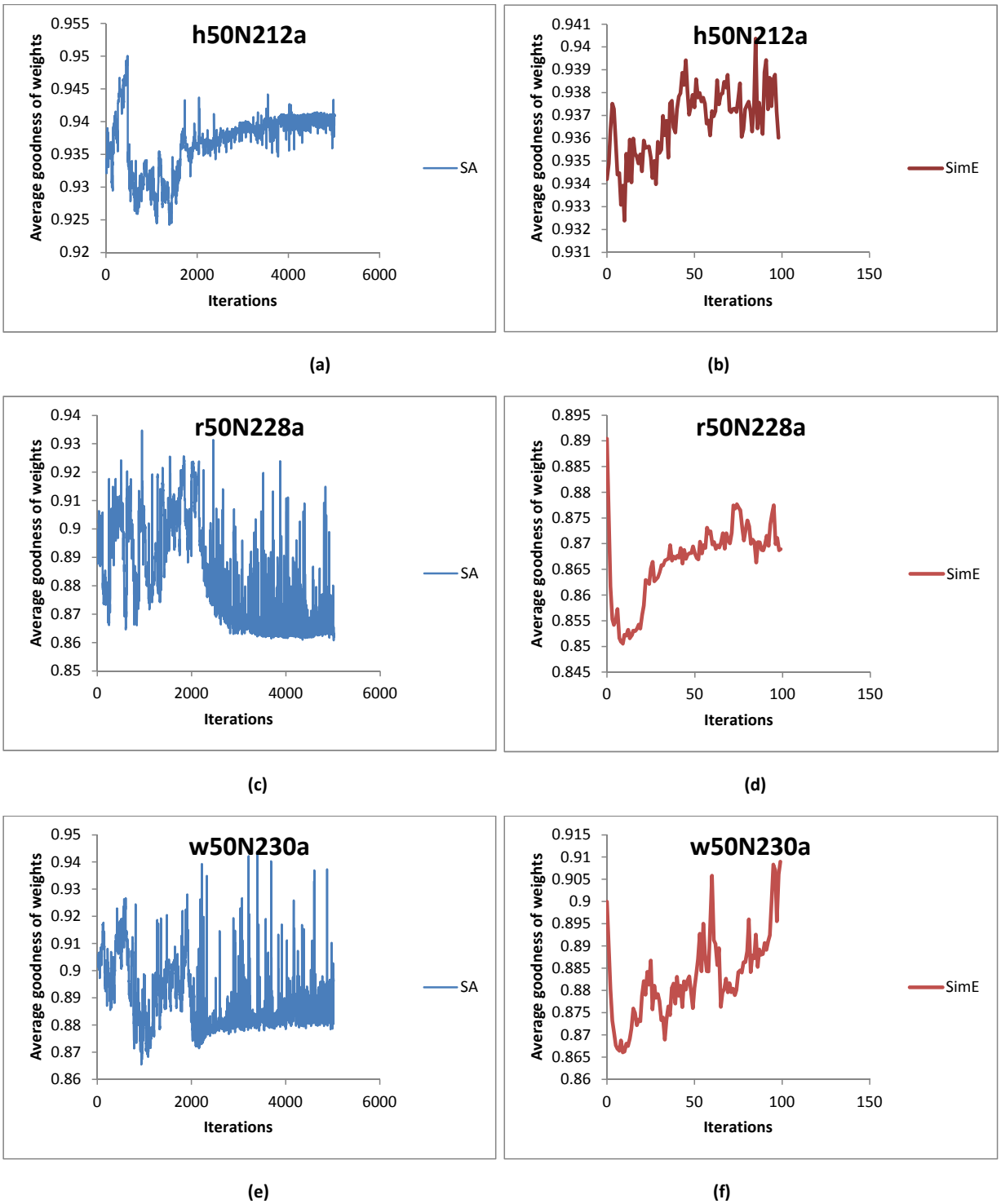


Figure 8.2: Plots of average goodness of weights for three sample test cases of size 50 with respect to FSA and FSimE. (a) h50N212a-FSA, (b) h50N212a-FSimE, (c) r50N228a-FSA, (d) r50N228a-FSimE, (e) w50N230a-FSA, and (f) w50N230a-FSimE.

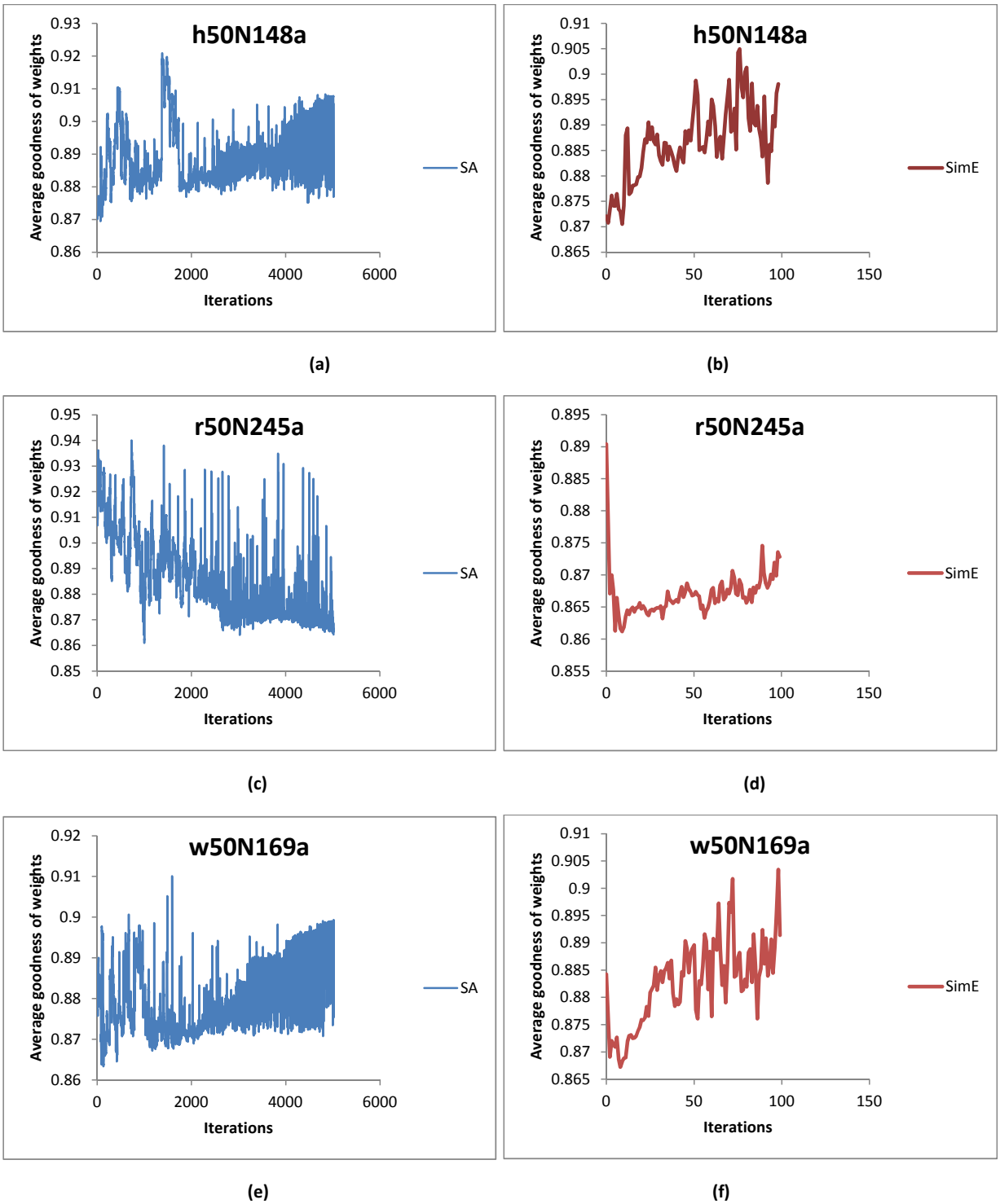


Figure 8.3: Plots of average goodness of weights for three sample test cases of size 50 with respect to FSA and FSimE. (a) h50N148a-FSA, (b) h50N148a-FSimE, (c) r50N245a-FSA, (d) r50N245a-FSimE, (e) w50N169a-FSA, and (f) w50N169a-FSimE.

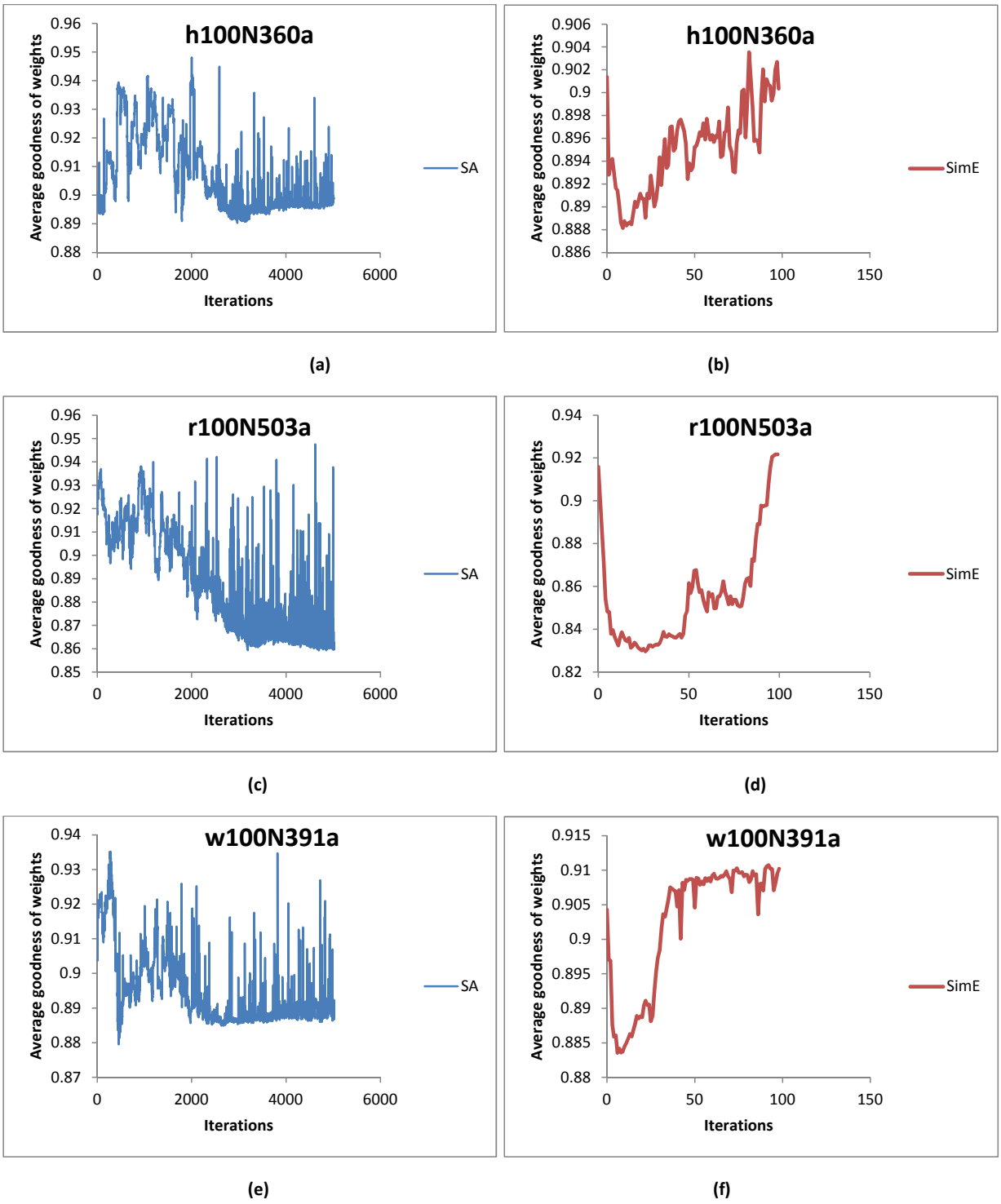


Figure 8.4: Plots of average goodness of weights for three sample test cases of size 100 with respect to FSA and FSimE. (a) h100N360a-FSA, (b) h100N360a-FSimE, (c) r100N503a-FSA, (d) r100N503a-FSimE, (e) w100N391a-FSA, and (f) w100N391a-FSimE.

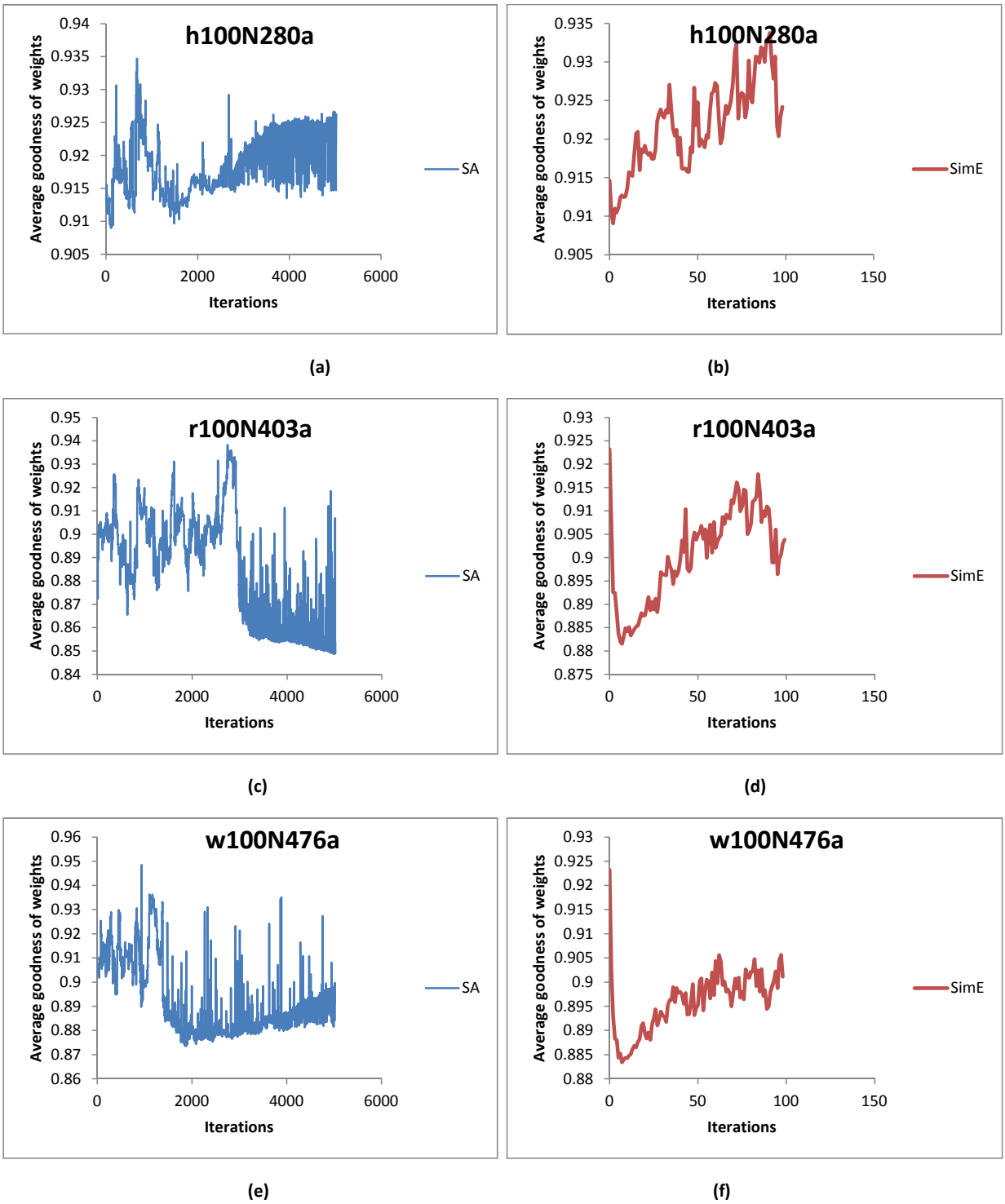


Figure 8.5: Plots of average goodness of weights for three sample test cases of size 100 with respect to FSA and FSimE. (a) h100N280a-FSA, (b) h100N280a-FSimE, (c) r100N403a-FSA, (d) r100N403a-FSimE, (e) w100N476a-FSA, and (f) w100N476a-FSimE.

Table 8.4: Comparison of FPSO and FEPSO with respect to fuzzy cost. Significant differences are highlighted in boldface. % Imp = percentage improvement.

Test Case	FPSO fuzzy cost	FEPSO fuzzy cost	% Imp	p-values
h100N280a	0.531±0.018	0.526±0.015	-0.91	0.323
h100N360a	0.543±0.036	0.605±0.012	**11.40**	0.042
h50N148a	0.437±0.021	0.469±0.019	**7.22**	0.021
h50N212a	0.504±0.015	0.528±0.013	**4.94**	0.023
r100N403a	0.481±0.017	0.595±0.011	**23.73**	0.011
r100N503a	0.543±0.013	0.710±0.012	**30.83**	0.033
r50N228a	0.543±0.019	0.610±0.016	**12.27**	0.003
r50N245a	0.557±0.019	0.644±0.014	**15.67**	0.007
w100N391a	0.609±0.029	0.725±0.010	**18.94**	0.017
w100N476a	0.657±0.025	0.757±0.011	**15.24**	0.039
w50N169a	0.595±0.021	0.640±0.012	**7.47**	0.043
w50N230a	0.591±0.029	0.711±0.022	**20.48**	0.015

the need for tuning an additional parameter, namely, the *bias (B)*, in addition to the parameters of the FPSO algorithm. This adds extra effort and time to find the best combination out of many possible combinations of all design parameters. Another issue is that the complexity of the algorithm increases, thus increasing the execution time. By considering Algorithm 2.5, If n_s is the swarm size, and $|A|$ is the total number of arcs in the network, then the time complexity of steps 1 to 5 is $O(n_s * |A|)$. The time complexity of the rest of the steps 6 to 16 is $O(n_s * I)$, where I is the number of iterations the algorithm runs. Thus, the total time complexity of FPSO is $O(n_s * |A|) + O(n_s * I)$. Contrary to this, FEPSO has an extra step of goodness evaluation for each existing weight in the set $V_i(t+1)$ as given in Algorithm 7.1. Because of this, the time complexity of FEPSO is $O(n_s * |A|) + O(n_s * |V_i(t+1)| * I)$. Thus, a single iteration of FEPSO will take more time than a single iteration of FPSO.

The proposed hybridization will allow faster convergence of FEPSO to an optimal or sub-optimal solution and will produce higher quality solutions as compared to FPSO. This is confirmed by the graphs shown in Figures 8.6 and 8.7.

Figures 8.8 and 8.9 show the plots of the average number of selected weights associated with links for replacement with respect to FEPSO and FPSO. All of these figures indicate that FEPSO selected fewer weights than FPSO. This shows that FEPSO did not select optimum or near-optimum position weights (evaluated by goodness) for replacement. This is not the case for FPSO, where weights are selected for replacement irrespective of their goodness. The average goodness of weights associated with links for FPSO and FEPSO is tabulated in Table 8.5. FEPSO performed statistically significantly better than FPSO for 10 test cases. For the other two test cases, h50N212a and w50N169a, the better performance of FEPSO is statistically insignificant.

8.4 Comparison of all the algorithms with respect to diversity

The purpose of this section is to understand the exploration and exploitation tradeoff of all the algorithms. Because FSA and FSimE are not population based algorithms but single solution algorithms, they are excluded from this comparison. Diversity is described as the variation amongst the candidate solutions (or particles w.r.t PSO) in

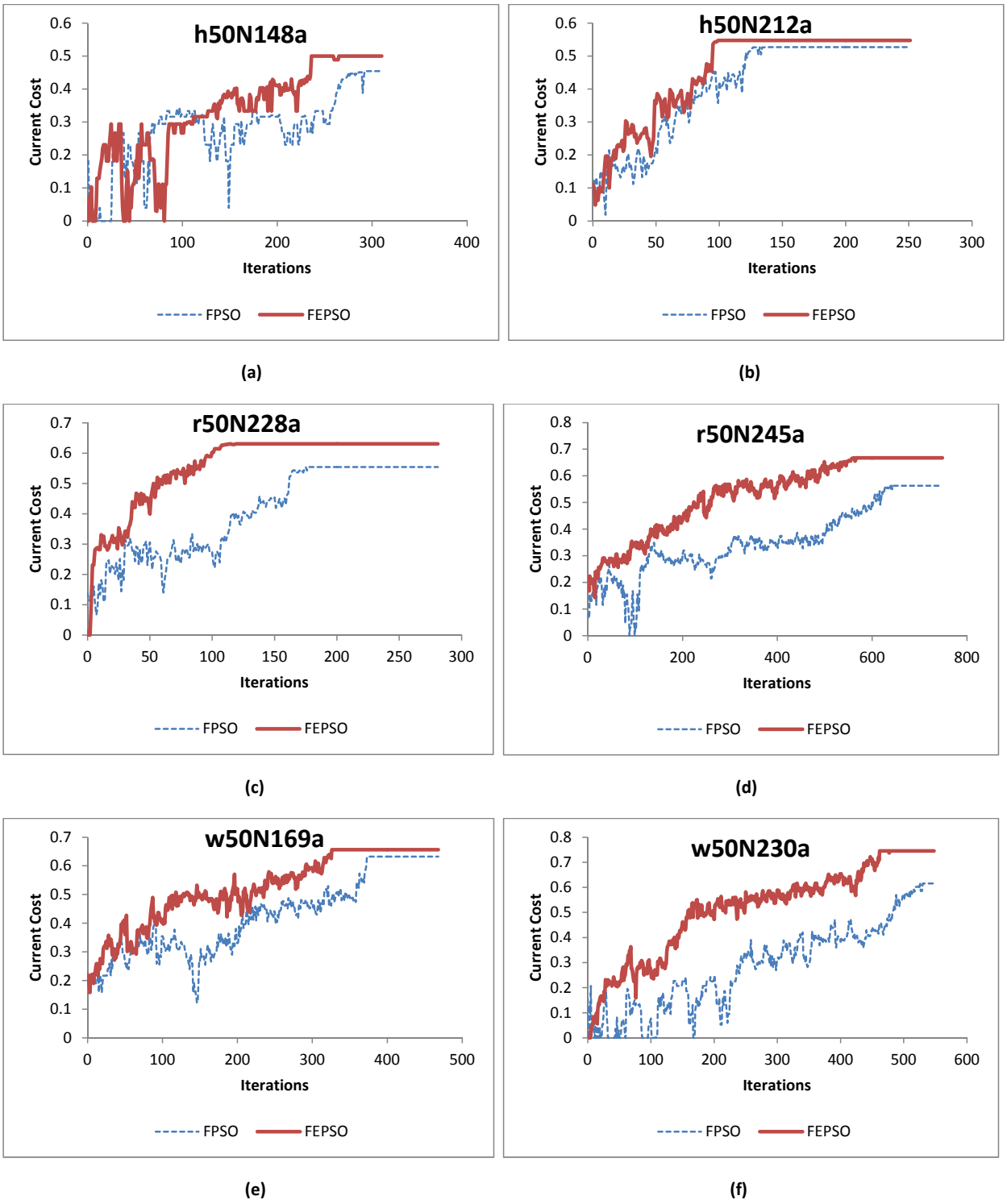


Figure 8.6: Plots of average current cost of FEPSO and FPSO for all the test cases with 50 nodes.

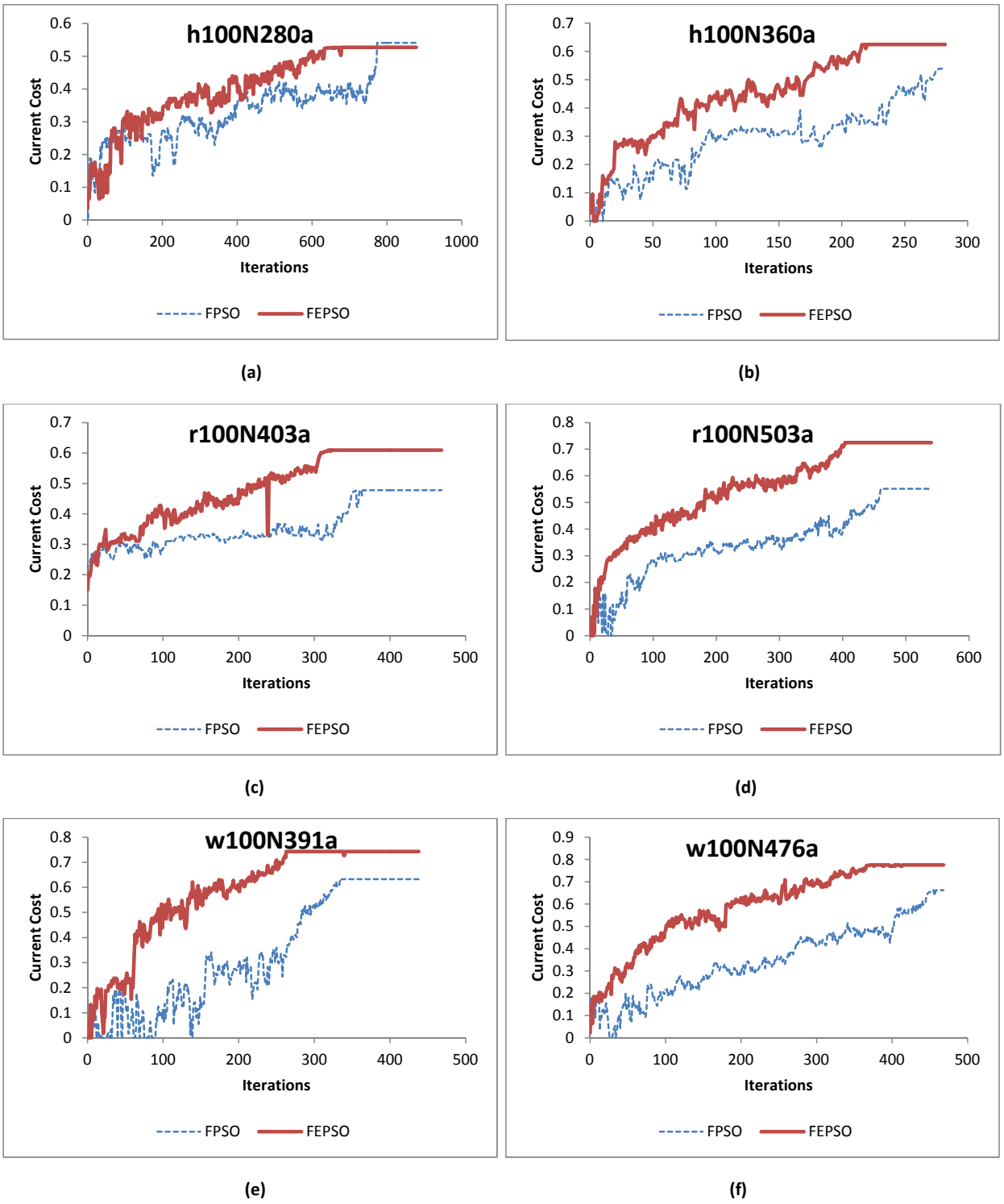


Figure 8.7: Plots of average current cost of FEPSO and FPSO for all the test cases with 100 nodes.

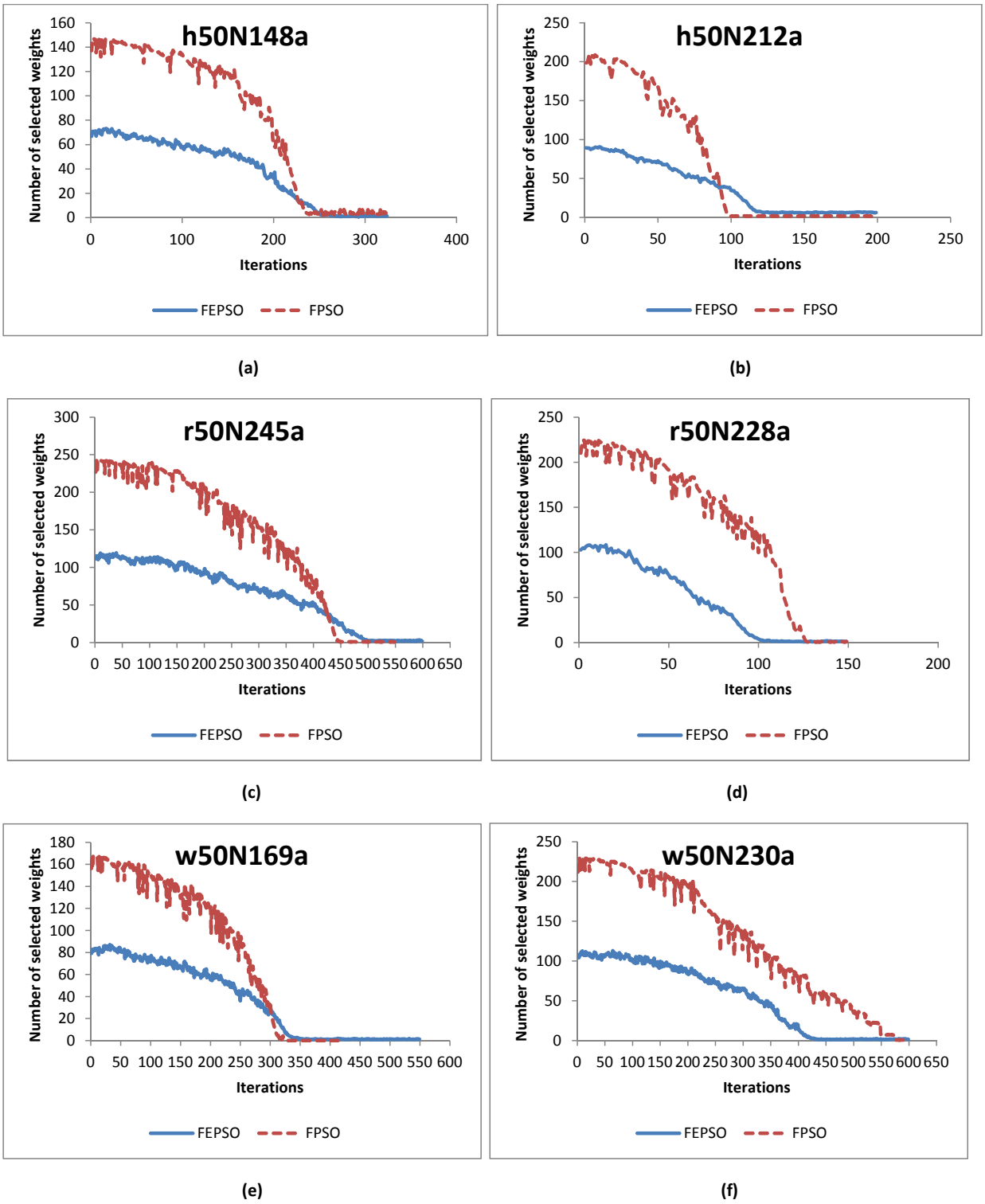


Figure 8.8: Plots of average number of selected weights associated with links of FEPSO and FPSO for all the test cases with 50 nodes.

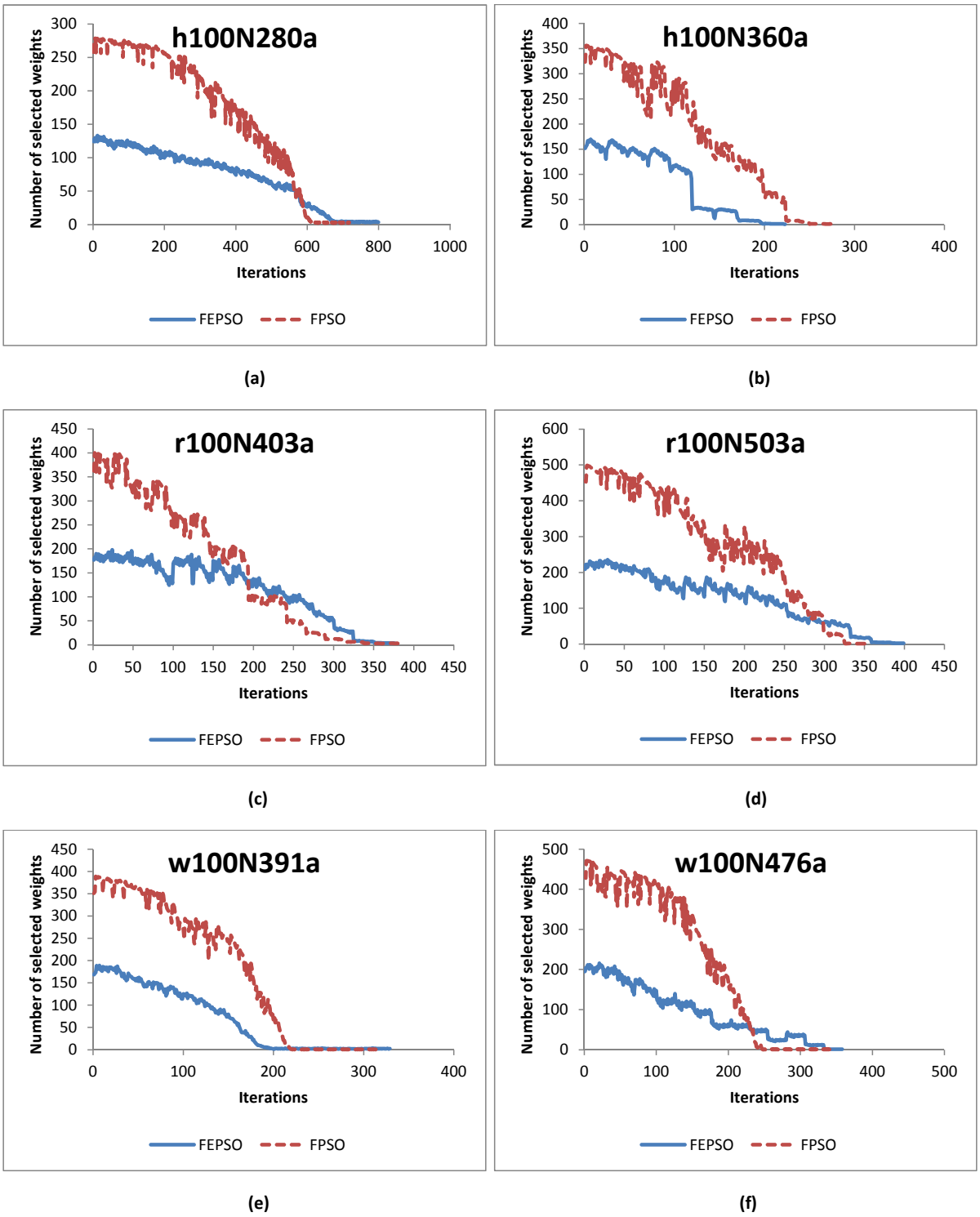


Figure 8.9: Plots of average number of selected weights associated with links of FEPSO and FPSO for all the test cases with 100 nodes.

Table 8.5: Comparison of FPSO and FEPSO with respect to average goodness of weights associated with links. Significant differences are highlighted in boldface. % Imp = percentage improvement.

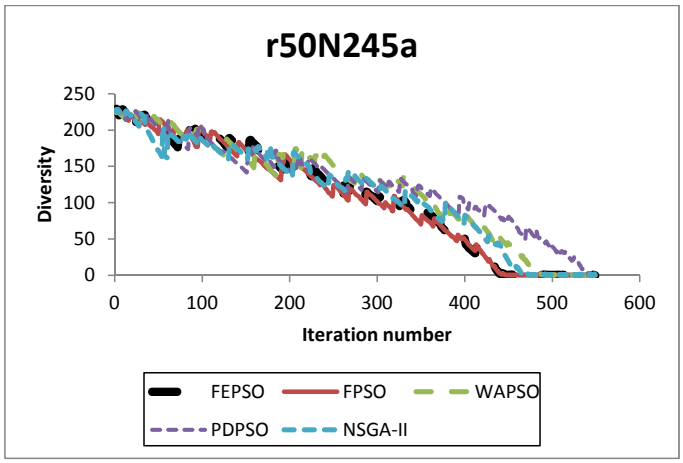
Test Case	FPSO average goodness	FEPSO average goodness	% Imp	p-values
h100N280a	0.886±0.018	0.929±0.017	**4.629**	0.017
h100N360a	0.894±0.013	0.928±0.018	**3.664**	0.012
h50N148a	0.891±0.027	0.921±0.016	**3.257**	0.022
h50N212a	0.938±0.054	0.951±0.067	1.367	0.066
r100N403a	0.862±0.082	0.937±0.083	**8.004**	0.019
r100N503a	0.877±0.074	0.933±0.036	**6.002**	0.006
r50N228a	0.865±0.042	0.928±0.061	**6.789**	0.012
r50N245a	0.873±0.013	0.932±0.067	**6.331**	0.024
w100N391a	0.884±0.062	0.948±0.061	**6.751**	0.011
w100N476a	0.891±0.043	0.935±0.072	**4.706**	0.016
w50N169a	0.892±0.015	0.911±0.021	2.086	0.185
w50N230a	0.885±0.023	0.933±0.012	**5.145**	0.026

the population [9]. The calculation of swarm diversity is explained subsequently with an example. Assume three solutions, $S_1 = \{18, 1, 7, 15, 3, 17, 14, 19, 13, 18, 4, 16, 16\}$, $S_2 = \{18, 1, 14, 15, 3, 17, 15, 19, 13, 18, 4, 16, 16\}$ and $S_3 = \{18, 1, 7, 15, 3, 17, 14, 19, 11, 18, 4, 16, 9\}$, in an iteration for the topology given in Figure 4.1. Also assume that S_3 is the global best solution obtained in that iteration. By considering the global best solution in each iteration as reference, the difference in elements between S_1 and S_3 is 2. Similarly, the difference in elements between S_2 and S_3 is 4. Because S_3 is the global best solution, the difference between S_3 and the global best solution is 0. The swarm diversity is then calculated as the average of these obtained values (i.e, 2, 4 and 0) and is equal to 2. Figures 8.10, 8.11, 8.12 and 8.13 compares the swarm diversity with respect to the FPSO, FEPSO, WAPSO, PDPSO and NSGA-II algorithms for

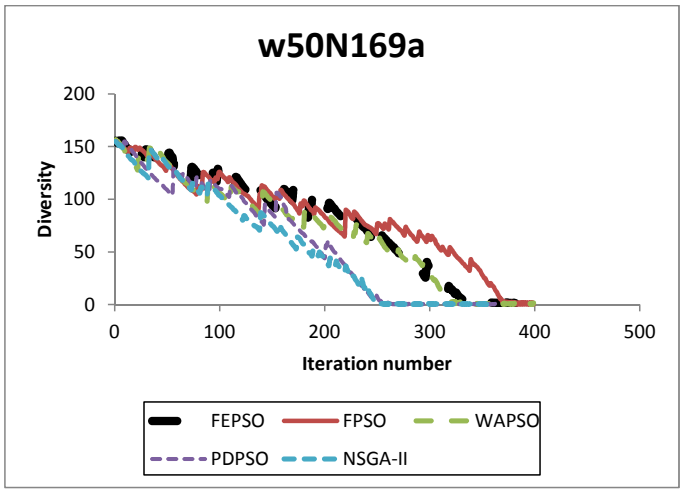
all the test cases. These figures were plotted by taking the average diversity over 30 independent runs. Similar exploration and exploitation trends with respect to all the algorithms were found for test cases r50N245a, w50N230a, h50N148a, and h100N280a. FEPSO exploited more than the other algorithms for the test case h100N360a. A better exploration ability of NSGA-II compared to the other algorithms was found for test cases h50N212a, h100N360a, and r100N503a. NSGA-II exploited more than the other algorithms for test case w50N169a. NSGA-II and WAPSO exploited more for test case r50N228a. FPSO explored better than the other algorithms for test cases r100N403a, w50N169a, and r50N228a. Based on the above observations, the diversity trends were found to be unpredictable across all the test cases for all the algorithms.

8.5 Comparison of all the algorithms with respect to MOO performance measures

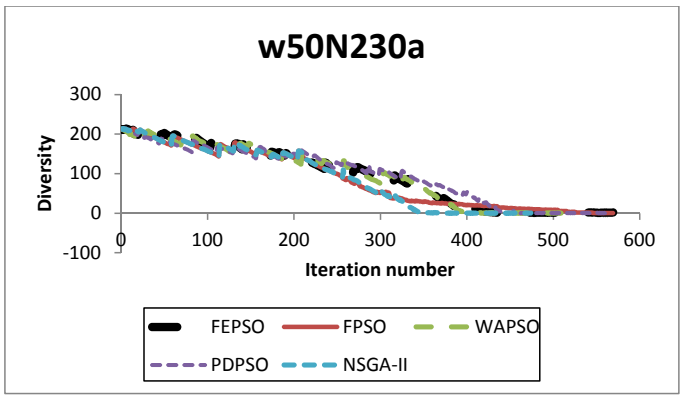
Three MOO performance measures, namely ONVG, spacing, and hypervolume were discussed in Section 2.4. Table 8.6 provides the average ONVG values for all the implemented algorithms over 30 independent runs. A comparison between FSA and FSimE with respect to ONVG is given in Table 8.7. FSA and FSimE are comparable in performance (i.e, no statistically significant differences were found) for six test cases (h50N212a, r100N403a, r50N245a, w100N391a, w100N476a, and w50N230a). For the



(a)

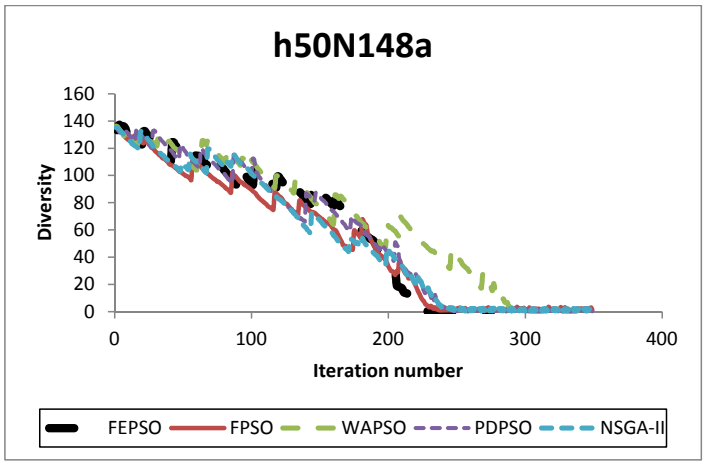


(b)

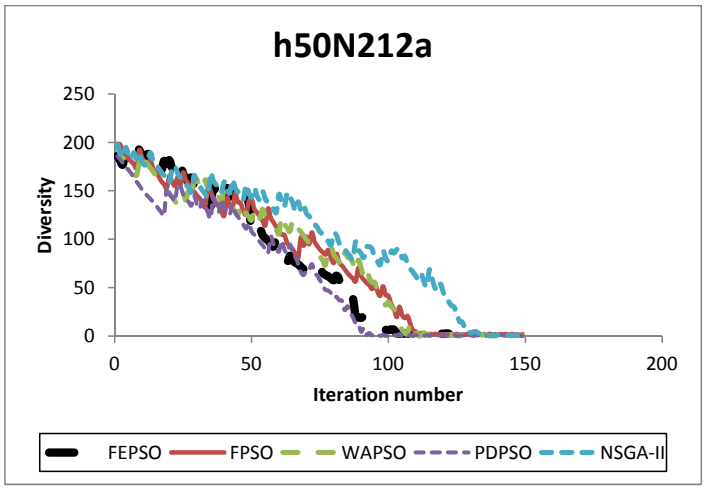


(c)

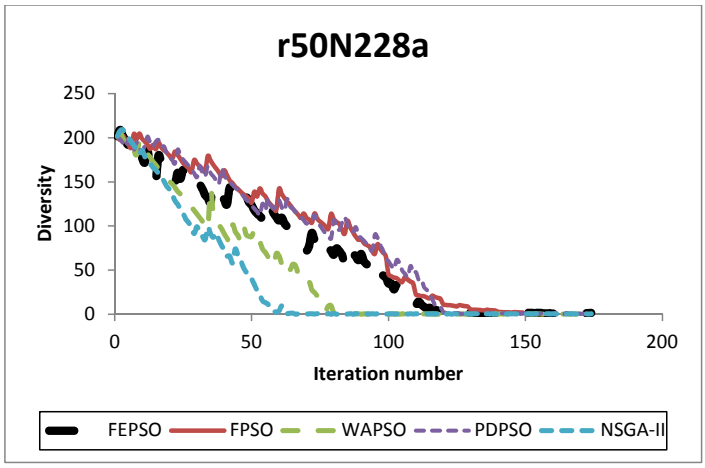
Figure 8.10: Swarm diversity plots with respect to FPSO, FEPSO, WAPSO, PDPSO and NSGA-II algorithms for the test cases (a) h50N148a, (b) h50N212a, and (c) r50N228a.



(a)

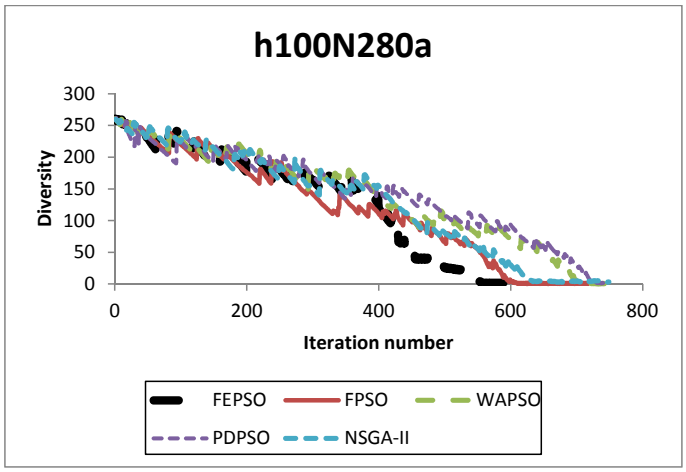


(b)

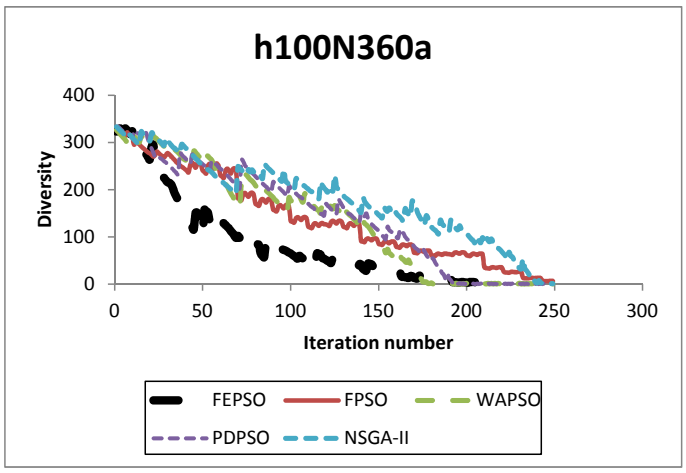


(c)

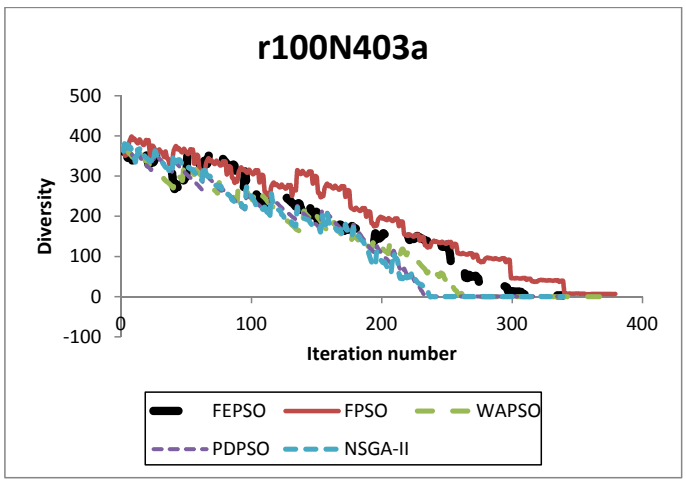
Figure 8.11: Swarm diversity plots with respect to FPSO, FEPSO, WAPSO, PDPSO and NSGA-II algorithms for the test cases (a) r50N245a, (b) w50N169a, and (c) w50N230a.



(a)

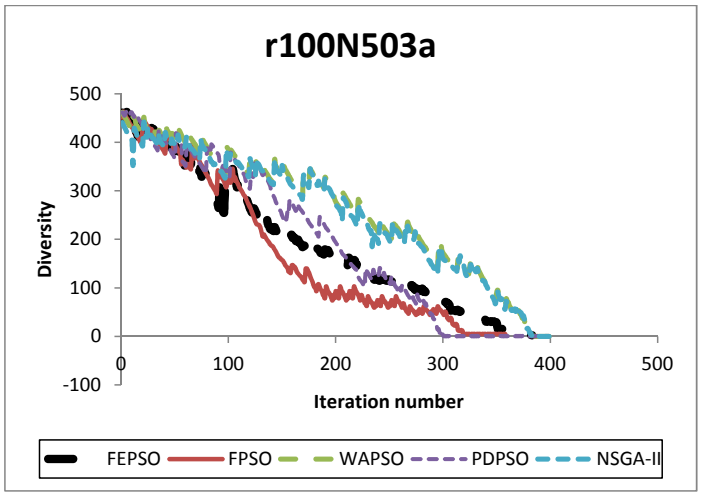


(b)

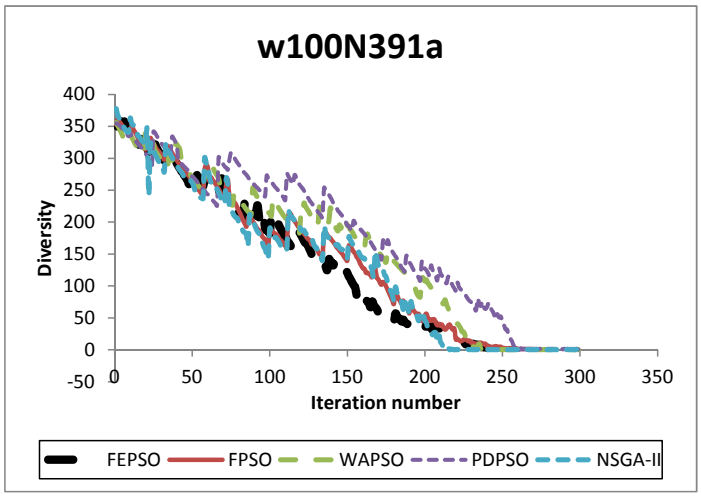


(c)

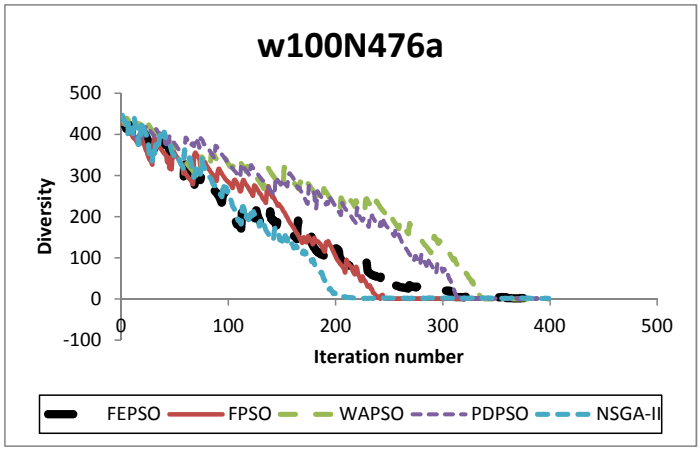
Figure 8.12: Swarm diversity plots with respect to FPSO, FEPSO, WAPSO, PDPSO and NSGA-II algorithms for the test cases (a) h100N280a, (b) h100N360a, and (c) r100N403a.



(a)



(b)



(c)

Figure 8.13: Swarm diversity plots with respect to FPSO, FEPSO, WAPSO, PDPSO and NSGA-II algorithms for the test cases (a) r100N503a, (b) w100N391a, and (c) w100N476a.

rest of the test cases, FSimE performed statistically significantly better than FSA. Summing the wins and losses of both the algorithms shows that FSimE performed better in the majority of the test cases. Hence, FSimE performed better than FSA with respect to the ONVG metric. The final winners are tabulated in Tables 8.8, 8.9 and 8.10 after following a similar process of comparison. Out of the 7 algorithms, FEPSO showed the best performance in all the comparisons with respect to ONVG metric.

Average spacing values are provided in Table 8.11 for all the implemented algorithms over 30 independent runs. A sample comparison between FSA and FSimE with respect to spacing is given in Table 8.12. No statistically significant difference was found in the performance between FSA and FSimE for two test cases (h50N212a and r50N228a). For the rest of the test cases, FSimE performed statistically significantly better than FSA. Summing the wins and losses of both the algorithms shows that FSimE performed better in the majority of the test cases. Hence, FSimE performed better than FSA with respect to the spacing metric. The final winners are tabulated in Tables 8.13, 8.14 and 8.15 after following a similar process of comparison. FSimE showed comparable performance to PDPSO. Out of all 7 algorithms, FEPSO showed the best performance in all the comparisons with respect to the spacing metric.

Table 8.16 provides the average hypervolume values and their standard deviations for the implemented algorithms over 30 independent runs. A sample comparison be-

Table 8.6: Average and standard deviation values of ONVG metric for all the algorithms, for all the test cases.

Test Case	FSA	FSimE	FPSO	FEPSO	WAPSO	PDPSO	NSGA-II
h100N280a	6.82± 1.48	5.83± 0.75	13.12± 3.24	15.21± 3.67	13.4± 3.43	14.83± 2.83	14.33± 3.21
h100N360a	5.75± 0.87	6.64± 1.43	9.17± 1.25	9.32± 1.83	7.22± 1.34	7.234± 2.52	8.154± 1.62
h50N148a	5.16± 0.65	7.143± 2.54	6.71± 1.52	5.4± 0.81	5.37± 0.87	6.43± 1.54	7.37± 1.56
h50N212a	8.46± 1.13	8.14± 1.54	11.48± 2.23	12.35± 1.23	12.66± 1.56	11.6± 1.65	10.63± 2.38
r100N403a	10.52± 2.4	9.57± 2.24	12.84± 1.78	13.12± 2.02	12.33± 1.47	11.342± 2.23	12.47± 2.78
r100N503a	5.65± 0.82	6.45± 1.26	10.23± 1.7	14.72± 1.67	9.41± 1.85	9.46± 1.23	11.51± 2.06
r50N228a	4.37± 0.74	6.42± 1.62	9.16± 1.56	10.634± 1.84	9.29± 1.81	8.62± 1.23	9.55± 1.83
r50N245a	10.74± 2.32	11.75± 1.93	15.25± 3.23	15.74± 3.82	14.63± 3.56	13.46± 2.25	12.75± 2.21
w100N391a	11.48± 1.89	10.81± 2.25	16.27± 3.56	13.37± 3.22	13.25± 3.25	17.76± 3.23	18.8± 3.52
w100N476a	7.84± 1.23	8.54± 1.03	10.67± 1.76	10.7± 2.33	11.62± 2.05	8.54± 1.56	9.41± 1.78
w50N169a	6.42± 2.32	5.51± 0.83	8.25± 1.34	7.55± 1.37	7.46± 1.66	4.62± 0.78	6.66± 1.53
w50N230a	8.34± 1.56	7.73± 2.65	14.51± 3.76	14.51± 3.13	12.74± 1.9	9.72± 1.85	10.07± 2.72

Table 8.7: Statistical results of FSA vs FSimE with respect to ONVG metric.

Test case	FSA	FSimE	FSA vs FSimE (%diff.)	p-value	FSA wins	FSA losses	FSimE wins	FSimE losses
h100N280a	6.82	5.83	14.52	0.036	1	0	0	1
h100N360a	5.75	6.64	-15.48	0.042	0	1	1	0
h50N148a	5.16	7.143	-38.43	0.013	0	1	1	0
h50N212a	8.46	8.14	3.78	0.266	1	0	1	0
r100N403a	10.52	9.57	9.03	0.111	1	0	1	0
r100N503a	5.65	6.45	-14.16	0.025	0	1	1	0
r50N228a	4.37	6.42	-46.91	0.013	0	1	1	0
r50N245a	10.74	11.75	-9.4	0.181	1	0	1	0
w100N391a	11.48	10.81	5.84	0.149	1	0	1	0
w100N476a	7.84	8.54	-8.93	0.118	1	0	1	0
w50N169a	6.42	5.51	14.17	0.031	1	0	0	1
w50N230a	8.34	7.73	7.31	0.134	1	0	1	0

Table 8.8: Comparison of algorithms with respect to ONVG metric.

Comparison of	FSA vs FSimE	FSA vs FPSO	FSA vs FEPSO	FSA vs WAPSO	FSA vs PDPSO	FSA vs NSGA-II	FSimE vs FPSO
Winner	FSimE	FPSO	FEPSO	WAPSO	PDPSO	NSGA-II	FPSO

Table 8.9: Comparison of algorithms with respect to ONVG metric (Continued).

Comparison of	FSimE vs FEPSO	FSimE vs WAPSO	FSimE vs PDPSO	FSimE vs NSGA-II	FPSO vs FEPSO	FPSO vs WAPSO	FPSO vs PDPSO
Winner	FEPSO	WAPSO	PDPSO	NSGA-II	FEPSO	FPSO	FPSO

Table 8.10: Comparison of algorithms with respect to ONVG metric (Continued).

Comparison of	FPSO vs NSGA-II	FEPSO vs WAPSO	FEPSO vs PDPSO	FEPSO vs NSGA-II	WAPSO vs PDPSO	WAPSO vs NSGA-II	PDPSO vs NSGA-II
Winner	FPSO	FEPSO	FEPSO	FEPSO	PDPSO	WAPSO	NSGA-II

Table 8.11: Average and standard deviation values of spacing metric for all the algorithms, for all the test cases.

Test Case	FSA	FSimE	FPSO	FEPSO	WAPSO	PDPSO	NSGA-II
h100N280a	1.144± 0.17	0.514± 0.064	0.424± 0.036	0.357± 0.16	0.362± 0.083	0.361± 0.024	1.00± 0.014
h100N360a	5.433± 0.36	1.385± 0.55	0.861± 0.051	0.416± 0.022	0.479± 0.075	0.959± 0.143	4.84± 0.42
h50N148a	1.831± 0.15	0.364± 0.067	0.552± 0.023	1.333± 0.877	0.511± 0.087	0.584± 0.016	0.67± 0.023
h50N212a	0.545± 0.035	0.518± 0.0284	0.449± 0.021	0.041± 0.01	0.382± 0.066	0.837± 0.06	0.45± 0.01
r100N403a	2.044± 0.326	0.392± 0.048	0.625± 0.08	0.511± 0.073	0.398± 0.053	0.606± 0.034	3.33± 0.18
r100N503a	1.59± 0.22	1.804± 0.148	0.507± 0.025	0.494± 0.031	1.049± 0.134	0.725± 0.061	0.643± 0.028
r50N228a	0.368± 0.145	0.356± 0.057	0.518± 0.0677	0.473± 0.083	0.303± 0.062	0.465± 0.034	0.464± 0.0245
r50N245a	0.691± 0.0363	0.414± 0.0323	0.453± 0.013	0.466± 0.042	0.965± 0.043	0.269± 0.021	0.79± 0.055
w100N391a	1.197± 0.242	1.415± 0.111	0.479± 0.028	0.0145± 0.001	0.277± 0.061	0.591± 0.075	1.12± 0.225
w100N476a	1.442± 0.286	0.706± 0.041	0.557± 0.033	0.492± 0.064	0.629± 0.023	0.811± 0.027	1.36± 0.028
w50N169a	0.481± 0.0342	0.821± 0.076	0.374± 0.044	0.388± 0.037	0.523± 0.061	0.687± 0.032	0.38± 0.043
w50N230a	5.835± 0.545	0.378± 0.053	0.119± 0.074	0.258± 0.012	0.264± 0.041	0.46± 0.041	1.95± 0.192

Table 8.12: Statistical results of FSA vs FSimE with respect to spacing metric.

Test case	FSA	FSimE	FSA vs FSimE (%diff.)	p-value	FSA wins	FSA losses	FSimE wins	FSimE losses
h100N280a	1.144	0.514	55.07	0.025	0	1	1	0
h100N360a	5.433	1.385	74.51	0.011	0	1	1	0
h50N148a	1.831	0.364	80.12	0.032	0	1	1	0
h50N212a	0.545	0.518	4.95	0.194	1	0	1	0
r100N403a	2.044	0.392	80.82	0.025	0	1	1	0
r100N503a	1.59	1.804	-13.46	0.038	1	0	0	1
r50N228a	0.368	0.356	3.26	0.607	1	0	1	0
r50N245a	0.691	0.414	40.09	0.024	0	1	1	0
w100N391a	1.197	1.415	-18.21	0.012	1	0	0	1
w100N476a	1.442	0.706	51.04	0.035	0	1	1	0
w50N169a	0.481	0.821	-70.69	0.031	1	0	0	1
w50N230a	5.835	0.378	93.52	0.038	0	1	1	0

Table 8.13: Comparison of algorithms with respect to spacing metric.

Comparison of	FSA vs FSimE	FSA vs FPSO	FSA vs FEPSO	FSA vs WAPSO	FSA vs PDPSO	FSA vs NSGA-II	FSimE vs FPSO
Winner	FSimE	FPSO	FEPSO	WAPSO	PDPSO	NSGA-II	FPSO

Table 8.14: Comparison of algorithms with respect to spacing metric (Continued).

Comparison of	FSimE vs FEPSO	FSimE vs WAPSO	FSimE vs PDPSO	FSimE vs NSGA-II	FPSO vs FEPSO	FPSO vs WAPSO	FPSO vs PDPSO
Winner	FEPSO	WAPSO	FSimE and PDPSO	FSimE	FEPSO	WAPSO	FPSO

Table 8.15: Comparison of algorithms with respect to spacing metric (Continued).

Comparison of	FPSO vs NSGA-II	FEPSO vs WAPSO	FEPSO vs PDPSO	FEPSO vs NSGA-II	WAPSO vs PDPSO	WAPSO vs NSGA-II	PDPSO vs NSGA-II
Winner	FPSO	FEPSO	FEPSO	FEPSO	WAPSO	WAPSO	PDPSO

tween FSA and FSimE with respect to the hypervolume metric is given in Table 8.17. For two test cases (h50N148a and h50N212a), there was no statistically significant difference in performance between FSA and FSimE. For the rest of the test cases, FSimE performed statistically significantly better than FSA. Aggregating the wins and losses of both the algorithms shows that FSimE performed better in the majority of the test cases. Hence, FSimE performed better than FSA with respect to the hypervolume metric. A similar process was followed to compare the rest of the algorithms. The final winners are tabulated in Tables 8.18, 8.19 and 8.20. Observe from these tables that FPSO showed comparable performance to PDPSO and FEPSO showed comparable performance to WAPSO. Out of all the algorithms, FEPSO showed the best performance in all the comparisons with respect to the hypervolume.

8.6 Conclusion

This chapter presented a comparison of the algorithms developed for solving the OSPFWS problem. A comparison of the single-solution algorithms, namely SA and

Table 8.16: Average and standard deviation values of hypervolume metric for all the algorithms, for all the test cases.

Test Case	FSA	FSimE	FPSO	FEPSO	WAPSO	PDPSO	NSGA-II
h100N280a	2.056± 0.054	1.451± 0.141	4.363± 0.178	6.68± 0.373	6.652± 0.193	3.924± 0.109	6.004± 0.184
h100N360a	1.815± 0.171	2.762± 0.136	2.311± 0.124	4.309± 0.207	4.628± 0.187	2.212± 0.226	2.517± 0.152
h50N148a	2.643± 0.613	2.734± 0.013	3.656± 0.212	3.322± 0.153	2.46± 0.078	3.592± 0.047	2.812± 0.145
h50N212a	2.573± 0.112	2.567± 0.131	2.113± 0.28	4.789± 0.208	6.196± 0.209	4.018± 0.363	3.629± 0.112
r100N403a	2.417± 0.312	3.01± 0.262	5.611± 0.345	5.899± 0.137	6.591± 0.225	4.865± 0.175	3.181± 0.114
r100N503a	1.872± 0.164	2.839± 0.118	4.838± 0.156	6.192± 0.138	6.179± 0.121	5.131± 0.139	3.03± 0.145
r50N228a	1.506± 0.172	2.083± 0.258	3.908± 0.102	4.565± 0.184	5.417± 0.137	4.185± 0.117	4.611± 0.254
r50N245a	2.689± 0.089	3.463± 0.176	5.092± 0.889	3.764± 0.112	3.26± 0.157	2.702± 0.185	3.101± 0.212
w100N391a	2.911± 0.126	3.143± 0.155	5.877± 0.841	5.864± 0.267	5.186± 0.345	6.401± 0.187	4.893± 0.211
w100N476a	2.539± 0.273	2.247± 0.123	4.507± 0.112	5.025± 0.226	5.523± 0.195	4.929± 0.058	1.79± 0.124
w50N169a	3.571± 0.192	1.798± 0.248	4.254± 0.274	2.665± 0.067	2.639± 0.131	4.202± 0.172	2.765± 0.085
w50N230a	2.2± 0.173	1.81± 0.123	5.223± 0.332	5.065± 0.295	5.687± 0.258	6.382± 0.036	4.857± 0.272

Table 8.17: Statistical results of FSA vs FSimE with respect to hypervolume metric.

Test case	FSA	FSimE	FSA vs FSimE (%diff.)	p-value	FSA wins	FSA losses	FSimE wins	FSimE losses
h100N280a	2.056	1.451	29.43	0.006	1	0	0	1
h100N360a	1.815	2.762	-52.18	0.035	0	1	1	0
h50N148a	2.643	2.734	-3.44	0.221	1	0	1	0
h50N212a	2.573	2.567	0.23	0.172	1	0	1	0
r100N403a	2.417	3.01	-24.53	0.033	0	1	1	0
r100N503a	1.872	2.839	-51.66	0.046	0	1	1	0
r50N228a	1.506	2.083	-38.31	0.032	0	1	1	0
r50N245a	2.689	3.463	-28.78	0.021	0	1	1	0
w100N391a	2.911	3.143	-7.97	0.017	0	1	1	0
w100N476a	2.539	2.247	11.5	0.012	1	0	0	1
w50N169a	3.571	1.798	49.65	0.014	1	0	0	1
w50N230a	2.2	1.81	17.73	0.027	1	0	0	1

Table 8.18: Comparison of algorithms with respect to hypervolume metric.

Comparison of	FSA vs FSimE	FSA vs FPSO	FSA vs FEPSO	FSA vs WAPSO	FSA vs PDPSO	FSA vs NSGA-II	FSimE vs FPSO
Winner	FSimE	FPSO	FEPSO	WAPSO	PDPSO	NSGA-II	FPSO

Table 8.19: Comparison of algorithms with respect to hypervolume metric (Continued).

Comparison of	FSimE vs FEPSO	FSimE vs WAPSO	FSimE vs PDPSO	FSimE vs NSGA-II	FPSO vs FEPSO	FPSO vs WAPSO	FPSO vs PDPSO
Winner	FEPSO	WAPSO	PDPSO	NSGA-II	FEPSO	WAPSO	FPSO and PDPSO

Table 8.20: Comparison of algorithms with respect to hypervolume metric (Continued).

Comparison of	FPSO vs NSGA-II	FEPSO vs WAPSO	FEPSO vs PDPSO	FEPSO vs NSGA-II	WAPSO vs PDPSO	WAPSO vs NSGA-II	PDPSO vs NSGA-II
Winner	FPSO	FEPSO and WAPSO	FEPSO	FEPSO	WAPSO	WAPSO	PDPSO

SimE, with respect to both FuzzyCF and SqalliCF revealed that SimE performed better than SA. A comparison of FPSO and FEPSO with respect to FuzzyCF revealed that FEPSO performed better than FPSO. Diversity trends across the algorithms are found to be comparable. This means that all the algorithms showed similar exploration and exploitation capabilities with few exceptions. With respect to the MOO performance measures, FEPSO performed better than the other algorithms.

Chapter 9

Conclusion

This thesis considered an optimization problem known as the *open shortest path first weight setting* (OSPFWS) problem. The OSPFWS problem is modelled as a multi-objective optimization problem. The first main objective of the thesis was to address the multi-objective nature of the problem. This was accomplished by using fuzzy logic to aggregate individual objectives into a multi-objective aggregation function.

The second main objective was the design and analysis of iterative heuristics to solve the OSPFWS problem. This objective was accomplished by developing a number of algorithms, including simulated annealing (SA), simulated evolution (SimE), and set-based particle swarm optimization (PSO) algorithms. A new hybrid PSO algorithm that incorporates characteristics of the SimE was also proposed and evaluated in the context of the OSPFWS problem.

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

9.1 Summary

The summary and main findings of this research work is as follows:

1. The first objective of this thesis was to model OSPFWS as a multi-objective optimization problem. An extensive literature review was thus conducted on the OSPFWS problem. The work of Fortz and Thorup [46] considered only optimizing Maximum Utilization (MU), while the work of Sqalli *et al.* [160] considered optimizing two objectives: MU and number of congested links (NOC). In a preliminary analysis, upon employing the cost functions of Fortz *et al.* and Sqalli *et al.*, a number of unused links (NUL) were available. This observation points to the fact that, to have a more stable traffic flow in the network, traffic from congested links can be shifted to these unused links. It was also not guaranteed that optimizing MU only would implicitly optimize NOC and NUL and vice versa. Thus, it was concluded that a multi-objective formulation of OSPFWS was not yet available. Besides the existing two objectives, namely MU and NOC, a third objective, NUL, was incorporated in the formulation of the OSPFWS problem. Fuzzy logic was employed to scalarize these multiple

objectives into a single-objective function, named FuzzyCF.

2. The next objective of this thesis was to develop a multi-objective fuzzy simulated annealing (FSA) algorithm. The FuzzyCF cost function was employed in the FSA algorithm. The control parameters of FSA, namely the cooling rate, α, and the length of the Markov chain, M, were tuned for all the test cases. It was found that $\alpha = 0.96$ and $M = 20$ produced better results.

3. The next objective of this thesis was to develop a multi-objective fuzzy simulated evolution (FSimE) algorithm. As for the FSA, the FuzzyCF cost function was also used in the FSimE algorithm. The control parameter, namely bias, B, was tuned for all the test cases. It was found that $B = -0.1$ produced better results. It was also determined that a dynamic bias could not produce better results.

4. The next objective of this thesis was to develop the fuzzy particle swarm optimization (FPSO) and a hybridized PSO, namely fuzzy evolutionary particle swarm optimization (FEPSO). The FuzzyCF was employed in both the algorithms. For the OSPFWS problem, the solution representation is defined as a set of weights on the network links. The control parameters namely swarm size, velocity clamping, acceleration coefficients, and the inertia weight were tuned for FPSO for all the test cases. After developing FPSO, a hybrid variant of FPSO using the SimE algorithm was developed to avoid replacement of high

quality (or fitness) weights on the network links. This was done by employing the evaluation and selection phases of the SimE algorithm in the FPSO algorithm. The best values obtained for the swarm size and velocity clamping for FPSO were then used for FEPSO. The acceleration coefficients and inertia weight parameters were tuned for FEPSO. This was done to ensure fair comparisons between the algorithms.

5. The next objective of this thesis was to develop the weighted-aggregation PSO (WAPSO) and Pareto-dominance PSO (PDPSO) algorithms. The aim was to compare the proposed fuzzy logic approach with another single-objective function approach, namely the weighted-aggregation method (used in WAPSO) and with a Pareto-dominance approach (used in PDPSO). The best values for swarm size and velocity clamping obtained for FPSO were used for WAPSO and PDPSO. The acceleration coefficients and inertia weight parameters were then tuned for both WAPSO and PDPSO.

6. The next objective was to compare the algorithms developed in this thesis to a well-established MOO algorithm, namely non-dominating sorting genetic algorithm (NSGA-II).

7. The next objective was to perform pairwise comparisons of all the algorithms. At first, FSA and FSimE were compared. It was found that FSimE performed

better than FSA. Then FPSO and FEPSO were compared. It was revealed that FEPSO performed better than FPSO. All the algorithms were then compared with respect to diversity. It was found that all the algorithms showed similar exploration and exploitation capabilities. Lastly, a comparison of all the algorithms with respect to the MOO performance measures ONVG, spacing, and hypervolume showed that FEPSO performed better than the other algorithms.

9.2 Future Research

Some directions for future research are summarized below.

Application of Other Techniques to OSPF Weight Setting Problem

While this thesis engineered and investigated a number of iterative algorithms to solve the OSPF weight setting problem, there are other techniques that can be applied to the underlying problem. Some of these techniques are biogeography based optimization, cuckoo search, the bat algorithm, the firefly algorithm, and the harmony search algorithm. These techniques have less parameters to tune compared to the PSO algorithm. Furthermore, application of these techniques to the domain of networking have been very limited, which motivates their application on the OSPFWS problem. Such

a study can further be enhanced by linking performance of algorithms to problem characteristics, through fitness landscape analysis.

Hyper-heuristic Approaches

A hyper-heuristic is a heuristic search method that involves several simpler heuristics (or meta-heuristics) to efficiently solve NP-hard problems. Hyper-heuristics [16, 136] provide a more comprehensive framework of solving a problem through an automated, intelligent process that allows selection of appropriate heuristics or meta-heuristics during different phases of the optimization process. Hyper-heuristic finds a solution to an optimization problem as well as finds the best heuristic for solving an optimization problem. A hyper-heuristic based approach can be developed encompassing the heuristics studied in this thesis (and including other as well) to solve the OSPFWS problem. This is because different algorithms have been shown to be best for different test cases. No one algorithm performed best for all the test cases.

Extension of the Problem into a Many-objective Optimization Problem

The current study considered three optimization objectives, namely, maximum utilization, number of congested links, and number of unused links. This motivated the use of fuzzy logic based multi-objective approach. However, more design objectives,

such as reliability, network delay, and link failure can be considered in the decision-making process. This would shift the paradigm of the problem from multi-objective to many-objective optimization. The many-objective optimization has its own challenges and highlights the need for new and more efficient algorithms that can handle more than three objectives. One possible direction could be the analysis of various fuzzy operators for many-objective optimization problems.

Other Aggregation Techniques

While the current study employed a fuzzy logic based approach for aggregation of multiple objectives, other approaches discussed in Chapter 2, such as goal programming, lexicographic ordering amongst others can also be exploited and compared with the results of the fuzzy logic based approach.

Bibliography

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

[2] J. Alvarez-Benitez, R. Everson, and J. Fieldsend. A MOPSO Algorithm Based Exclusively on Pareto Dominance Concepts. *3rd International Conference on Evolutionary Multi-criterion Optimization, Lecture Notes in Computer Science*, 3410:459–473, March 2005.

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

[4] 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*, 1447:601–610, March 1998.

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

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

[7] F. V. D. Bergh. *An Analysis of Particle Swarm Optimizers*. PhD Thesis, University of Pretoria, 2001.

[8] N. H. Bhagat. A New Hybrid Approach to OSPF Weight Setting Problem. *International Journal on Recent and Innovation Trends in Computing and Communication*, 1(5):443–450, 2013.

[9] M. Bhattacharya. Evolutionary Landscape and Management of Population Diversity. *Combinations of Intelligent Methods and Applications*, 46:1–18, January 2016.

[10] U. N. Black. *IP Routing Protocols*. Prentice Hall Series, 2000.

[11] A. Bley. On the Approximability of the Minimum Congestion Unsplittable Shortest Path Routing Problem. In *Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science*, volume 3509, pages 97–210, June 2005.

[12] A. Bley. Approximability of Unsplittable Shortest Path Routing Problems. *Networks*, 54(1):23–46, August 2009.

[13] A. L. Blumel, E. G. Hughes, and B. A. White. Fuzzy Auto Pilot Design Using a Multi-objective Evolutionary Algorithm. *IEEE Congress on Evolutionary Computation*, pages 80–83, 1999.

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

[15] L. Buriol, M. Resende, C. Rebeiro, and M. Thorup. A Memetic Algorithm for OSPF Routing. In *6th INFORMS Telecom*, pages 187–188, 2002.

[16] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. Hyper-heuristics: A Survey Of The State Of The Art. *Journal of the Operational Research Society*, 64:1695–1724, 2013.

[17] K. Calvert, M. Doar, and E. W. Zegura. Modeling Internet Topology. *IEEE Communications Magazine*, (35):160–163, 1997.

[18] A. Charnes and W. W. Cooper. Management Models and Industrial Applications of Linear Programming. *John Wiley, New York*, 1, 1961.

[19] W. N Chen, J. Zhang, H. Chung, W. L Zhong, W. G Wu, and Y. Shi. A

Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization Problems. *IEE Transactions on Evolutionary Computation*, 14(2):278–300, 2010.

[20] M. Chiampi, C. Ragusa, and M. Repetto. Fuzzy Approach for Multi-objective Optimization in Magnetics. *IEEE Transactions on Magnetics*, 32:1234–1237, 1996.

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

[22] C. A. C. Coello. A Comprehensive Survey of Evolutionary-Based Multi-objective Optimization Techniques. *Knowledge and Information Systems*, 1: 269–308, August 1999.

[23] E. Correa, A. Freitas, and C. Johnson. A New Discrete Particle Swarm Optimization Algorithm Applied to Attribute Selection in a Bioinformatics Data Set. *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 35–42, 2006.

[24] J. Crosby. *Computer Simulation in Genetics*. John Wiley & Sons, 1973.

[25] A. G. Cunha, P. Oliveira, and A. J. Covas. Genetic Algorithms in Multi-

objective Optimization Problems: an Application to Polymer Extrusion. *Genetic and Evolutionary Computation Conference*, pages 129–130, 1999.

[26] C. Darwin. Inception of Darwin's Evolutionary Theory. April 19 2008. http://en.wikipedia.org/wiki/Charles Darwin.

[27] 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.

[28] K. Deb. Multi-objective Optimization Using Evolutionary Algorithms. *John Wiley and Sons*, 2001.

[29] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[30] E. W. Dijkstra. A Node on Two Problems in Connection of Graphs. *Numerical Mathematics*, 1959.

[31] 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.

[32] J. Dombi. Basic Concepts for a Theory of Evaluation: The Aggregative Operator. *European Journal of Operational Research*, 10:282–293, 1982.

[33] D. Dubois and H. Prade. Operations in Fuzzy-valued Logic. *Information and Control*, 43:224–240, 1979.

[34] D. Dubois and H. Prade. A Class of Fuzzy Measures Based on Triangular Norms. *International Journal of General Systems*, 8:105–116, 1982.

[35] L. Duckstein. *Multi-objective Optimization in Structural Design: The Model Choice Problem*. North-Holland, Amsterdam, 1984.

[36] W. Elshamy, H. Emara, and A. Bahgat. Clubs-based Particle Swarm Optimization. *Proceedings of the IEEE Swarm Intelligence Symposium*, 2007.

[37] T. A. Ely, W. A. Crossley, and E. A. Williams. Satellite Constellation Design for Zonal Coverage Using Genetic Algorithms. *8th AAS/AIAA Space Flight Mechanics Meeting*, pages 124–129, 1998.

[38] A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons, 2005.

[39] A. P. Engelbrecht. Particle Swarm Optimization: Global Best or Local Best ? *BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, pages 124–135, September 2013.

[40] M. Ericsson, M. G. C. Resende, and P. M. Pardalos. A Genetic Algorithm for the Weight Setting Problem in OSPF Routing. *Journal of Combinatorial Optimisation conference*, 2002.

[41] L. R. Esau and K. C Williams. On Teleprocessing System Design. A Method for Approximating the Optimal Network. *IBM Systems Journal*, 5:142–147, 1966.

[42] C. M. M. Fonseca. Multi-objective Genetic Algorithms with Application to Control Engineering Problems. *Phd Thesis, University of Sheffield*, September 1995.

[43] Internet Engineering Task Force. OSPF Version 2. *Technical Report RFC 1583*, 1994.

[44] B. Fortz. Combinatorial Optimization and Telecommunications. http://www.poms.ucl.ac.be/staff/bf/en/COCom-5.pdf.

[45] B. Fortz, J. Rexford, and M. Thorup. Traffic Engineering with Tradional IP Routing Protocols. *IEEE Communications Magazine*, pages 118–124, 2002.

[46] B. Fortz and M. Thorup. Internet Traffic Engineering by Optimizing OSPF Weights. *IEEE Conference on Computer Communications*, pages 519–528, March 2000.

[47] B. Fortz and M. Thorup. Increasing Internet Capacity using Local Search. *Computational Optimization and Applications*, 29(1):13–48, October 2004.

[48] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS Weights in a Changing World. *IEEE Journal on Selected Area in Communications*, 20(4):756–767, September 2006.

[49] M. Frank. On the Simultaneous Associativity of $F(x, y)$ and $x + y - F(x, y)$. *Aequationes Mathematicae*, 19:194–226, 1979.

[50] A. Fraser. Simulation of Genetic Systems by Automatic Digital Computers. *Australian Journal of Biological Sciences*, 10:484–491, 1957.

[51] A. Fraser and D. Burnell. *Computer Models in Genetics*. McGraw-Hill, 1970.

[52] D. Frigioni, M. Loffreda, U. Nanni, and G. Pasqualone. Experimental Analysis of Dynamic Algorithms for the Single Source Shortest Paths Problem. *ACM Journal of Experimental Algorithms*, 1998.

[53] A. L. Garcia and I. Widjaja. *Communication Networks: Fundamental Concepts and Key Architectures*. McGraw Hill Higher Education.

[54] S. Gass and T. Saaty. The Computational Algorithm for the Parametric Objective Function. *Naval Research Logistics*, 2:39–45, March 1955.

[55] F. Gembicki. *Vector Optimization for Control with Performance and Parameter Sensitivity Indices*. Ph.D. Thesis, Case Western Reserve University, USA, 1974.

[56] A. A. Ghazala and A. E. Sayed. Open Shortest Path First Weight Setting (OSPFWS) Solving Algorithms Comparison and New Method for Updating Weights. *International Journal of Computer Science and Network Security*, 9 (5):191–197, May 2009.

[57] A. A. Ghazala, A. E. Sayed, and M. Mousa. A New Approach for Open Shortest Path Weight Setting (OSPFWS) Problem. *Convergence and Hybrid Information Technology*, pages 188–193, November 2008.

[58] A. A. Ghazala, A. E. Sayed, and M. Mousa. A Survey for Open Shortest Path First Weight Setting (OSPFWS) Problem. *The 2nd International Conference on Information Security and Assurance*, pages 24–26, April 2008.

[59] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[60] D. Goldberg and R. Lingle. Alleles, Loci, and the Traveling Salesman Problem. *1st International Conference on Genetic Algorithms*, pages 154–159, 1985.

[61] D. E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. *Addison-Wesley Publishing Company*, 1989.

[62] P. Gray, W. Hart, L. Painton, C. Phillips, M. Trahan, and J. Wagner. A

Survey of Global Optimization Methods. *In Sandia National Laboratories*, 1997. http://www.cs.sandia/gov/opt/survey.

[63] 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.

[64] H. Hamacher. Ueber Logische Verknupfungen Unschalfer Aussagen und deren Zugehoerige Bewertungs-funktione. *Progress in Cybernetics and Systems Research*, 3:276–288, 1978.

[65] W. Hines and D. Montgomery. *Probability and Statistics in Engineering and Management Science, 3rd Ed.* John Wiley & Sons, 1990.

[66] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[67] E. Horowitz and S. Sahni. Fundamentals of Computer Algorithms. *Computer Science Press*, 1984.

[68] S. M. H. Hosseini. A Multi-objective Genetic Algorithm (MOGA) for Hybrid Flow Shop Scheduling Problem with Assembly Operation. *Journal of Industrial and Systems Engineering*, 10:132–154, February 2017.

[69] X. Hu and R. Eberhart. Multi-objective Optimization Using Dynamic Neighborhood Particle Swarm Optimization. *IEEE Congress on Evolutionary Computation*, pages 1677–1681, May 2002.

[70] Y. Hu, Z. Bie, T. Ding, and Y. Lin. An NSGA-II based Multi-objective Optimization for Combined Gas and Electricity Network Expansion Planning. *Applied Energy*, 167:280–293, 2016.

[71] S. Huang, N. Tian, Y. Wang, and Z. Ji. Multi-objective Flexible Job-shop Scheduling Problem Using Modified Discrete Particle Swarm Optimization. *SpringerPlus*, 5, August 2016.

[72] T. Huang and A. Mohan. Significance of Neighborhood Topologies for the Reconstruction of Microwave Images using Particle Swarm Optimization. *Proceedings of the Asia-Pacific Microwave Conference*, 1, December 2005.

[73] T. Thomas II. *OSPF Network Design Solutions*. Cisco Press, 1998.

[74] Y. Ijiri. Management Goals and Accounting for Control. *North-Holland, Amsterdan*, 1965.

[75] 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 Power Delivery*, pages 1070–1078, 2002.

[76] C. Kahraman, D. Ruan, and I. Doan. Fuzzy Group Decision-making for Facility Location Selection. *Information Sciences*, 157:135–153, 2003.

[77] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the Tightrope: Responsive Yet Stable Traffic Engineering. In *ACM 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 253–264, 2005.

[78] 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.

[79] J. Kennedy and R. C. Eberhart. Particle Swarm Optimization. *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[80] J. Kennedy and R. Mendes. Population Structure and Particle Swarm Performance. *Proceedings of the IEEE Congress on Evolutionary Computation*, 2: 1671–1676, May 2002.

[81] J. Kennedy and R. Mendes. Neighborhood Topologies in Fully-Informed and Best-of-Neighborhood Particle Swarms. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 36(4):515–519, July 2006.

[82] S. Khajehpour and D. Grierson. Conceptual Design using Adaptive Computing. *Genetic and Evolutionary Computation Conference*, pages 62–67, 2001.

[83] S. Khan and S. Rehman. On the Use of Unified And-Or Fuzzy Aggregation Operator for Multi-criteria Decision Making in Wind Farm Design Process Using Wind Turbines in 500 kW to 750 kW Range. *IEEE World Congress on Computational Intelligence*, pages 1–6, 2012.

[84] S. A. Khan. *Design and Analysis of Evolutionary and Swarm Intelligence Techniques for Topology Design of Distributed Local Area Networks.* PhD Thesis, University of Pretoria, 2009.

[85] S. A. Khan and Z. A. Baig. On the Use of Unified And-Or Fuzzy Operator for Distributed Node Exhaustion Attack Decision-making in Wireless Sensor Networks. *IEEE 2010 International Conference on Fuzzy Systems*, pages 1–7, 2010.

[86] S. A. Khan and A. P. Engelbrecht. A New Fuzzy Operator and its Application to Topology Design of Distributed Local Area Networks. *Information Sciences*, 177(12):2692–2711, 2007.

[87] S. A. Khan and A. P. Engelbrecht. Application of Ordered Weighted Averaging and Unified And-Or Operators to Multi-objective Particle Swarm Optimization

Algorithm. *IEEE 6th International Conference on Fuzzy Systems and Knowledge Discovery*, pages 176–180, 2009.

[88] S. A. Khan and A. P. Engelbrecht. Fuzzy Hybrid Simulated Annealing Algorithms for Topology Design of Switched Local Area Networks. *Soft Computing*, 3(1):45–61, 2009.

[89] S. A. Khan and A. P. Engelbrecht. A Fuzzy Particle Swarm Optimization Algorithm for Computer Communication Network Topology Design. *Applied Intelligence*, 36:161–177, 2012.

[90] H. Kim, Y. Hayashi, and K. Nara. The Performance of Hybridized Algorithm of GA, SA and TS for Thermal Unit Maintenance Scheduling. *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 114–119, November 1995.

[91] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):498–516, May 1983.

[92] 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.

[93] 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.

[94] 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*, pages 1303–1305, October 1991.

[95] R. M. Kling and P. Banerjee. ESP: A New Standard Cell Placement Package Using Simulated Evolution. *Proceedings of 24th Design Automation Conference*, pages 60–66, 1987.

[96] A. Konak, D. W. Coit, and A. E. Smith. Multi-objective Optimization Using Genetic Algorithms: A Tutorial. *Reliability Engineering and System Safety*, 91 (9):992–1007, September 2006.

[97] J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *American Mathematical Society*, (7(1)):48–50, 1956.

[98] A. Kurapati and S. Azarm. Immune Network Simulation with Multi-objective Genetic Algorithms for Multidisciplinary Design Optimization. *Engineering Optimization*, pages 245–260, 2000.

[99] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Prentice Hall Series, 2002.

[100] P. Laarhoven and E. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer Academic, Norwell, Massachusetts, 1987.

[101] J. Langeveld and A. P. Engelbrecht. A Generic Set-Based Particle Swarm Optimization Algorithm. *International Conference on Swarm Intelligence*, June 2011.

[102] L. H. Lee, E. P. Chew, and Y. Chen. Multi-objective Simulation-based Evolutionary Algorithm for an Aircraft Spare Parts Allocation Problem. *European Journal of Operational Research*, 189:476–491, 2008.

[103] S. Lee, T. Po-Kai, and A. Chen. Link Weight Assignment and Loop-free Routing Table Update for Link State Routing Protocols in Energy-aware Internet. 28: 437–445, 2012.

[104] H. Li and D. Landa-Silva. Evolutionary Multi-objective Simulated Annealing with Adaptive and Competitive Search Direction. *IEEE Congress on Evolutionary Computation*, pages 3310–3317, 2008.

[105] H. Li and V. Yen. *Fuzzy Sets and Fuzzy Decision-Making*. CRC Press, USA, 1995.

[106] Z. Lian. A United Search Particle Swarm Optimization Algorithm for Multi-

objective Scheduling Problem. *Applied Mathematical Modelling*, 34:3518–3526, November 2010.

[107] L. Lin and M. Gen. Priority-Based Genetic Algorithm for Shortest Path Routing Problem in OSPF. *Intelligent and Evolutionary Systems, Studies in Computational Intelligence*, 187:91–104, 2009.

[108] Y. Marinakis and M. Marinaki. Particle Swarm Optimization with Expanding Neighborhood Topology for the Permutation Flowshop Scheduling Problem. *Soft Computing*, 2013.

[109] E. Masehian and D. Sedighizadeh. A Multi-objective PSO-based Algorithm for Robot Path Planning. *Industrial Technology, IEEE International Conference*, March 2010.

[110] I. MatSuba. Optimal Simulated Annealing Method and its Application to Combinatorial Problems. *Proceedings of the International Joint Conference on Neural Networks*, pages 541–546, 1989.

[111] D. Medhi and K. Ramasamy. *Network Routing: Algorithms, Protocols, and Architectures*. 2010.

[112] J. M. Mendel. Fuzzy Logic Systems for Engineering: A Tutorial. *Proceedings of the IEEE*, 83(3):345–377, March 1995.

[113] K. Miettinen. Some Methods for Nonlinear Multi-objective Optimization. *IEEE/ACM 1st International Conference on Evolutionary Multi-Criterion Optimization, Lecture notes in Computer Science*, 1993:1–20, July 2001.

[114] M. Mizumoto. Fuzzy Sets and Their Operations. *Information and Control*, 48: 30–48, 1981.

[115] M. Mizumoto. Comparison of Various Fuzzy Reasoning Methods. In *2nd International Fuzzy Systems Association Congress*, volume 8, pages 253–283, 1982.

[116] M. Mohiuddin, S. A. Khan, and A. P. Engelbrecht. Simulated Evolution and Simulated Annealing Algorithms for Solving Multi-objective Open Shortest Path First Weight Setting Problem. *Applied Intelligence, Springer*, 40(3):1–20, 2014.

[117] S. Mostaghim and J. Teich. Strategies for Finding Good Local Guides in Multi-objective Particle Swarm Optimization (MOPSO). *IEEE Swarm Intelligence Symposium*, pages 26–33, 2003.

[118] J. T. Moy. OSPF version 2, April 1998.

[119] J. T. Moy. *OSPF: Anatomy of an Internet Routing Protocol*. Addison-Wesley, 1999.

[120] T. Murata, H. Ishibuchi, and H. Tanaka. Multi-objective Genetic Algorithm

and its Applications to Flowshop Scheduling. *Computers and Industrial Engineering*, 30:957–968, September 1996.

[121] S. Nahar, S. Sahni, and E. Shragowitz. Experiments with Simulated Annealing. *22nd Design Automation Conference*, pages 748–752, 1985.

[122] J. F. Nash. The Bargaining Problem. *Econometrica*, 18:155–162, 1950.

[123] C. Neethling and A. Engelbrecht. Determining RNA Secondary Structure Using Set-based Particle Swarm Optimization. *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1670–1677, 2006.

[124] G. Nemhauser, A. Rinnooy Kan, and M. Todd (Eds.). Optimization. *North-Holland*, 1989.

[125] A. Nucci, S. Bhattacharyya, N. Taft, and C. Diot. IGP Link Weight Assignment for Operational Tier-1 Backbones. *IEEE/ACM Transactions on Networking*, 15 (4):789–802, 2007.

[126] 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*, 1986.

[127] G. Oltean, C. Miron, and E. Moccan. Multi-objective Optimization for Analog Circuits Design Based on Fuzzy Logic. *9th International Conference on Electronics, Circuits and Systems*, pages 777–780, September 2002.

[128] M. Omran. Particle Swarm Optimization Methods for Pattern Recognition and Image Processing. *PhD Thesis, University of Pretoria*, 2005.

[129] I. H. Osman. Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem. *Annals of Operation Research*, (41):421–451, 1993.

[130] S. Palaniappan, S. Zein-Sabatto, and A. Sekmen. Dynamic Multi-objective Optimization of War Resource Allocation using Adaptive Genetic Algorithms. *IEEE Southeast Conference*, pages 160–165, March 2001.

[131] C. Papadimiriou and K. Steiglitz. Combinatorial Optimization: Algorithms and Complexity. *Dover Publications*, 1998.

[132] A. Parmar, S. Ahmed, and J. Sokol. *An Integer Programming Approach to the OSPF Weight Setting Problem.* NSF Technical Report no. DMI-0457066, 2006.

[133] K. Parsopoulos and M. Vrahatis. Particle Swarm Optimization Method in Multi-objective Problems. *ACM Symposium on Applied Computing*, pages 603–607, March 2002.

[134] E. Peer, F. van den Bergh, and A. Engelbrecht. Using Neighborhoods with the Guaranteed Convergence PSO. *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 235–242, 2003.

[135] C. Perttunen. Non-parametric Cooling Schedules in Simulated Annealing Using the Normal Score Transformations. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 609–612, 1991.

[136] S. Petrovic and R. Qu. Case-based Reasoning as a Heuristic Selector in a Hyper-Heuristic for Course Timetaling. *Sixth International Conference on Knowledge-based Intelligent Information and Engineering Systems*, September 2002.

[137] M. Pioro, A. Szentsi, J. Harmatos, A. Juttner, P. Gajownicczek, and S. Kozdrowski. On Open Shortest Path First Related Network Optimization Problems. *Performance Evaluation*, 48(4):201–223, 2002.

[138] R. C. Prim. Shortest Connection Networks and Some Generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[139] M. Rabbani, H. Farrokhi-Asl, and B. Asgarian. Solving a Bi-objective Location Routing Problem by a NSGA-II Combined with Clustering Approach: Application in Waste Collection Problem. *Journal of Industrial Engineering International*, 13(1):13–27, Mar 2017. ISSN 2251-712X. URL `https://doi.org/10.1007/s40092-016-0172-8`.

[140] G. Ramalingam and T. Reps. An Incremental Algorithm for a Generalization of the Shortest Path Problem. *Journal of Algorithms*, pages 267–305, 1996.

[141] R. Reis, M. Ritt, L. Buriol, and M. Resende. A Memetic Algorithm for the Weight Setting Problem in DEFT. In *COMCEV 2007*, pages 1–6, 2007.

[142] G. Retvari, J. Biro, and T. Cinkler. On Improving the Accuracy of OSPF Traffic Engineering. In *NETWORKING 2006, Lecture Notes in Computer Science, Vol. 3976*, pages 51–62, 2006.

[143] G. Retvari and T. Cinkler. Practical OSPF Traffic Engineering. *IEEE Commununications Letters*, 8:689–691, 2004.

[144] J. H. Reynolds and E. D. Ford. Multi-criteria Assessment of Ecological Process Models. *Ecology*, pages 538–553, 1999.

[145] RFC1058. Routing Information Protocol.

[146] A. Riedl. Optimized Routing Adaptation in IP Networks Utilizing OSPF and MPLS. In *IEEE International Conference on Communications*, pages 1754–1758, 2003.

[147] M. Rodrigues and K. G. Ramakrishnan. Optimal Routing in Data Networks. *Bell Labs Technical Journal*, 6(1):117–138, 2002.

[148] S. Sait, H. Youssef, and A. Hussain. Fuzzy Simulated Evolution Algorithm for Multi-objective Optimization of VLSI Placement. In *IEEE Congress on Evolutionary Computation, Washington*, pages 91–97, 1999.

[149] S. M. Sait, M. H. Sqalli, and M. A. Mohiuddin. Engineering Evolutionary Algorithm to Solve Multi-objective OSPF Weight Setting Problem. *Australian Conference on Artificial Intelligence*, 4304:950–955, 2006.

[150] S. M. Sait and H. Youssef. *Iterative Computer Algorithms and their Application to Engineering*. IEEE Computer Society Press, December 1999.

[151] D. Savic. Single-objective vs. Multi-objective Optimization for Integrated Decision Support. *Technical Report, University of Exeter*, 2001.

[152] J. R. Schott. Fault Tolerant Design Using Single and Multi-criteria Genetic Algorithm Optimization. *Master's Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, USA*, 1995.

[153] B. Schweizer and A. Sklar. Associative Functions and Abstract Semigroups. *Publicationes Mathematicae Debrecen*, 10:69–81, 1963.

[154] Y. Shi and R. C Eberhart. Parameter Selection in Particle Swarm Optimization. *In: Porto VW, Saravanan N, Waagen D, Eiben A (eds) Evolutionary programming VII. Springer, Berlin*.

[155] 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.

[156] G. Shrimali, A. Akella, and A. Mutapcic. Cooperative Interdomain Traffic Engineering Using Nash Bargaining and Decomposition. *IEEE/ACM Transactions on Networking*, 18(2):341–352, 2010.

[157] K. I. Smith. A Study of Simulated Annealing Techniques for Multi-objective Optimization. *Phd Thesis, University of Exeter*, October 2006.

[158] R. Soland. Multi-criteria Optimization: A General Characterization of Efficient Solutions. *Decision Sciences*, 10:26–38, 1979.

[159] M. H. Sqalli, S. M. Sait, and S. Asadullah. Minimizing the Number of Congested Links in OSPF Routing. *ATNAC*, December 2008.

[160] M. H. Sqalli, S. M. Sait, and M. A. Mohiuddin. An Enhanced Estimator to Multi-objective OSPF Weight Setting Problem. *Network Operations and Management Symposium, NOMS*, April 2006.

[161] D. Srinivasan and T. H Seow. Particle Swarm Inspired Evolutionary Algorithm (PS-EA) for Multi-objective Optimization Problems. *IEEE Congress on Evolutionary Computation*, pages 2292–2297, 2003.

[162] S. Srivastava, G. Agarwal, D. Medhi, and M. Pioro. Determining Feasible

Link Weight Systems Under Various Objectives for OSPF Networks. *IEEE eTransactions on Network and Service Management*, 2(1), 2005.

[163] A. Suppapitnarm, K. A. Seffen, G. T. Parks, and P. J. Clarkson. A Simulated Annealing Algorithm For Multi-objective Optimization. *Engineering Optimization*, 33:59–85, April 2007.

[164] A. Syberfeldt, A. Ng, R. I. John, and P. Moore. Multi-objective Evolutionary Simulation-optimization of a Real-world Manufacturing Problem. *Robotics and Computer-Integrated Manufacturing*, pages 926–931, 2009.

[165] G. Sywerda. Uniform Crossover in Genetic Algorithms. *3rd International Conference on Genetic Algorithms*, pages 2–9, 1989.

[166] K. S. Tang, T. M. Chan, R. J. Yin, and K. F Man. *Multi-objective Optimization Methodology: A Jumping Gene Approach*. CRC Press Taylor and Francis Group, 2012.

[167] H. S. Urade and R. Patel. Dynamic Particle Swarm Optimization to Solve Multi-objective Optimization Problem. *2nd International Conference on Communication*, 6:283–290, 2012.

[168] M. Vazquez and L. D. Whitley. A Hybrid Genetic Algorithm for the Quadratic

Assignment Problem. *Genetic and Evolutionary Computation Conference*, pages 169–178, 2000.

[169] C. Veenhuis. A Set-Based Particle Swarm Optimization Method. *Proceedings of the Problem Solving from Nature Conference*, 5199:971–980, 2008.

[170] I. Venanzy and A. L. Materazzi. Multi-objective Optimization of Wind-excited Structures. *Engineering Structures*, 29:983–990, 2007.

[171] 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.

[172] H. Weiss. Genetic Algorithm and Optimum Robot Design. *Technical Report, Institute of Robotics and Mechatronics*, 2003.

[173] 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.

[174] Z. Wu, Z. Ni, L. Gu, and X. Liu. A Revised Discrete Particle Swarm Optimization for Cloud Workflow Scheduling. *Proceedings of the International Conference on Computational Intelligence and Security*, pages 184–188, 2010.

[175] X. X. Xu, X. M. Hu, W. N. Chen, and Y. Li. Set-based Particle Swarm Optimization for Mapping and Scheduling Tasks on Heterogeneous Embedded Sys-

tems. *Eighth International Conference on Advanced Computational Intelligence*, pages 318–325, February 2016.

[176] H. Youssef, S. Sait, and S. Khan. Fuzzy Evolutionary Hybrid Metaheuristic for Network Topology Design. *IEEE/ACM 1st International Conference on Evolutionary Multi-Criterion Optimization, Lecture Notes in Computer Science*, 1993:400–415, 2001.

[177] H. Youssef, S. Sait, and S. Khan. Topology Design of Switched Enterprise Networks using a Fuzzy Simulated Evolution Algorithm. *Engineering Applications of Artificial Intelligence*, 15:327–340, 2002.

[178] Y. Yu. Multi-objective Decision Theory for Computational Optimization in Radiation Therapy. *Medical Physics*, pages 1445–1454, 1997.

[179] L. A. Zadeh. Optimality and Non-Scalar-Valued Performance Criteria. *IEEE Transactions on Automatic Control*, 8:59–60, January 1963.

[180] L. A. Zadeh. Fuzzy Sets. *Information Control*, 8:338–353, 1965.

[181] L. A. Zadeh. The Concept of a Linguistic Variable and its Application to Approximate Reasoning. *Information Sciences*, 8:199–249, 1975.

[182] M. Zagozdzon, M. Dzida, and M. Pioro. Traffic Flow Optimization in Networks

with Combined OSPF/MPLS Routing. In *IEEE 15th International Conference on Advanced Computing and Communications*, pages 131–137, 2007.

[183] E. W. Zegura. GT-ITM: Georgia Tech Internetwork Topology Models (software). 1996. http://www.cc.gatech.edu/faq/Ellen.Zegura/gt-itm/gt-itm.tar.gz.

[184] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How To Model An Internetwork. *15th IEE Conference on Computer Communications*, pages 594–602, 1996.

[185] H.J Zimmerman. *Fuzzy Set Theory and its Applications*. Kluwer Academic Publishers, third edition, 1996.

Appendix

Symbols

This appendix defines the commonly used symbols in this thesis. Symbols are listed under each chapter according to their first appearance.

Chapter 2

f	The fitness function
$\mathbf{x}^*$	The global optimum solution vector
h_m	The equality constraints
g_m	The inequality constraints
n_g	The number of inequality constraints
n_h	The number of equality constraints
n_o	The number of objectives
n_x	The number of decision variables
x_{min}	The minimum boundary constraint
x_{max}	The maximum boundary constraint
λ_k	The weight value in weighted sum method for the objective k
ε_k	The upper bound for the objective k
$Q^{'}$	The set of solutions

Q	The non-dominated set of solutions
μ	The fuzzy logic membership function
C	Crisp set
U	Fuzzy set
Ω	A linguistic variable
$T(\Omega)$	Collection of values for a linguistic variable
R	Syntactic rule to characterize a linguistic variable
$\aleph$	Semantic rule to characterize a linguistic variable
S_0	Initial solution
T_0	Initial temperature in SA algorithm
α	Cooling rate parameter
M	Markov chain parameter
β	Constant value by which Markov chain parameter is increased
T_{max}	Maximum amount of time
r	A random number sampled from a uniform distribution in the range (0,1)
g_i	Goodness of individual i in SimE algorithm
O_i	Optimal cost of individual i in SimE algorithm
C_i	Actual cost of individual i in SimE algorithm
B	Selection bias parameter of SimE algorithm
P	Population set

p_m	Mutation rate
$\mathbf{x_i}$	Current position of particle i in PSO
$\mathbf{v_i}$	Current velocity of particle i in PSO
$\mathbf{y_i}$	The personal best position of particle i in PSO
$\hat{\mathbf{y}}_\mathbf{i}$	The global best position of particle i in PSO
w	Inertia weight
c_1	Cognitive acceleration constant
c_2	Social acceleration constant
$n_{\mathcal{N}}$	Number of immediate neighbors in ring topology
n_s	Swarm size
n_d	Number of dimensions in PSO
V_{max}	Velocity clamping
S	Spacing, MOO performance measure
f_m^i	The m^{th} objective function value of solution i
h_i	The volume of the hypercube constructed with non-dominating solution i

Chapter 3

d_{ij}	The cost between nodes i and j of the network
D_{ij}	The computed minimum cost from node i to node j
N	Set of nodes in the graph
L	Set of permanent list of nodes used in Dijkstra algorithm

L'	Set of temporary list of nodes used in Dijkstra algorithm
Φ	The cost function
l_a	The total traffic load on arc a
Φ_a	The cost associated with arc a
A	Set of arcs
c_a	Capacity of arc a in the graph
$f_a^{(s,t)}$	Traffic flow from node s to t over arc a

Chapter 4

G	Graph
A^t	Set of arcs representing shortest paths from all sources to destination node t
a	A single element in set A. It can also be represented as (i, j)
D	Demand matrix
ω_{max}	The maximum value of a weight on the network link
$\bar{N}$	Set of destination nodes
d_u^t	Shortest distance from node u to the destination node t
ω_{ij}	Weight on arc (i, j); if $a = (i, j)$, then it can also be represented as ω_a
δ_u^t	Outdegree of node u when destination node is t
G^t	Sub graph when t is the destination node
$l_{(v,h)}^t$	The traffic load on the link (v, h) when t is the destination node
MU_{min}	The minimum value of maximum utilization

MU_{max}	The maximum value of maximum utilization
μ_{MU}	The membership value for maximum utilization
NOC_{min}	The minimum value of number of congested links
NOC_{max}	The maximum value of number of congested links
μ_{NOC}	The membership value for number of congested links
NUL_{min}	The minimum value of number of unused links
NUL_{max}	The maximum value of number of unused links
μ_{NUL}	The membership value for number of unused links

Chapter 5

N_{T_0}	The number of moves accepted at temperature T_0 in SA algorithm
T_{T_0}	The total number of moves attempted at temperature T_0 in SA algorithm

Chapter 6

g_i	The goodness of link i
u_i	The utilization of link i

Chapter 7

n_D	The number of non-dominated solutions
S_D	The set of dominated solutions
c_d	The crowding distance

Acronyms

Below is a list of acronyms used in this thesis in alphabetic order.

CO	Combinatorial optimization
EGP	Exterior Gateway Routing Protocol
FEPSO	Fuzzy Evolutionary Particle Swarm Optimization
FPSO	Fuzzy Particle Swarm Optimization
FSA	Fuzzy Simulated Annealing
FSimE	Fuzzy Simulated Evolution
GA	Genetic Algorithm
HV	Hypervolume, MOO performance measure
IP	Internet Protocol
IGP	Interior Gateway Routing Protocol
MOO	Multi-objective Optimization
MU	Maximum Utilization
NOC	Number of Congested Links/Arcs
NSGA-II	Non-dominating Sorting Genetic Algorithm-II
NUL	Number of Unused Links/Arcs
ONVG	Overall non-dominated vector generation
OSPF	Open Shortest Path First Algorithm

OSPFWS	Open Shortest Path First Algorithm Weight Setting
PSO	Particle Swarm Optimization
RIP	Routing Information Protocol
SA	Simulated Annealing
SimE	Simulated Evolution
SOO	Single-objective Optimization

Derived Journal Publications

1. Mohammed A. Mohiuddin, Salman A. Khan and Andries P. Engelbrecht, "Simulated Evolution and Simulated Annealing Algorithms for Solving Multi-objective Open Shortest Path First Weight Setting Problem", Applied Intelligence, Springer, Vol. 41, September 2014, pp. 348-365.

2. Mohammed A. Mohiuddin, Salman A. Khan and Andries P. Engelbrecht, "Fuzzy Particle Swarm Optimization Algorithms for the Open Shortest Path First Weight Setting Problem", Applied Intelligence, Springer, Vol. 45, October 2016, pp. 598-621.