USING EVOLUTIONARY APPROACH TO OPTIMIZE AND MODEL MULTI-SCENARIO, MULTI-OBJECTIVE FAULT-TOLERANT PROBLEMS

By

Ling Zhu

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science - Doctor of Philosophy

2017

ABSTRACT

USING EVOLUTIONARY APPROACH TO OPTIMIZE AND MODEL MULTI-SCENARIO, MULTI-OBJECTIVE FAULT-TOLERANT PROBLEMS

By

Ling Zhu

Fault-tolerant design involves different scenarios, such as scenarios with no fault in the system, with faults occurring randomly, with different operation conditions, and with different loading conditions. For each scenario, there can be multiple requirements (objectives). To assess the performance of a design (solution), it needs to be evaluated over a number of different scenarios containing various requirements in each scenario. We consider this problem as a multi-scenario, multi-objective (MSMO) problem.

Despite its practical importance and prevalence in engineering application, there are not many studies which systematically solve the MSMO problem. In this dissertation, we focus on optimizing and modeling MSMO problems, and propose various approaches to solve different types of MSMO optimization problems, especially multi-objective fault-tolerant problems.

We classify MSMO optimization problem into two categories: scenario-dependent and scenario-independent. For the scenario-dependent MSMO problem, we review existing methodologies and suggest two evolutionary-based methods for handling multiple scenarios and objectives: aggregated method and integrated method. The effectiveness of both methods are demonstrated on several case studies including numerical problems and engineering design problems. The engineering problems include cantilever-type welded beam design, truss bridge design, four-bar truss design. The experimental results show that both methods can find a set of widely distributed solutions that are compromised among the respective objective values under all scenarios. We also model fault-tolerant programs using the aggregated method. We synthesize three fault-tolerant dis-

tributed programs: Byzantine agreement program, token ring circulation program and consensus program with failure detector S. The results show that evolutionary-base MSMO approach, as a generic method, can effectively model fault-tolerant programs.

For the scenario-independent MSMO problem, we apply evolutionary multi-objective approach. As a case study, we optimize a probabilistic self-stabilizing program, a special type of fault-tolerant program, and obtain several interesting counter-intuitive observations under different scenarios.

ACKNOWLEDGMENTS

First of all, I am very grateful to my doctoral adviser and mentor Professor Sandeep Kulkarni for his advice, encouragement, support and mentorship. He taught me about research and science, and always encouraged and supported me throughout the course of study, even when I was going through difficult times in my life. Also, His expertise, patience, kindness, and creative thinking have inspired me a lot. It is impossible for me to overstate my gratitude to him.

I was very fortunate to have another doctoral adviser. I would like to take this opportunity to thank Professor Kalyanmoy Deb. He introduced me to the research area of evolutionary optimization. I greatly appreciate and value his expertise and knowledge which had a significant impact on my research. Also, he always inspires us to go beyond and break new grounds. It is my honor to work with him.

I have been blessed with a number of excellent teachers, classmates and friends who taught me, helped me, and supported me. I am always grateful for their kindness and help.

Finally, I would like to thank my family who supported me with their unconditional love. Special thanks to my husband Lin Li for his unwavering support and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Computing systems are inherently critical in our lives. During their execution, they are subject to a variety of expected and unexpected faults including 1) message faults such as message loss, duplication; 2) process faults such as crash faults, byzantine (malicious) faults, transient faults, and so on. Fault-tolerance refers to the ability of a system to provide acceptable specification even if faults occur. Since cost of failure can be unacceptably high [2], correctness of fault-tolerant systems is vital. Automated approach enhances the ability to provide assurance about computing systems and, hence, are highly desirable.

Most of the previous work on modeling fault-tolerant programs focus on designing the program with the use of (manually designed) heuristics. These heuristics attempt to reduce the design complexity of fault-tolerant systems by performing efficient search in the given state space. However, heuristics have certain limitations in automatically designing some problems [3]. Since these heuristics are pre-defined, identifying new heuristics in a different setting is difficult. Moreover, if these heuristics fail, then designing the corresponding fault-tolerant program is impossible. Besides computing systems, fault-tolerance plays a significant role in other engineering systems, such as hardware design, automotive engineering and structural optimization, etc. Hence, a generic approach that can be applied in different areas is highly desired.

In this dissertation, we use evolutionary algorithm (EA), a generic optimization methodology, to model and optimize the fault-tolerance property. EA is a computational method inspired by

natural evolution to solve optimization problems. It is routinely used to generate useful solutions and is broadly applied in many different fields, such as bioinformatics, computational science, engineering, economics, and manufacturing, etc. We utilize two formats of EA in this work: Genetic programming (GP) [4, 5] and Genetic Algorithm (GA). They both are evolutionary algorithm-based methodologies, where the former is broadly used as a modeling method, and the latter is applied as an optimization method.

1.1 Multi-scenario, Multi-objective Fault-tolerant Problem

Generic EA begins with a population of randomly generated solutions. Then, it evaluates generated solutions and assigns objective values to each solution. For a fault-tolerant solution, it must be evaluated against a number of scenarios: a scenario in the absence of faults, another scenario in the presence of one or multiple faults, and other scenarios. For example, in fault-tolerant mutual exclusion program synthesis, program must be verified under different scenarios such as when no faults occurs, when a single process failure occurs, and when multiple failures occur. Similarly, in a structural optimization problem, a solution must be checked under a number of loading conditions arising from various considerations, such as severe wind conditions providing lateral loads and vertical loads, heavy snow conditions, and failure of one truss member etc. Furthermore, in each scenario, there usually have multiple criteria (objectives) that need to be considered. In these problems, a solution is considered acceptable only if it performs in a satisfactory manner to not one but all specified scenarios with considering all criteria in each scenario. We denote such problems as multi-scenario, multi-objective (MSMO) problems.

The optimization and design under various scenarios is very important and challenging. In most design optimization studies, a single scenario is considered and an optimal design is obtained for

that scenario. However, the optimal solutions obtained in single scenario can be overestimated or underestimated in terms of optimizing the respective objectives, and thus, may not be appropriate for other scenarios. From a practical standpoint, such a design does not make a good compromise of all scenarios. In this dissertation, we address this challenging issue by suggesting evolutionary methodologies based on handling of multiple objectives in multiple scenarios to design and optimize fault-tolerance property.

1.2 Scenarios

In general, scenarios are the different descriptions of a problem. The scenario can be defined differently depending on the problem. Previous studies [6, 7, 8] describe the scenarios in different fields. In structural analysis field, loading cases are commonly considered as scenario. In engineering design involving different design disciplines, each design discipline is considered as scenario. In automotive design, the scenario can be variations of a product, vehicle testing environments, etc. This dissertation focuses on scenarios in software synthesis and structural optimization. In software synthesis (or program synthesis), scenarios can be program running environments or program variations. The programs often run in an unpredictable environment where faults can occur. In order to synthesize (or model) fault-tolerant program, scenarios with different types of faults and the ideal scenario without any faults need to be considered. In structural optimization, the fault-tolerant design considers different loading conditions as well as some faulty conditions.

1.3 Two Types of MSMO

According to attributes of MSMO problems, they can be classified into two categories: scenario-dependent and scenario-independent. In scenario- dependent problems, one solution compromises in all different scenarios. All scenarios are taken into consideration during one optimization. This is most common approach to solve MSMO problems in the literature. In contrast, scenario-independent problems have different solutions for each scenario. These problems optimize each scenario independently, and require different optimal solutions for each scenario. Although the scenarios are related to each other, it is not necessary to consider all of them in one optimization. However, the optimal solutions from each scenario provide useful problem knowledge. Solutions are analyzed to decipher useful relationships among scenarios so as to provide a better understanding of the problem to a designer.

1.4 Contributions

The main contributions of this dissertation are summarized as follows.

1. Methodologies

    (a) We transform fault-tolerant problem into MSMO problem, and suggest multiple scenarios handling which maintains the multi-objective problem solving principle and demonstrates its working on a number of numerical problems, distributed programs and engineering design problems.

    (b) We propose two different types of methods for handling multiple scenarios for scenario-dependent problem: aggregated method and integrated method, both of which are based on posterior decision making approach which provides the solution set compromising

from all scenarios for decision maker.

The aggregated method combines all scenarios for each objective function and builds an aggregate function for objective, which are then handled by an evolutionary multi-objective optimization (EMO) procedure. The integrated approach considers objectives in all scenarios in an integrated manner in an EMO, and attempts to find a set of compromise solutions.

(c) We further provide a comparative study of proposed aggregated method with a classic method.

2. Applications

(a) **Modeling Fault-Tolerant Programs**

i. We demonstrate the feasibility of using multi-scenario multi-objetive GP to synthesize fault-tolerant programs.

ii. We use two types of techniques to analyze programs that are evolved by GP: simulation based and model-checking based approaches. We also compare the effectiveness of both approaches.

iii. We demonstrate that model-checking based analysis can improve the performance of GP in terms of the number of generations required to obtain the desired fault-tolerant program as well as the time spent in each generation.

iv. We find that GP can effectively synthesize fault-tolerant programs such as Byzantine agreement program, token ring program and consensus program. This is the first successful approach to automated design of consensus protocol with failure detector S.

(b) **Optimization of Fault-Tolerant Programs**

i. We apply evolutionary approach to optimize two properties of probabilistic stabilizing programs to identify the trade-offs between two properties.

ii. We handle the different variations (consider each variation as a scenario) of probabilistic stabilizing programs and analyze the evolved solutions.

iii. We obtain observations from different scenarios and gain better understanding of probabilistic stabilizing programs.

1.5 Outline

The rest of the dissertation is organized as follows. Chapter 2 provides background on distributed programs, fault-tolerant programs, and evolutionary technique. In Chapter 3, we introduce scenario-dependent problem. In the following two chapters, we discuss two MSMO approaches to optimize scenario-dependent problem, and demonstrates each approach using various applications. Chapter 6 depicts scenario-independent problem and the related application, and Chapter 7 points out previous work related to fault-tolerant design, multi-scenario handling, and corresponding evolutionary techniques. The conclusion and future work are presented in Chapter 8.

Chapter 2

Preliminaries

2.1 Distributed Programs

In this section, we define the notion of a program and its computation. We also identify how the probabilities are specified in such a program and how they are interpreted. These definitions are based on guarded commands defined by Dijkstra [9] that have been used widely in the literature (e.g. [10, 11]).

2.1.1 Distributed Programs Structure

A distributed program consists of a set of variables and a set of actions. Each action is of the form $guard \longrightarrow st$ where $guard$ is a Boolean expression over program variables and st updates program variables.

Program. A program p is specified in terms of a set of processes. In turn, each process is associated with a set of variables and a set of actions. Furthermore, each variable is associated with a finite domain. An action of program p is of the form:

$$\begin{aligned} Action\_name :: \langle guard \rangle \longrightarrow pr_1 \quad & \langle statement_1 \rangle, \\ pr_2 \quad & \langle statement_2 \rangle, \\ & \ldots; \end{aligned}$$

where 1) $Action\_name$ is the name given to the action, 2) guard is a Boolean expression involving program variables (i.e., the union of all process variables), 3) statement(s) updates one or more variables of the process, and 4) $pr_1, pr_2 \cdots$ are real numbers ranging in [0..1] which represent probabilities of executing corresponding statements. If there is only one statement for a guard, then usually the probability notation is omit from the program. In this case, it implies that when the guard is true, the probability of executing the statement is equal to 1. Furthermore, the sum of probability values associated with a given guard is 1.

State space and state predicate. For such a program, its state is obtained by assigning each program variable a value from its domain. The state space of the program is the set of all possible states of the program. A state predicate of program is a subset of its state space.

Program computation. A sequence $\mathbf{s_0}, \mathbf{s_1}, \cdots$ is a computation of given program p iff $\forall j : j >= 0 : (s_j, s_{j+1})$ is a step of program p.

Invariant S is an invariant of program p for specification iff $S \neq \{\}$ and p refines the specification from S.

Enabled. We say that action is enabled in state s iff the guard of that action evaluates to true in state s. For simplicity, we assume that if a process has multiple actions then the guards corresponding to those actions are disjoint. In other words, at most one action is enabled in a given state.

We say that an action is enabled if its guard is true in the current state. We consider execution of programs under two semantics:

Interleaving semantics. In interleaving semantics, in one step, we choose one enabled action to execute. If multiple actions are enabled, one of them is chosen non-deterministically. The new state of the system is obtained by executing the selected action. In other words, a program computation is a sequence $\langle s_0, s_1, \cdots \rangle$ where

- $\forall j : j > 0 : s_{j+1}$ is obtained by after executing any action enabled in state s_j, and
- If the sequence is finite and terminates in state s_l then no actions are enabled in state s_l.

This process is repeated indefinitely or until we reach a state where no actions are enabled.

Maximum parallelism semantics. In maximum parallelism semantics, in one step, we choose enabled actions of all processes to execute. All these actions utilize the current state to evaluate the guards and execute statements. (In other words, if two actions are chosen for execution together in maximum parallelism semantics, then their executions are not visible to each other.) The new state of the system is obtained by executing all actions simultaneously. This process is repeated indefinitely until we reach a state where no actions are enabled.

Interleaving semantics is useful when we can serialize the actions of different processes. Examples of such systems have been discussed in [12, 13]. Maximum parallelism semantics captures executions of round-based algorithms where in each round, each process communicates information with other processes and utilizes this information to decide its own successive state. It can also be utilized to define time-driven systems such as TTA [14]. A program designed in maximum parallelism semantics can be implemented in a distributed system by adding a round number to every message and requiring every process to act only on messages received in current round. Examples of such protocols include [15, 16, 17].

2.2 Fault-tolerant System

A system is called fault-tolerant when the system returns to its legal configuration after all faults stop executing[2]. A fault-tolerant program satisfies specification in the absence of faults as well as in the presence of faults. Stabilization is a type of fault-tolerance.

2.2.1 Self-stabilization

A stabilizing program [9] ensures that starting from an arbitrary state, the program recovers to its legitimate states. Moreover, after reaching a legitimate state, in the absence of faults, it remains in legitimate states forever. Thus, a stabilizing programs ensures that it can recover to its legitimate state from any transient fault. Examples of stabilizing systems include [9, 18, 19, 20, 21]. These include programs [9, 20, 21] that ensure that any computation of the program will inevitably reach the legitimate states as well as programs [18, 19] that guarantee that legitimate states will be reached.

Intuitively, a program p is stabilizing (or self-stabilizing) with invariant S iff every computation of p (starting from an arbitrary state) eventually reaches a state in S, and stays in S thereafter. Thus, a stabilizing program satisfies the following two constraints (properties):

- **Closure:** Starting from a state in S, if any enabled action of p is executed then the resulting state is in S.
- **Convergence:** Starting from an arbitrary state, say s_0, every computation of p eventually reaches a state in S.

2.2.2 Probabilistic Self-stabilization

As mentioned, we say that a program p stabilizes to state predicate S iff (1) if p starts from an arbitrary state, and it reaches a state in S with probability 1, and (2) if p starts from a state in S then it remains in S forever.

Probabilistic stabilization is related to probabilistic computations, where the probability of reaching a state in S is 1, i.e., the probability of computation staying outside S forever is 0.

An Example. To illustrate the definitions in this dissertation, we use a simple example where we have two processes j and k, and two program variables x and y. The actions of the program are as follows:

$$
\begin{array}{llcll}
A_{j1} :: & (x \neq y) & \longrightarrow & 0.6 & x = y \\
 & & \longrightarrow & 0.4 & \text{skip} \\
A_{j2} :: & (x = y) & \longrightarrow & 1 & \text{skip} \\
A_{k1} :: & (x \neq y) & \longrightarrow & 0.6 & y = x \\
 & & \longrightarrow & 0.4 & \text{skip} \\
A_{k2} :: & (x = y) & \longrightarrow & 1 & \text{skip}
\end{array}
$$

In this program, if x and y are unequal then process j sets x to be equal to y with probability 0.6, and it leaves x unchanged with probability 0.4. Process k behaves similarly except that it changes the value of y. Furthermore, we consider maximum parallelism (or synchronous) execution where all processes execute in one step.

Now, starting from a state where $x = 1$ and $y = 0$. Then, with probability 0.36 process j and k change both x and y and, hence, the resulting state is one where $x = 1$ and $y = 0$. With probability 0.16, neither process changes its value. And, with probability 0.48, only one process changes the value causing x and y values to be equal.

In this program, the probability that the computation will ever reach a state where x and y are equal is 1. In other words, the probability that the computation proceeds where x and y always differ can be made as small as possible. Moreover, once x and y are equal, there is no change in either of them. Thus, the above program is stabilizing to state predicate S that denotes that x and y values are equal.

2.3 Evolutionary Algorithm (EA)

2.3.1 Genetic Algorithm

GA [22, 23], as one of the most popular EAs, is a guided search and an optimization technique, inspired by the biologic evolution. The solution to a problem is encoded into artificial chromosomes (genomes), and these genomes preserve the problem structure and information during the evolution. The implementation of GA is as follows. It begins with a population of randomly generated genomes, then GA evaluates genomes using one or more objective functions (or fitness) and assigns objective values (or fitness value). These genomes are the population of first generation. After that GA selects and recombines the population of current generation (called parents) to create population for next generation (called offsprings). Iteratively, GA evolves the populations that the average fitness value gets better and better until some stopping criteria are reached. The stopping criterion can be either the number of generation reaches the maximum allowed generation or the optimal solutions are found.

2.3.2 Genetic Operators

The new population are created by two genetic operators - crossover and mutation. The crossover operator recombines selected parent genomes to produce offsprings. This is done with a crossover probability p_c. By recombining good genomes, it is likely to create better solution. Mutation operator modifies solutions by randomly changing some part of the genome with a mutation probability p_m. Mutation maintains diversity within the population.

2.3.3 Objective Function

In EA, objective function is to evaluate the quality of the genome. Depends on the number of objectives, optimization can be classified into three categories: single-objective, multi-objective and many-objective case. It is called single-objective if there are only one objective for the problem, multi-objective if there are two or three objectives, and many-objective if there are more than three objectives. In this dissertation, we consider first two cases.

2.3.4 Multi-objective Optimization

In multi-objective optimization, there can be a set of optimal solutions, especially when optimizing the conflicting objectives. This is due to the trade-off between objectives. It compares two solutions using domination concept. The domination concept is defined as follow. A solution A dominates B, 1) if A is no worse than B in all objectives, and at the same time 2) if A is strictly better than B in at least one objective. A solution is called non-dominated, if no other solutions dominate it. The set of non-dominated solutions shows the trade-off among the different objectives. For example, Figure 2.1 shows the optimal solutions (A to G) for bi-objective ($f1, f2$) minimization problem. E is better than solution H, since E's two objectives are smaller than H's. It is called E dominates H. For E and D, E is better in $f2$ but worse in $f1$, hence, E and D are non-dominated to each other. The non-dominated solutions constitute a front and the front is called non-dominated front, and the optimal non-dominated front is called Pareto Optimal front. The goal of multi-objective optimization is to find the Pareto Optimal front.

NSGA-II: A fast elitist multiobjective genetic algorithm In this dissertation, we apply NSGAII [24] to solve multi-objective optimization, and modified NSGA-II to solve MSMO problem. NSGA-II is one of the state-of-art multi-objective GAs. NSGA-II preserves good solutions, and

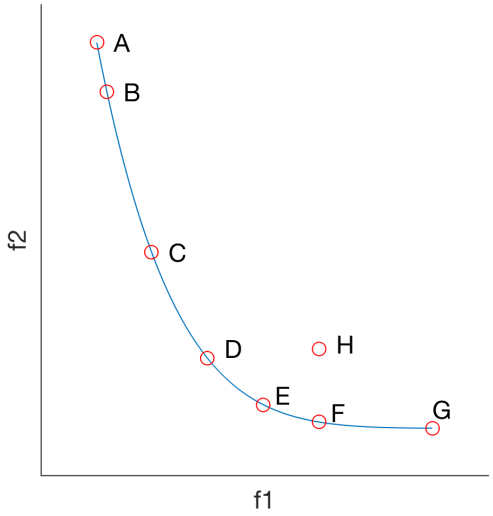

Figure 2.1: Optimal Solutions in Multi-Objective Optimization: Trade-off between Objectives

at the same time, maintains diversity of the search. It uses non-dominated sorting which compares two program by the concept of domination. Non-dominated solutions in a generation are called the non-dominated solutions of first rank and considered as best ones in that generations. If non-dominated solutions of first rank are removed from the population, then another set of non-dominated solutions are emerged from the remaining population and called non-dominated solutions of second rank. Accordingly, NSGA-II sorts the population rank by rank, and in each rank the solutions are compared based on the crowding distance. Less crowded solution is considered better. Using non-dominated sorting and crowding distance, NSGA-II converges to the optimal front as well as maintain the diversity among the solutions in the front.

2.3.5 Genetic Programming

Genetic programming (GP) is also an evolutionary method that models problems, especially computer programs. GP starts with an initial population consisting of individual programs and evolves them into optimized ones. In each generation, GP evaluates generated programs and calculates the

objective functions to check if the program satisfy all program requirements. Based on objective functions, GP selects candidates, i.e., programs that have smaller objective value (for minimization problem), and applies computational analogs of biological mutation and crossover to reproduce new programs. By iterating this process the best programs that satisfy all program requirements.

Chapter 3

Description of Scenario-dependent Problems

3.1 Introdunction

Fault-tolerant design requires solutions that work in different scenarios of the problem: in presence of faults or in absence of faults. Regardless of faults, some design problem requires solution to work in other scenarios. Other scenarios can be different loadings, variations of the problem, operational conditions, etc. Solutions are usually evaluated for their performance over a number of scenarios, and solutions are required to be optimal in all required scenarios at the same time. We call these optimization problems as scenario-dependent MSMO problem. For instance, in fault-tolerant program synthesis, a designer needs to test each generated program in many different environments: with no fault, with single fault, with multiple faults of the same type or different types. Also, in structural engineering field, for instance, when designing a bridge over a river, a structural engineer will design the bridge for different loading conditions as well as potential structural damage conditions. These conditions can be (i) a scenario depicting a bumper-to-bumper traffic jam over the bridge providing the largest vertical loading that the bridge has to withstand, (ii) a scenario for which a series of cars zipping through the bridge at a very high speed, thereby causing a periodic dynamic load, (iii) a scenario where some parts of the bridge are partially damaged but

usable. Over a span of 100 years of life of the bridge, each of these scenarios has a finite probability to occur and the designer must take into consideration each of such scenarios in designing the bridge and make sure that the bridge is safe under each of these scenarios. For scenario-dependent MSMO fault-tolerant problems, it is very important that all scenarios are taken into consideration during the optimization. If we consider each scenario independently and optimize or model problem only for single scenario, we will fail to find the desired solutions for other scenarios. It is clear that the final design must satisfy all constraints from all scenarios.

The optimization under multiple scenarios is challenging due to potential conflicts and trade-offs among scenarios. The optimal solutions from one scenario can perform poorly in other scenarios, and also solutions that are optimal in all scenarios may not exist. For instance, consider the bi-scenario, bi-objective problem given below, and the goal is to find an optimal solution that minimizes two objectives, $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$, in both scenarios. The detailed description of $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ can be found in Equation 4.14 of Section 4.4.2. The calculation of f_1 and f_2 are different in two scenarios. To clarify, we denote two objectives as f_1^1, f_2^1 in scenario 1, and f_1^2, f_2^2 in scenario 2.

$$\begin{aligned} &\text{Minimize} \quad \left\{ f_1^1(\mathbf{x}), f_2^1(\mathbf{x}) \right\} in\ scenario\ 1, \\ &\text{Minimize} \quad \left\{ f_1^2(\mathbf{x}), f_2^2(\mathbf{x}) \right\} in\ scenario\ 2 \end{aligned} \tag{3.1}$$

Instead of considering both scenarios at the same time, we individually optimize each scenario using NSGA-II. Two efficient sets X^1 and X^2 are obtained after two optimizations, where X^1 is the solution set from minimization of f_1^1 and f_2^1, and X^2 is from minimization of f_1^2 and f_2^2. Then, we plot both solution sets in scenario 1 and scenario 2 objective space respectively in Figure 3.1 and 3.2.

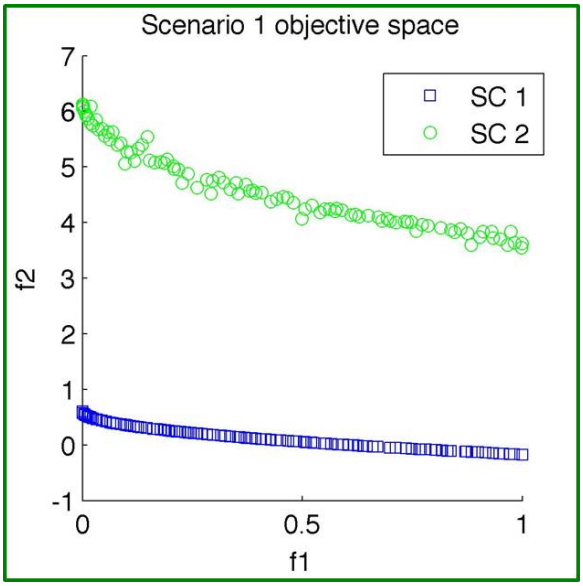

Figure 3.1: Scenario 1 objective space

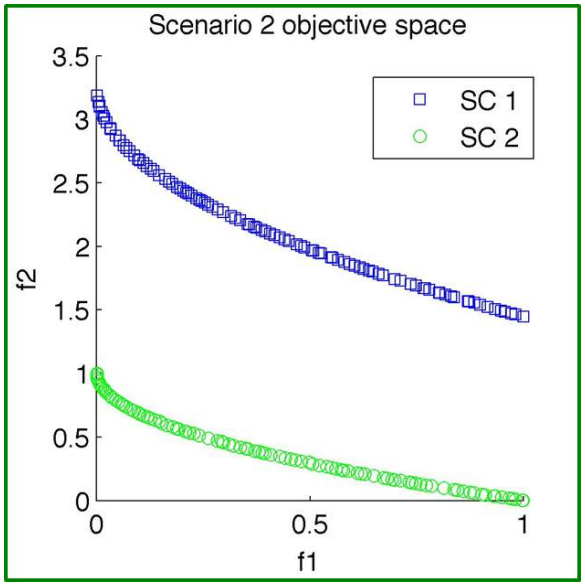

Figure 3.2: Scenario 2 objective space

For scenario 1, we calculate f_1^1 and f_2^1 for solution sets X^1 and X^2 and plot the corresponding set of objective values $SC1$ and $SC2$ in Figure 3.1. Similarly, for scenario 2, we calculate f_1^2 and f_2^2 for X^1 and X^2, and plot them in Figure 3.2. In Figure 3.1, $SC1$ forms Pareto-optimal front as expected, however, the $SC2$ which are optimal in scenario 2 performs very poorly in scenario 1, and vice versa. Solutions from these two optimal solution sets are not desirable. In order to gain the optimal in one scenario, it needs to sacrifice in the other scenario.

Finding solutions whose objectives arrive at the optimal throughout the all scenarios is not possible for many problem. Thus, the way to solve problem is using scenario treatment to obtain compromised optimal designs from different scenarios. This also the main goal of this work.

3.2 General Description

Considering all scenarios in each solution, the general scenario-dependent MSMO optimization problem consists of an optimization problem with K different scenarios and $M^{(k)}(M^{(k)} > 0)$ objectives for k-th scenario. The general structure of scenario-dependent problem is illustrated in

Figure 3.3. In this figure, there are total of K scenarios, and each scenario has more than one objectives to optimize. The optimization considers all objectives in each scenario. The optimal solutions from one scenario are not necessarily optimal in other scenarios. Instead of evaluating a solution ($\mathbf{x}$) for single scenario here, it must be evaluated for K different scenarios.

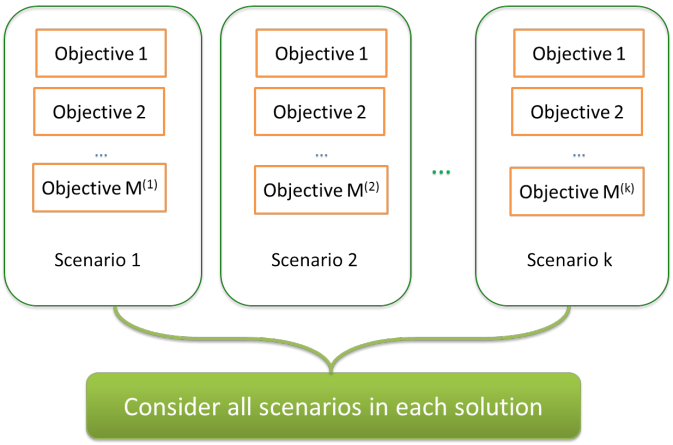

Figure 3.3: Consider all scenarios in one optimization

It is important to realize that in the context of a design problem, the solution to the problem is already implemented before any of the scenarios is applied in practice. Thus, the optimization problem to arrive at the optimized solution shares the same design variables (solutions) and their values for all specified scenarios. Importantly, every feasible solution to the problem must be feasible for all specified scenarios. Usually, if there is no single feasible solution that is optimal for all scenarios, the final feasible optimized solutions are a set of compromised solutions balancing optimality of objectives of all scenarios.

Let us denote $f_m^{(k)}(\mathbf{x})$ as the m-th objective function value for the k-th scenario, and $g_j^k(\mathbf{x}) \leq 0$ as the j-th inequality constraint for the k-th scenario. For brevity, we do not consider equality constraints here, but they can be handled in a similar way as well. A generic K-scenario, M-

objective, and J-constraint problem can be written as follows:

$$\begin{aligned}
\text{Minimize} \quad & \left\{ f_1^{(1)}(\mathbf{x}), f_2^{(1)}(\mathbf{x}), \ldots, f_{M^{(1)}}^{(1)}(\mathbf{x}) \right\}, \\
\text{Minimize} \quad & \left\{ f_1^{(2)}(\mathbf{x}), f_2^{(2)}(\mathbf{x}), \ldots, f_{M^{(2)}}^{(2)}(\mathbf{x}) \right\}, \\
& \ldots \\
\text{Minimize} \quad & \left\{ f_1^{(K)}(\mathbf{x}), f_2^{(K)}(\mathbf{x}), \ldots, f_{M^{(K)}}^{(K)}(\mathbf{x}) \right\} \\
\text{Subject to} \quad & g_j^k(\mathbf{x}) \leq 0, \quad j = 1, 2, \ldots, J^{(k)}, \; k = 1, 2, \ldots, K.
\end{aligned} \tag{3.2}$$

The scenario-dependent MSMO problem is different from dynamic optimization. In the latter, the objectives, or constraint functions, or problem parameters change with time. As a result, the optimal solution is also expected to change with time. However, each trade-off optimal solution in scenario-dependent MSMO must make a balance of optimizing all scenarios and satisfy constraints arising from all scenarios. It is also different from robust optimization in where the scenario-dependent MSMO finds solutions which are relatively insensitive to uncertainties of decision variables or problem parameters. Meanwhile, in scenario-dependent MSMO, a few different formulations of objective and constraint functions arising from different scenarios are considered, usually not from an uncertainty consideration of any kind.

One simple idea to solve the scenario-dependent MSMO problem is to consider all $\sum_{n=1}^{K} M^{(n)}$ objectives independently and convert the above MSMO optimization problem into a many-objective optimization problem. However, the resulting problem has too many objectives that produce too many trade-offs causing all solutions to be non-dominated even in early generations. Also, the user may not actually be interested in achieving a trade-off among all such objectives coming out of multiple scenarios. Instead, in this work, we search for compromised solutions that provide trade-offs either among scenarios or among objectives depending on problems.

In the next two chapters, we consider two different approaches to handle the scenarios and objectives in scenario-dependent problem: objective-wise MSMO approach (Chapter 4) and scenario-wise MSMO approach (Chapter 5).

Chapter 4

Objective-wise Approach for Scenario-dependent MSMO Problems

4.1 General Description

In objective-wise approach, each objective throughout the scenarios is considered as a criterion. Figure 4.1 shows the structure of the objective-wise approach. In this approach, we assume the objectives across the scenarios have the same meaning but possibly different calculation, and the number of the objectives in each scenario is the same. For instance, in the bridge design mentioned in Section 3.1, the first objective in all scenarios can be nodal deflections, and the calculation of deflection in different scenarios is different. The second objective is the stress, and third one is the volume, etc. Each criterion considers corresponding objective in all scenarios.

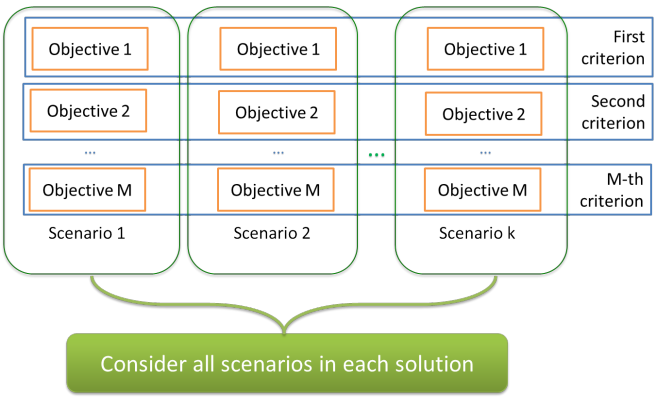

Figure 4.1: Objective-wise Approach

The general problem description for objective-wise approach is very similar to the one in the Section 3.2 except all scenarios have the same number of objectives. Let us denote $f_m^{(k)}(\mathbf{x})$ as the m-th objective function value for the k-th scenario, and $g_j^k(\mathbf{x}) \leq 0$ is the j-th inequality constraint for the k-th scenario. The K-scenario, M-objective, J-constraint scenario-dependent MSMO problem for objective-wise approach that is solved by objective-wise approach can be written as follows:

$$\begin{aligned}
&\text{Minimize} \quad \left\{ f_1^{(1)}(\mathbf{x}), f_2^{(1)}(\mathbf{x}), \ldots, f_M^{(1)}(\mathbf{x}) \right\}, \\
&\text{Minimize} \quad \left\{ f_1^{(2)}(\mathbf{x}), f_2^{(2)}(\mathbf{x}), \ldots, f_M^{(2)}(\mathbf{x}) \right\}, \\
&\qquad \ldots \\
&\text{Minimize} \quad \left\{ f_1^{(K)}(\mathbf{x}), f_2^{(K)}(\mathbf{x}), \ldots, f_M^{(K)}(\mathbf{x}) \right\} \\
&\text{Subject to} \quad g_j^k(\mathbf{x}) \leq 0, \quad j = 1, 2, \ldots, J^{(k)}, \; k = 1, 2, \ldots, K.
\end{aligned} \tag{4.1}$$

In this work, we proposes two methods - aggregated method and integrated method to solve the above problem. Aggregated method is combining all scenarios for each objective, then the original problem is converted into multi-objective problem optimizing all criteria. Integrated method, on the other hand, consider each objectives in each scenarios respectively during the optimization. These two methods are given in details in Section 4.2 and 4.3. We discuss these methods in the context of an evolutionary multi-objective optimization (EMO) algorithm in this dissertation.

4.2 Aggregated Method

The aggregated method for handling multiple scenarios in scenario-dependent MSMO problem is to combine all scenarios for each objective function in an aggregated manner. The resulting

problem becomes as follows:

$$\begin{aligned} &\text{Minimize} \quad \left\{ \sqcup_{k=1}^{K} f_1^{(k)}(\mathbf{x}), \sqcup_{k=1}^{K} f_2^{(k)}(\mathbf{x}), \ldots, \sqcup_{k=1}^{K} f_M^{(k)}(\mathbf{x}) \right\}, \\ &\text{Subject to} \quad g_j^{(k)}(\mathbf{x}) \leq 0, \quad j = 1, 2, ..., J^{(k)}, \ k = 1, 2, \ldots, K. \end{aligned} \tag{4.2}$$

The operator $\sqcup$ denotes the aggregate operator. Note that the above formulation allows a different aggregate function to be used for every objective, if desired. We describe the following two aggregate functions, which are popular and also used in this study. Notice that due to the use of all problem constraints for each scenario as strict constraints to the above problem, the obtained solutions are always feasible under each scenario.

4.2.1 Worst-case Aggregation

For a minimization problem, the aggregate function is the $\max$ function evaluating the worst value for all K scenarios:

$$\sqcup_{k=1}^{K} f_m^{(k)}(\mathbf{x}) = \max_{k=1}^{K} f_m^{(k)}(\mathbf{x}). \tag{4.3}$$

This is by far the most popularly used aggregate function in practice. It makes the most pessimistic case and often times the resulting solutions are "over-designs" for some scenarios.

4.2.2 Average-case Aggregation

For average aggregation, the aggregate function is a weighted average of all scenarios:

$$\sqcup_{k=1}^{K} f_m^{(k)}(\mathbf{x}) = \sum_{k=1}^{K} w_k f_m^{(k)}(\mathbf{x}), \tag{4.4}$$

for which a predefined weight vector $\mathbf{w}$ is supplied to indicate the preference of some scenarios over the other. If objective functions are normalized to take values within the same range, weights can be adjusted so that $\sum_k w_k = 1$. This way, weights may mean the importance given to respective scenarios. Some other aggregate functions [25] can also be used.

The above two formulations aggregate multiple scenarios to produce a single aggregated function for each objective, thereby making the above problem a standard M-objective optimization problem. Thereafter, any EMO algorithm [26] or any generative classical multi-objective optimization method [27] can be used to find a set of trade-off Pareto-optimal solutions corresponding to the formulated aggregate functions. The trade-off solutions can be visualized as a trade-off front in each scenario space separately to understand the trade-off among objectives in different scenarios. An analysis of multiple fronts(one front for each scenario) can provide additional information to the decision-makers, but individual front from one scenario is not guaranteed to have solutions that are all non-dominated to each other. This aspect may make the decision-making difficult and warrant another MCDM process to get a clear idea of the trade-off information in the obtained solutions. Nevertheless, the above aggregate is simple-minded and is a natural extension of single-scenario optimization for multiple scenarios, and we have used NSGA-II [24] to solve the trade-off aggregated problem, although other EMO methods can also be used. It is interesting to note that the above aggregated method also easily extends to a multi-scenario, single-objective optimization problem.

4.3 Integrated Method

Next, we discuss our proposed integrated method, which is more involved than the aggregated method, and it provides the user with a more balanced and compromised trade-off front. The

integrated method considers every scenario independently and integrates the information from all scenarios in an EMO's selection operator. The integrated method requires a pre-processing of every scenario to find reference points before the integrated multi-objective optimization is applied. We discuss this procedure next.

4.3.1 Reference Points Determination Procedure

The overall reference point determination procedure is presented in Figure 4.4. In Step 1, every scenario (say, k-th one) is considered individually and the respective trade-off front (having a solution set $\mathcal{X}^{(k)}$) is found by using any EMO procedure (here, we have used NSGA-II [24]). After finding all K trade-off fronts (assume there are total K scenarios), they are all considered for each scenario one at a time, as follows. Every trade-off solutions are evaluated for all scenarios. It is obvious that the front $\mathcal{X}^{(k)}$ optimized for k-th scenario will stay as the best non-dominated front compared to all other fronts in the k-th scenario space and other fronts will be either dominated or stay non-dominated with the $\mathcal{X}^{(k)}$ front-solutions. This is because, objectives from other scenarios were not considered while optimizing the k-th scenario problem. Figures 4.2 and 4.3 show the optimized trade-off solutions of a specific two-scenario, two-objective problem. In the Scenario 1 objective space (shown in left figure), the non-dominated solutions (marked in blue squares) are found from Scenario 1 optimization and the dominated solutions (marked in green circles) comes from Scenario 2 optimization. The opposite situation occurs when the same trade-off solutions are plotted in Scenario 2 objective space (shown in right figure).

This is not surprising since the optimized solutions for a specific scenario should not get dominated by any other solutions when they are plotted in this specific scenario objective space. It is interesting to note that not all Scenario 2 optimized solutions appear as non-dominated to each other when they are plotted in Scenario 1 objective space, however when the same Scenario 2 so-

lutions are plotted in its own scenario objective space (right figure), they all appear non-dominated to each other. This is a typical behavior of individually optimized trade-off solutions highlighting the need for obtaining a compromise set of solutions in a MSMO optimization problem.

In Step 2, all scenario-wise optimized solutions combined into a collective set ($\mathcal{S} = \cup_{k=1}^{K} \mathcal{X}^{(k)}$), and are normalized using the best and worst objective values of each scenario. Thus, in each scenario (say, k), the ideal point $\mathbf{z}^{k,ideal}$ of set $\mathcal{S}$ is simply the origin and the nadir point $\mathbf{z}^{k,nadir}$ is a vector of ones. Figures 4.2 and 4.3 show that two sets span over $[0, 1]$ range on each objective axis in both scenario objective spaces.

Then, in Step 3, for each scenario (say, k), a number of achievement scalarizing functions (ASFs) [28] are formed using the ideal point $\mathbf{z}^{k,ideal}$ and a set of H well-distributed weight vectors $\mathbf{w}^{(j)}$ (for $j = 1, 2, \ldots, H$):

$$\text{ASF}^{(k,j)}(\mathbf{x}) = \max_{i=1}^{M} \left(f_i^{(k,j)}(\mathbf{x}) - z_i^{k,0} \right) / w_i^{(j)}. \tag{4.5}$$

The standard Das and Dennis's method [29] is used to create weight vectors in this study. For the j-th ASF problem(weight vector), points having the best ($\mathbf{x}_{best}^{(k,j)}$) and worst ($\mathbf{x}_{worst}^{(k,j)}$) ASF values from the collective set are identified, as follows:

$$\mathbf{x}_{best}^{(k,j)} = \text{argmin}_{\mathbf{X} \in \mathcal{S}} \text{ASF}^{(k,j)}(\mathbf{x}), \tag{4.6}$$

$$\mathbf{x}_{worst}^{(k,j)} = \text{argmax}_{\mathbf{X} \in \mathcal{S}} \text{ASF}^{(k,j)}(\mathbf{x}). \tag{4.7}$$

Thereafter, in Step 4, a mean objective vector ($\mathbf{z}^{(j)}$) is calculated as a reference point for j-th

weight vector in our proposed integrated method:

$$z_i^{(k,j)} = 0.5\left(f_i^{(k)}(\mathbf{x}_{best}^{(k,j)}) + f_i^{(k)}(\mathbf{x}_{worst}^{(k,j)})\right). \tag{4.8}$$

Figures 4.2 and 4.3 show $H = 10$ calculated reference points in diamonds on Scenarios 1 and 2, respectively.

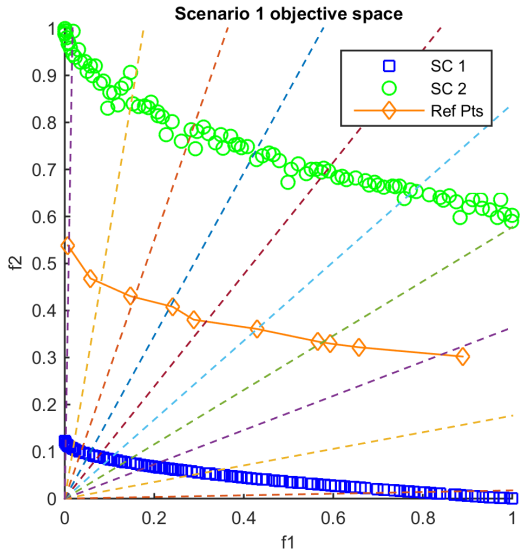

Figure 4.2: Individual non-dominated solutions, 10 Reference lines, and corresponding reference points are shown in Scenario 1 space.

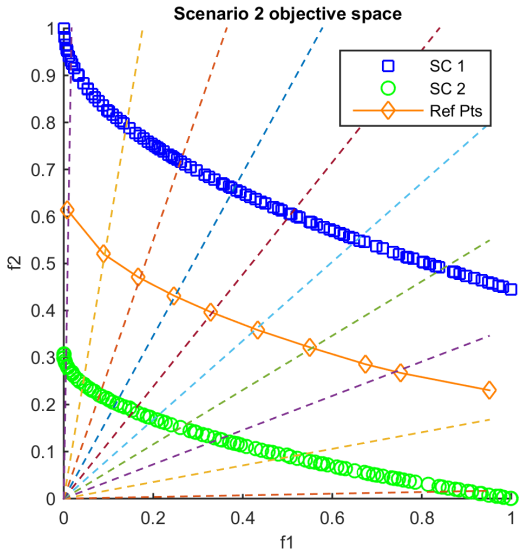

Figure 4.3: Individual non-dominated solutions, 10 Reference lines and corresponding reference points are shown in Scenario 2 space.

Step 1: Optimize to find individual scenario-wise efficient solution set $\mathcal{X}^{(k)}$ for each (k th) scenario using an EMO algorithm.

Step 2: Combine all sets together ($\mathcal{S}$) and normalize objective vectors in each scenario (k) by using minimum and maximum k-th scenario objective values from $\mathcal{S}$.

Step 3: Choose H reference directions uniformly like in MOEA/D [30] or NSGA-III procedures [31] and apply the ASF method to choose two extreme objective vectors (having minimum and maximum ASF values) from $\mathcal{S}$ for each reference direction (j) and in each scenario (k).

Step 4: Compute H reference points $\mathbf{z}^{(k,j)}$ as the mean of the resulting extreme points.

Figure 4.4: Reference points determination procedure.

Once the reference points are calculated by the above four-step procedure, the main optimization process starts by using the already-found non-dominated solutions from each scenario as the initial population for the integrated method. We modify the definition of domination, crowded rank computation, and constraint handling method of the NSGA-II procedure [24] for handling multiple scenarios, as described in the following subsections.

4.3.2 Scenario-based Domination Principle

Unlike in a single-scenario problem, here, a solution $\mathbf{x}$ might dominate another solution $\mathbf{y}$ in one scenario, but $\mathbf{x}$ might not dominate $\mathbf{y}$ in another scenario. Thus, we need to modify the definition of domination for a multi-scenario problem so that both such solutions are emphasized by the selection operator. For this purpose, we define a *scenario-based domination* concept, which is depicted for a two-scenario problem in Table 4.1. The concept can be applied to more than two or three scenarios as well.

Table 4.1: Scenario-based domination for two scenarios.

SC 1 / SC 2	$\mathbf{x}$ flipped-ϵ-dominates $\mathbf{y}$	ϵ-nondominate	$\mathbf{y}$ flipped-ϵ-dominates $\mathbf{x}$
$\mathbf{x}$ flipped-ϵ-dominates $\mathbf{y}$	Choose $\mathbf{x}$	Choose $\mathbf{x}$	Compare CR
ϵ-non-dominated	Choose $\mathbf{x}$	Compare CR	Choose $\mathbf{y}$
$\mathbf{y}$ flipped-ϵ-dominates $\mathbf{x}$	Compare CR	Choose $\mathbf{y}$	Choose $\mathbf{y}$

The concept of ϵ-domination was introduced elsewhere [32, 33], but is used here in an flipped sense. Here, a solution $\mathbf{x}$ flipped-ϵ-dominates a solution $\mathbf{y}$, if $f_i(\mathbf{x}) \leq f_i(\mathbf{y}) - \epsilon_i$ for all i and $f_i(\mathbf{x}) < f_i(\mathbf{y}) - \epsilon_i$ for at least one i. Figure 4.5 shows that a point A dominates all solutions that lie above the shifted quadrant by ϵ_1 and ϵ_2 in f_1 and f_2 axis, respectively. Interestingly, a solution in the strip between the two quadrants (one passing through A and the other shifted one) is ϵ-non-dominated to A, although it would be dominated by the usual domination principle. This

principle allows certain dominated solutions of one scenario to be considered in another scenario in order to find a compromise set of solutions.

The concept of scenario-based domination is defined as follows. $\mathbf{x}$ scenario-dominates $\mathbf{y}$ in following two cases: (i) $\mathbf{x}$ flipped-ϵ-dominates $\mathbf{y}$ in all scenarios, or (ii) $\mathbf{x}$ flipped-ϵ-dominates $\mathbf{y}$ in some scenarios, and $\mathbf{x}$ is ϵ-non-dominated with $\mathbf{y}$ in all other scenarios. If $\mathbf{x}$ does not scenario-dominate $\mathbf{y}$, and $\mathbf{y}$ does not scenario-dominate $\mathbf{x}$, then we call $\mathbf{x}$ and $\mathbf{y}$ are scenario-non-dominated. For further selection, when two solutions are scenario-non-dominated, we choose the one with better crowded rank (CR). The scenario-based crowded rank metric is defined in the next subsection.

Figures 4.5 and 4.6 describe three hypothetical solutions in a two-objective minimization problem having two scenarios.

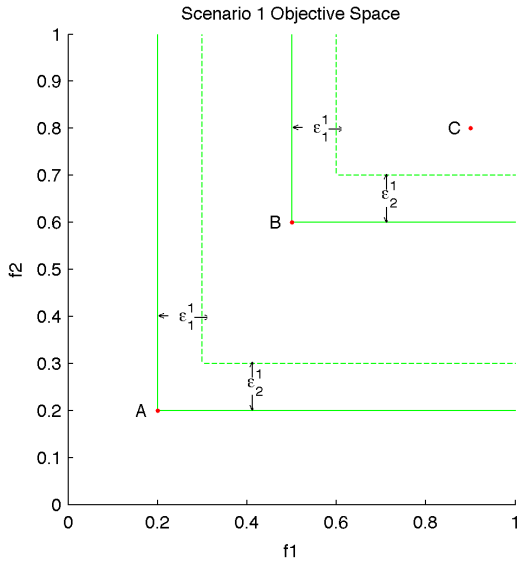

Figure 4.5: Three points in Scenario 1.

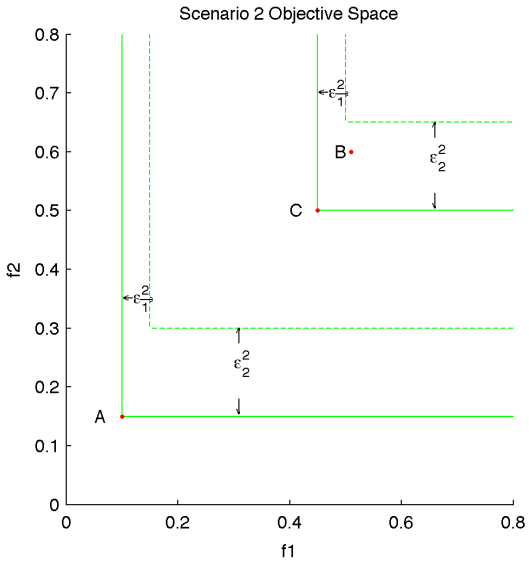

Figure 4.6: Same three points in Scenario 2.

In Scenario 1, A flipped-ϵ-dominates B and C, and B flipped-ϵ-dominates C. In Scenario 2, A flipped-ϵ-dominates B and C as well, but C ϵ-nondominates B. According to the scenario-based domination concept describe above, solution A scenario-dominates both solutions B and C, since A flipped-ϵ-dominates B and C in both scenarios. However, solution B scenario-dominates C, since B flipped-ϵ-dominates C in scenario 1 and is ϵ-non-dominated to C in Scenario 2.

Since the domination check is performed throughout all scenarios, the optimal solutions of a specific scenario cannot be dominated by any other solution (optimal or otherwise in another scenario) in the same specific scenario space.

4.3.3 Scenario-based Crowded Rank Procedure

Next, we present the *scenario-based crowded rank* computation procedure which is used in the scenario-domination concept described in Subsection 4.3.2. The purpose of calculating and using the crowded rank metric is to maintain the diversity among evolving population members in NSGA-II. In addition, there is an important task of maintaining diversity of solutions in all scenarios. This is where we use the reference points calculated in Subsection 4.3.1. First, to preserve the extreme points in each scenario space, they are assigned a rank of zero. Then, for each scenario (say k-th one), we compute a set of H reference points ($\mathbf{z}^{(k,j)}$) located midway between the best and worst points identified for j-th ASF scalarization. In order to compute the scenario-based crowded rank for a solution $\mathbf{x}$, we first compute the crowded rank $\mathrm{CR}^{(k)}(\mathbf{x})$ for each scenario by first associating the solution $\mathbf{x}$ with its nearest (Euclidean distance sense) reference point (say j), and then determining the sorted rank of the solution among all associated population members of the same (j-th) reference point. The lower the rank of a solution, the better is solution, as our goal to find a distinct solution close to each reference point. Thereafter, overall crowded rank ($\mathrm{CR}^{(k)}(\mathbf{x})$) is computed as follows:

$$\mathrm{CR}(\mathbf{x}) = \mu_{\mathrm{CR}}(\mathbf{x}) + \kappa\sigma_{\mathrm{CR}}(\mathbf{x}), \tag{4.9}$$

where $\mu_{\mathrm{CR}}(\mathbf{x})$ and $\sigma_{\mathrm{CR}}(\mathbf{x})$ are the mean and standard deviation of different $\mathrm{CR}^{(k)}$ values of x in all scenarios. We choose $\kappa = 2$ to provide a reasonable importance to the standard deviation in

the rank computation process. For example, for a two-scenario problem, the crowded rank (CR) is calculated from individual CR-values for a scenario as follows:

$$\mathrm{CR}(\mathbf{x}) = 0.5\left(\mathrm{CR}^{(1)}(\mathbf{x}) + \mathrm{CR}^{(2)}(\mathbf{x})\right) + |\mathrm{CR}^{(1)}(\mathbf{x}) - \mathrm{CR}^{(2)}(\mathbf{x})|$$

Figure 4.7 describes the crowded rank computation procedure in detail.

Step 1:	For scenario k, the extreme points are assigned a crowded rank of zero. For all other points, following steps are performed.
Step 2:	For scenario k, for each population member $\mathbf{x}$, find the nearest reference point $\mathbf{z}^{(k,j)}$. Associate $\mathbf{x}$ with the reference point $\mathbf{z}^{(k,j)}$.
Step 3:	For each (say j-th) reference point $\mathbf{z}^{(k,j)}$, sort the associated members based on ascending order of their Euclidean distance from $\mathbf{z}^{(k,j)}$ in the objective space. Note the rank of $\mathbf{x}$.
Step 4:	Compute the crowded rank $\mathrm{CR}(\mathbf{x})$ by using mean and standard deviation of CR-values of from different scenarios.

Figure 4.7: Crowded rank $\mathrm{CR}(\mathbf{x})$ computation for a solution $\mathbf{x}$.

4.3.4 Hierarchical Constraint Handling Procedure

Next, we describe the procedure for handling constraints in a multi-scenario problem. In a design optimization problem, a feasible design must satisfy all constraints specified for each and every scenario. If a solution violates any constraint in any of the scenarios (of $g_j^{(k)} \leq 0$ type), it is not an acceptable solution, as this means that the design solution will fail when that particular scenario occurs. We compute constraint violation of a solution $\mathbf{x}$ for k-th scenario, in the following manner:

$$\mathrm{CV}^{(k)}(\mathbf{x}) = \sum_{j=1}^{J^{(k)}} \langle \bar{g}_j^{(k)}(\mathbf{x}) \rangle, \tag{4.10}$$

where $\langle \alpha \rangle = \alpha$, if $\alpha > 0$; zero, otherwise. Constraint $\bar{g}$ is the normalized value of the constraint [34]. Thereafter, we compute the average constraint violation for a solution $\mathbf{x}$ for all scenarios, as

$\mathrm{CV}(\mathbf{x}) = \sum_{k=1}^{(k)} \mathrm{CV}^{(k)}(\mathbf{x})/K$. We also count the number of violated scenarios (NC) that a solution violates constraint. Of course, a solution is defined feasible, if $\mathrm{NC} = 0$; otherwise, the solution is declared as infeasible. We then establish a hierarchical emphasis for feasible solutions first, followed by solutions having fewer NC, and then for solutions having smaller average constraint violation. This hierarchy provides a requisite selection pressure for population members to progress from infeasible to feasible solutions in a systematic manner.

Figure 4.8 shows the step-by-step procedure for comparing two solutions ($\mathbf{x}$ and $\mathbf{y}$) and determining the better solution based on constraint scenario-domination principle described above, which involves the use of scenario-domination which involves the crowded rank procedure, and the constraint preference concept. The approach requires the user to supply the ϵ-vector, which is directly related to the trade-off among objectives desired at the final optimized solutions. Interestingly, the different ϵ-values can be chosen for each scenario, as desired in a problem.

$\mathbf{x}$ constraint scenario-dominates $\mathbf{y}$, if any of the following condition is true:

1. $\mathbf{x}$ is feasible and $\mathbf{y}$ is not.
2. $\mathbf{x}$ and $\mathbf{y}$ are feasible, and $\mathbf{x}$ scenario-dominates $\mathbf{y}$.
3. $\mathbf{x}$ and $\mathbf{y}$ are infeasible, and one of the following condition is true:
    1. NC($\mathbf{x}$) $<$ NC($\mathbf{y}$)
    2. NC($\mathbf{x}$) $==$ NC($\mathbf{y}$) AND CV($\mathbf{x}$) $<$ CV($\mathbf{y}$)

Figure 4.8: Constraint scenario-based domination principle between two solutions $\mathbf{x}$ and $\mathbf{y}$.

The evaluation procedure of a population member $\mathbf{x}$ is presented in Figure 4.9.

4.3.5 Overall Integrated MS-NSGA-II Procedure

Based on the above discussion, the work flow of the overall integrated MS-NSGA-II method is described in Figure 4.10.

Step 1: If $\mathbf{x}$ is infeasible, compute the number of violated scenarios $\mathrm{NC}(\mathbf{x})$ and average constraint violation $\mathrm{CV}(\mathbf{x})$.
Step 2: If a population member is feasible, then in each scenario j, compute all objectives in all of the scenario, $f_1^{(j)}(\mathbf{x}), f_2^{(j)}(\mathbf{x}), \ldots, f_M^{(j)}(\mathbf{x})$ for $j = 1, \ldots, K$.

Figure 4.9: Evaluation of a population member $\mathbf{x}$ for all scenarios.

4.4 Case Study

In this section, we first consider a two-variable numerical optimization problem having two scenarios which was considered as a case study in an earlier study [1]. Thereafter, we consider more involved problems including engineering design problems to assess the efficacy of our proposed procedures.

4.4.1 Numerical Problem 1

The problem introduced in the serial MSMO method [1, 7] has two objectives and two scenarios, as given below:

$$
\begin{aligned}
&\text{Minimize} \\
&f_1(x_1, x_2) \quad \begin{cases} f_1^{(1)} = (x_1 - 2)^2 + (x_2 - 1)^2, & \text{for SC 1,} \\ f_1^{(2)} = (x_1 - 1)^2 + (x_2 + 1)^2, & \text{for SC 2,} \end{cases} \\
&f_2(x_1, x_2) \quad \begin{cases} f_2^{(1)} = x_1^2 + (x_2 - 3)^2, & \text{for SC 1,} \\ f_2^{(2)} = (x_1 + 1)^2 + (x_2 - 1)^2, & \text{for SC 2,} \end{cases} \\
&\text{Subject to} \quad x_1^2 - x_2 \leq 0, \\
&\qquad\qquad\quad x_1 + x_2 \leq 2, \\
&\qquad\qquad\quad x_1 \leq 1.
\end{aligned}
\tag{4.11}
$$

Step 1: Apply reference point determination procedure (Figure 4.4) to find fronts from all individual scenarios and also intermediate reference points.

Step 2: Create an initial population (P_0) using all individual solutions from all different scenarios. Set $t = 0$.

Step 3: Repeat following steps, until a termination criterion is met.

- 3.1: Apply non-dominated sorting using constraint scenario-based domination principle discussed in Figure 4.8 and crowded rank procedure (outlined in Figure 4.7) to identify non-dominated rank and crowded rank of each population member of P_t.
- 3.2: Perform binary tournament selection operation on population P_t to create a mating pool.
- 3.3: Apply standard recombination and mutation operators to the mating pool to create an offspring population Q_t.
- 3.4: Evaluate every member of Q_t using steps outlined in Figure 4.9.
- 3.4: Merge P_t and Q_t together to create a combined population R_t. Then, apply Step 3.1 to identify constraint non-dominated and crowded ranks of each member of R_t using steps in Figures 4.8 and 4.7, respectively.
- 3.5: Perform NSGA-II's hierarchical acceptance of top non-dominated front members followed by top crowded rank solutions from the last accepted non-dominated rank to create next generation population P_{t+1}. Set $t \leftarrow t + 1$.

Step 4: Declare all solutions of the first constraint non-dominated rank as optimized solutions.

Figure 4.10: Overall integrated MS-NSGA-II method.

We denote the feasible space as $\mathcal{Y}$ here. As discussed before, the existing scenario-wise method requires to solve the following two-scenario, two-variable, two-objective optimization problems:

$$\begin{cases} \text{Minimize} & \{f_1^{(1)}(x_1, x_2), f_2^{(1)}(x_1, x_2)\}, \\ \text{Subject to} & \mathbf{x} \in \mathcal{Y}, \end{cases} \tag{4.12}$$

$$\begin{cases} \text{Minimize} & \{f_1^{(2)}(x_1, x_2), f_2^{(2)}(x_1, x_2)\}, \\ \text{Subject to} & \mathbf{x} \in \mathcal{Y}, \end{cases} \tag{4.13}$$

In all simulations here, we use NSGA-II (for aggregated methods) and MS-NSGA-II (for integrated method) with a population size of 100, the SBX recombination operator [35] with $p_c = 0.8$

and index $\eta_c = 10$, and the polynomial mutation operator [26] with $p_m = 1/n$ (n is number of variables) and index $\eta_m = 20$. The algorithms are run for a maximum of 5,000 generations to have a better understanding of the true optimized non-dominated front of the problem. As an illustration, we have used $\epsilon = 0.2$ for all normalized objectives and for all scenarios.

Figure 4.11 shows five different efficient solution sets. The blue squares and green circles are the efficient solutions for independent optimization of Scenario 1 and Scenario 2, respectively. The rest of the three solution sets are obtained from three proposed methods – average aggregate method, worst-case aggregate method, and the integrated MS-NSGA-II method. All these solutions are shown on the Scenario 1 objective space. The reference points used in the integrated MS-NSGA-II method is also shown in red diamonds in respective figures. In the Scenario 1 objective space, since the blue points are optimized for Scenario 1, it is not surprising that they are best non-dominated points and green circles, which are optimized for Scenario 2, appear to be dominated by the Scenario 1 points.

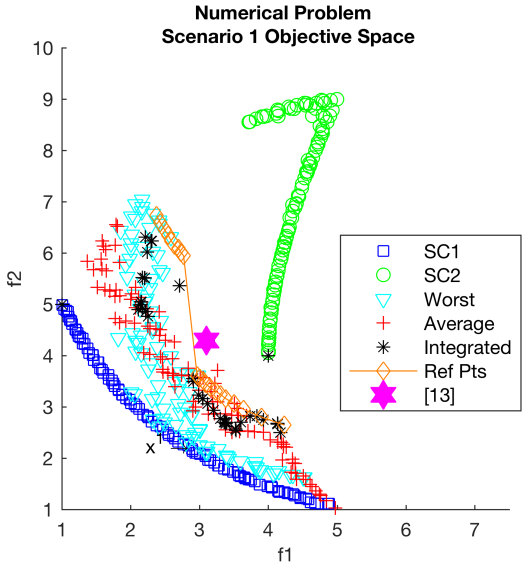

Figure 4.11: Optimized solutions for different multi-objective optimization algorithms plotted on Scenario 1 objective space for the numerical optimization problem.

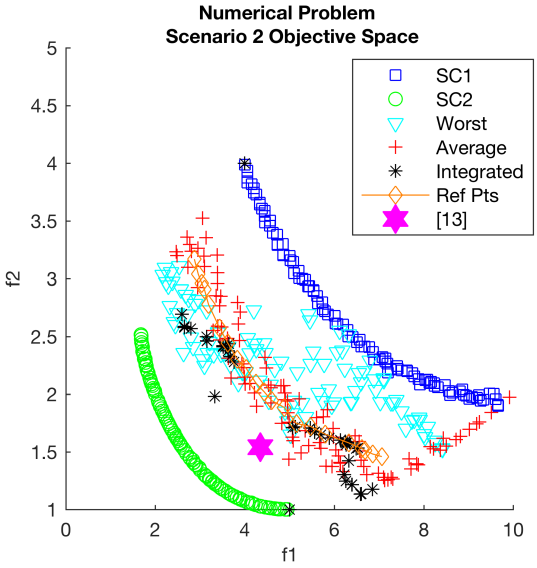

Figure 4.12: Optimized solutions for different multi-objective optimization algorithms plotted on Scenario 2 objective space for the numerical optimization problem.

Similarly, Figure 4.12 shows the efficient solutions of five optimization algorithms on Scenario 2 objective space. In this figure, green circles (trade-off solutions obtained for Scenario 2) dominate the blue squares (trade-off solutions obtained for Scenario 1). These two efficient fronts are exactly the same as those presented in the original existing study [8, 7, 1].

The figures also show the suggested final solution from the earlier study (serial MSMO) [8], which suggested a purely geometric (and fixed) approach for choosing the final preferred solution as a part of the overall multi-scenario, multi-objective optimization procedure. In [1], the decision-maker first needs to choose a single preferred solution ($\mathbf{x}^1$) from the trade-off front of the first scenario. Then, take the $\mathbf{x}^1$ as the baseline and solve the optimization problem for Scenario 2 using two additional constraints: $f_1^{(1)}(\mathbf{x}) \leq f_1^{(1)}(\mathbf{x}^1) + \epsilon_1, f_2^{(1)}(\mathbf{x}) \leq f_2^{(1)}(\mathbf{x}^1) + \epsilon_2$, where ϵ_1 and ϵ_2 are the user-supplied performance tolerances. These tolerances are decided and updated based on a trade-off and a sensitivity analysis at the every intermediate solution with respect to its performance in different scenarios. Once these performance tolerances are updated, a new and revised multi-objective optimization run is executed. This process is continued until an acceptable solution is found. Figures 4.11 and 4.12 mark the final solution obtained by the above procedure and by using performance tolerances used in the original study. First, it is interesting to note that if a different solution ($\mathbf{x}^1$) (than we have chosen here) was chosen from the trade-off set of the Scenario 1 run, the final solution would be different. The final solution of the process may also depend on the specific sequence of scenarios considered in the optimization process. These early dependencies of preferred solutions to later iterations leading to the final solution makes the overall original approach difficult to be used in practice.

On the contrary, In this dissertation, we attempt to keep the EMO principle of first finding a set of non-dominated solutions and then make a decision for a preferred solution. The average and worst aggregate solutions in both scenario spaces appear to be not all non-dominated to each

other, but if they are redrawn in their respective average and worst-scenario objective spaces, they will appear to be non-dominated to each other. As an example, Figure 4.13 shows the plot for all algorithms on worst-case aggregated space. It can be seen that now solutions obtained by the worst-case aggregate method are best.

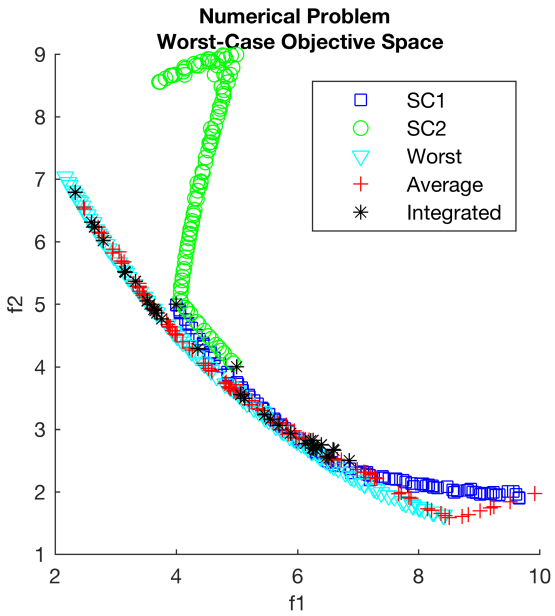

Figure 4.13: Optimized solutions for different multi-objective optimization algorithms plotted on the worst-case aggregated objective space for the numerical optimization problem.

Solutions marked using black star points in all three figures are obtained using the proposed integrated MS-NSGA-II method. They all do not appear to be non-dominated due to their plotting in a different scenario space than where these points were optimized for. However, the spread and diversity of these solutions are clear. They allow the decision-makers with a wide range of possible solutions before they choose a single preferred solution. Moreover, since a pressure is introduced for them to lie in between extreme Pareto-optimal sets for individual scenarios, they always appear as compromise solutions of all scenarios. Figures 4.14 and 4.15 shows that that solutions obtained by the integrated method are all epsilon-non-dominated.

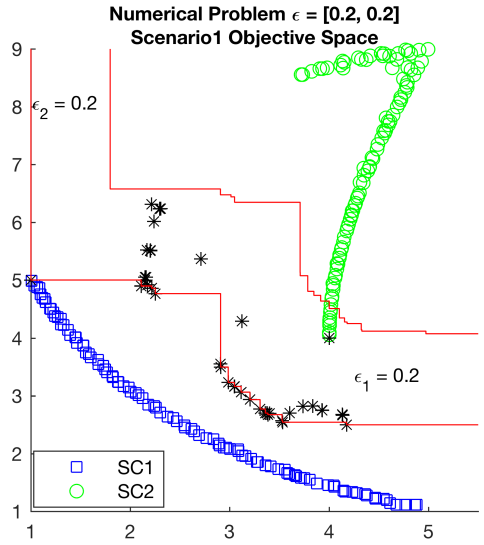

Figure 4.14: Optimized solutions obtained by the integrated method shown on Scenario 1 space for the numerical optimization problem.

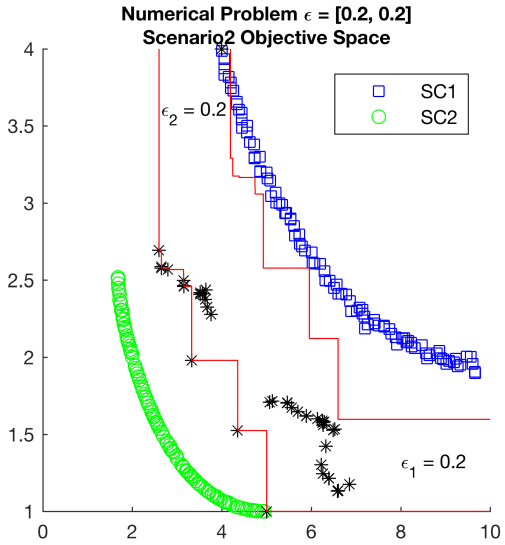

Figure 4.15: Optimized solutions obtained by the integrated method shown on Scenario 2 space for the numerical optimization problem.

The ϵ-non-dominated region corresponding to the obtained solutions are marked. Since all solutions lie within the two lines, they are all ϵ-non-dominated to each other. The ϵ-values are applied on the normalized objective values.

It is difficult to compare three of our proposed methods with the earlier serial MSMO approach [8], as our methods find a widely distributed set of compromise solutions and the serial MSMO approach finds a single preferred solution based on a number of preferences for tolerances and intermediate solutions. In this problem, the chosen preferred solution based on earlier study appears to be dominated in Scenario 1 space, whereas it dominates a few of our solutions in Scenario 2 space. In this sense, this solution may act as a potential compromise solution to the two-scenario problem, but the lack of a diverse set of solutions at the end of the optimization procedure may not provide decision-makers with other trade-off solutions to make a more informed decision-making task.

4.4.2 Numerical Problem 2: Modified ZDT1 Problem

We now introduce another numerical problem by modifying a standard single-scenario ZDT1 problem [36] into a two-scenario, two-objective optimization test problem:

$$\begin{aligned}
&\text{Minimize} \\
&f_1(x_1) \quad = x_1, f_2(\mathbf{x}) = g(1 - \sqrt{(\frac{f_1(x_1)}{g(\mathbf{x})})}) \\
&g(\mathbf{x}) \quad = \begin{cases} 0.6 + 11 \sum_{i=2}^{n} x_i/(n-1), & \text{for SC 1,} \\ 1 + 9 \sum_{i=2}^{n} (x_i - 0.5)^2/(n-1), & \text{for SC 2,} \end{cases} \\
&\text{subject to} \quad 0 \leq x_i \leq 1.
\end{aligned} \tag{4.14}$$

The first objective is identical to the original ZDT1 problem, but the second objective is varied according two scenarios. Since the $g()$ function is different for two scenarios, the corresponding Pareto-optimal solutions for each scenario are also different from each other. For the first scenario, $x_i = 0$ for $i \geq 2$ (and $x_1 \in [0, 1]$) are Pareto-optimal solutions as in the original ZDT1 problem, but for the second scenario, $x_i = 0.5$ for $i \geq 2$ (and $x_1 \in [0, 1]$) are Pareto-optimal. Since two scenarios are expected to produce two different Pareto-optimal fronts, it is of interest to investigate what solutions will correspond to the solution to the above bi-scenario, bi-objective problem. Since the above problem has variable separability, it may stay as a straightforward problem for any multi-scenario, multi-objective optimization algorithm.

An identical NSGA-II parameter values as those used in the previous problem are used here. We make a comparative evaluation of our MS-NSGA-II method with other proposed method.

4.4.2.1 Comparison in Average Aggregation Objective Space

Figure 4.16 shows obtained solutions of five optimizations (average-case, worst-case, integrated method, Scenario 1 case, and Scenario 2 case) in the average aggregated objective space. The

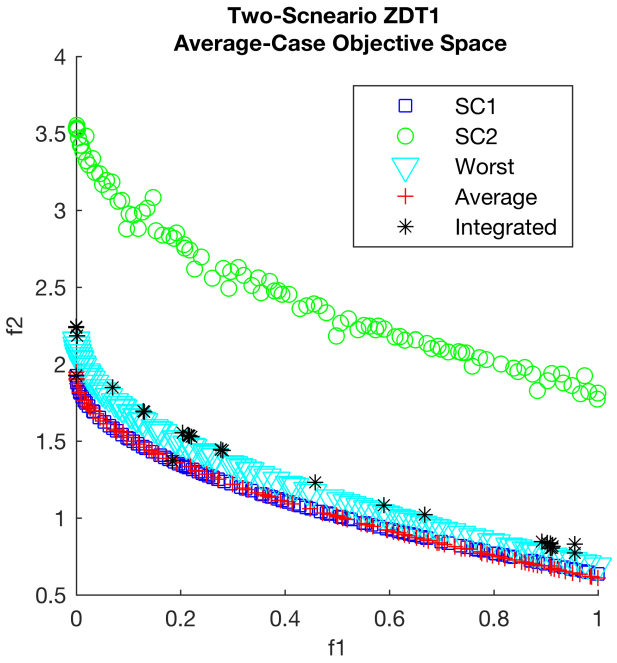

Figure 4.16: Optimized solutions for different multi-objective optimization algorithms plotted in average-case aggregated objective space for ZDT1.

non-dominated front with an average aggregate of two scenarios is marked as red '+' points. The cyan ∇-points represent solutions obtained using the worst-case aggregated method. The same problem is solved independently for each scenario, results of which are also shown as SC1 and SC2, respectively. The black star points are obtained using the integrated method.

Several observations can be made from this figure. It is important to note that all the results are plotted in the average aggregate objective space for which the x-axis values are calculated as $0.5f_1^{(1)}(\mathbf{x}) + 0.5f_1^{(2)}(\mathbf{x})$ and y-axis values are calculated as $0.5f_2^{(1)}(\mathbf{x}) + 0.5f_2^{(2)}(\mathbf{x})$. Naturally, the average aggregate results dominate (or are non-dominated to) other four optimization solutions in this specific objective space. Since the individual Scenario 2 solutions appear as worst in this

objective space, it implies that the first scenario makes the objective functions more predominant than the second scenario. This situation represents a real practical possibility: when multiple scenarios are considered for a design, one specific scenario may appear to be predominant compared to others. Thus, any average or worst-case scenario optimization will result in a solution set that is almost identical to the predominant scenario. It can be clearly seen from the figure that average-aggregated, worst-aggregated and Scenario 1 fronts are close to each other for this problem. On the other hand, our integrated method is able to find much better compromise solutions and with a wide spread on the entire x_1 range.

4.4.2.2 Comparison in Worst-case Aggregation Objective Space

Next, in Figure 4.17, we re-plot all five optimized efficient fronts in the worst-case objective space (x-axis is computed using $\max(f_1^{(1)}(\mathbf{x}), f_1^{(2)}(\mathbf{x}))$ and y-axis is computed using $\max(f_2^{(1)}(\mathbf{x}), f_2^{(2)}(\mathbf{x}))$). It is clear from the figure that the worst-case solutions, which were dominated by solutions of a few algorithms, now appear to dominate solutions from all other algorithms in the worst-aggregated objective space. The Scenario 2 solutions still appear the worst in this objective space, confirming that it is a weaker scenario compared to Scenario 1. However, it is interesting that despite Scenario 1 being more dominant, all three proposed methods (average-case aggregated, worst-case aggregated, and integrated methods) find a better set of solutions than individual scenario solutions in the worst-case objective space. This means that decision-makers have better compromise and non-dominated solutions with any of the three proposed method than individual scenario solutions, if optimization and trade-off of solutions in the worst-case objectives are their goals.

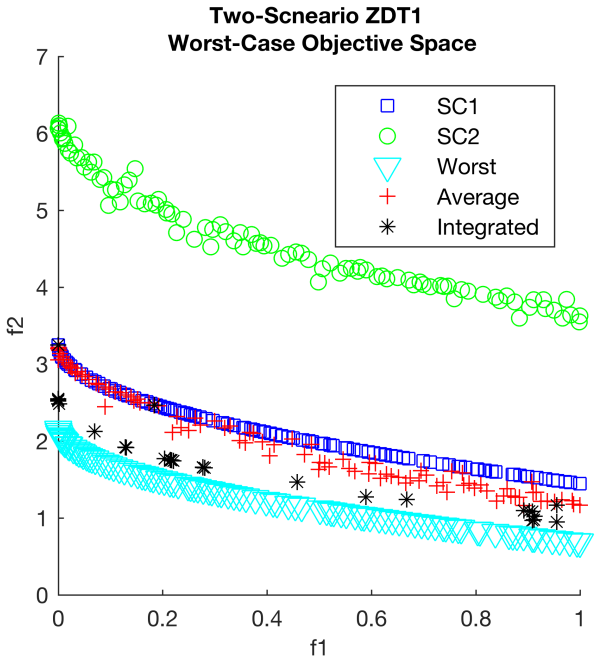

Figure 4.17: Optimized solutions for different multi-objective optimization runs plotted in worst-case aggregated objective space for ZDT1.

4.4.2.3 Comparison in Scenario 1 and Scenario 2 Objective Space

Figures 4.18 and 4.19 show all five trade-off solution sets and the respective reference points used for the integrated method in Scenario 1 ($f_1^{(1)}(\mathbf{x})$ versus $f_2^{(1)}(\mathbf{x})$) and Scenario 2 ($f_1^{(2)}(\mathbf{x})$ versus $f_2^{(2)}(\mathbf{x})$) objective spaces, respectively. Solutions are seen in a completely different way in each of these two spaces. While Scenario 2 solutions appear worst in the previous two spaces, in Figure 4.19, drawn with Scenario 2 objective values of each solution, they appear to be the best. It is interesting that in Scenario 1 space, Scenario 2 solutions are worst and vice versa. but in each space, our three proposed methods appear as compromise solutions spanning the entire range of x_1S values. In each space, solutions with the integrated method appear to be well compromised for both scenarios, despite Scenario 1 being more dominant than Scenario 2 in this problem.

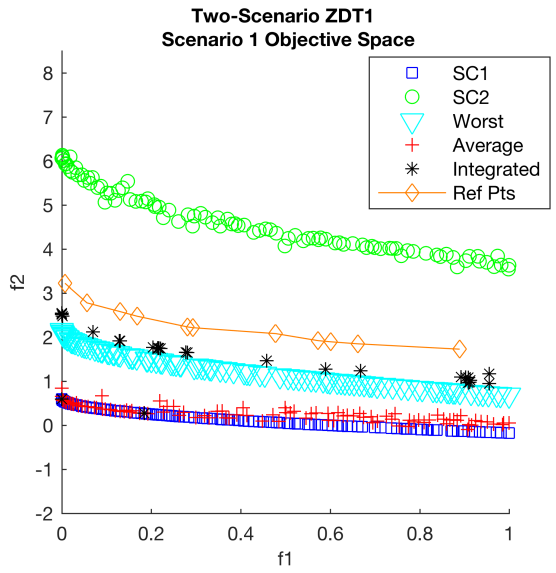

Figure 4.18: Optimized solutions for different multi-objective optimization runs plotted in Scenario 1 objective space for ZDT1.

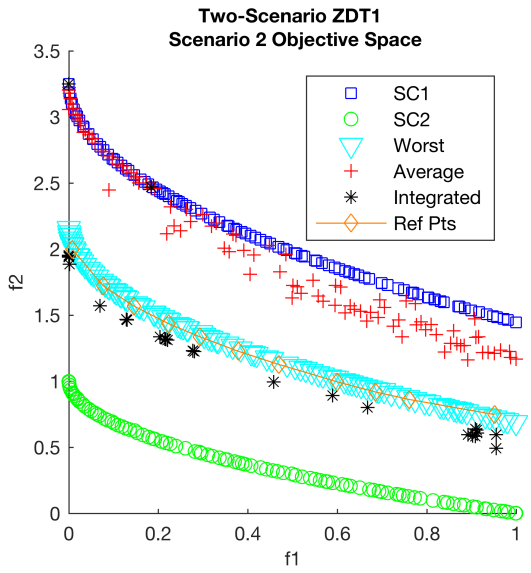

Figure 4.19: Optimized solutions for different multi-objective optimization runs plotted in Scenario 2 objective space for ZDT1.

Next, we present results of our proposed methods to three engineering design problems.

4.4.3 Welded Beam Design Problem

We consider a cantilever-type welded beam design problem with three variables (width x_1 and height x_2 of the rectangular cross-sectional area, and length x_3 of the beam, in inches) and two objectives – minimization of weight (f_1), and minimization of stored strain energy (f_2), under two different loading conditions. Figure 4.20 shows a sketch of the welded beam. Scenario 1 has an end

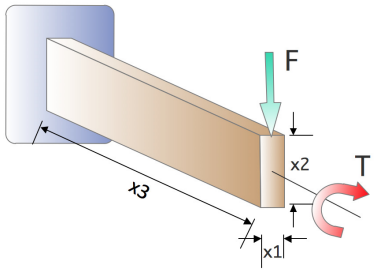

Figure 4.20: A cantilever welded beam design problem is subjected to two scenarios.

load (F) and Scenario 2 uses a torque load (T), as shown in the figure. The end load will develop a

maximum bending stress (σ) at the root of the beam and the torque load will develop a shear stress (τ) at the root of the beam. The complete optimization problem is formulated (Equation 4.15) and is given below:

$$\begin{aligned}
\text{Min. } f_1^{(1,2)}(\mathbf{x}) &= \rho x_1 x_2 x_3, \\
\text{Min. } f_2^{(1)}(\mathbf{x}) &= \frac{2F^2 x_3^3}{E x_1 x_2^3}, \\
\text{Min. } f_2^{(2)}(\mathbf{x}) &= \frac{T^2 x_3}{2c_2 G \max(x_1,x_2)\min(x_1,x_2)^3}, \\
\text{subject to} \quad & g^{(1)}(\mathbf{x}) = \frac{6Fx_3}{x_1 x_2^2} - \sigma \leq 0, \\
& g^{(2)}(\mathbf{x}) = \frac{c_1 T}{\max(x_1,x_2)\min(x_1,x_2)^2} - \tau \leq 0, \\
& 1 \leq (x_1, x_2) \leq 10, \quad 15 \leq x_3 \leq 25,
\end{aligned} \tag{4.15}$$

where F=6,000 lb, T=25,000 lb-in, ρ = 0.248 lb/in^3, E = 29.0(10^6) psi, G =11.5(10^6) psi, $\sigma = 30,000$ psi, and $\tau = 13,600$ psi. Parameters c_1 and c_2 depend on x_1/x_2 ratio and is taken from [37]. NSGA-II and MS-NSGA-II with identical parameter setting as in the previous case studies are used to optimize two-objective problems. An $\epsilon = 0.1$ is used for all normalized objectives and scenarios.

4.4.3.1 Aggregated and Integrated Methods

Figures 4.21 and 4.22 show five non-dominated fronts in two individual scenario spaces. For clarity, the plot is made in logarithmic scales. The worst and average fronts lie within Scenario 1 and Scenario 2 non-dominated fronts in these two figures. The fact that average-case, worst-case, and integrated methods' efficient fronts lie in the middle of the two individual Scenario-based efficient fronts reiterates the ability of three methods to find better compromise solutions providing importance to both scenarios. The integrated method finds solutions close to the obtained reference

points (which are in the middle of the scenario-wise solutions).

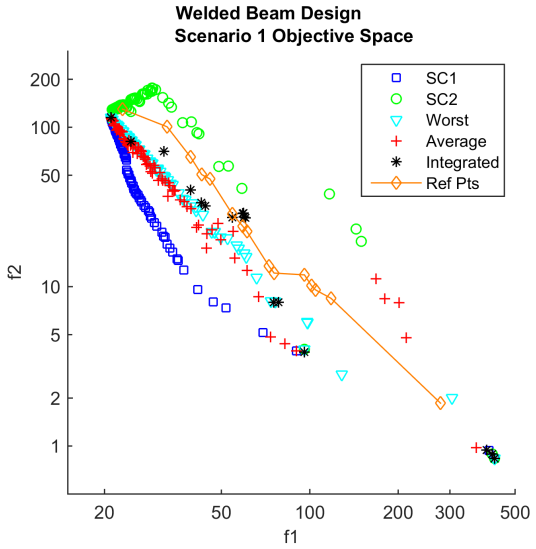

Figure 4.21: Optimized solutions for multi-objective optimization with Scenario 1 objective space for the welded beam design problem.

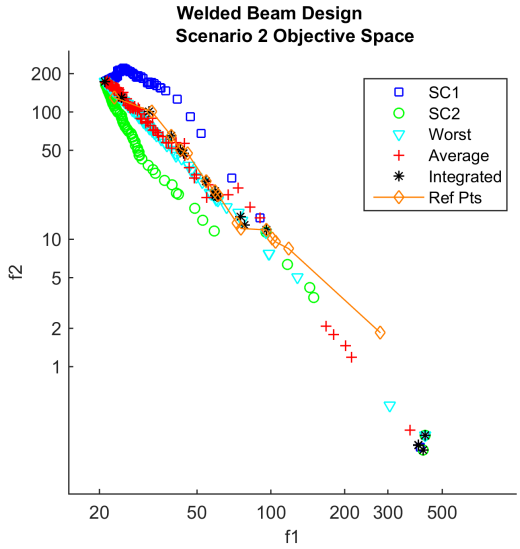

Figure 4.22: Optimized solutions for multi-objective optimization with Scenario 2 objective space for the welded beam design problem.

Since the aggregated and integrated methods find solutions almost at the middle of two individual scenarios and their role exchanges in both scenario-wise spaces, it implies that the two scenarios considered here are not overly predominant to each other. Although loading conditions are different, they have a similar effect on the obtained optimized solutions.

4.4.3.2 Investigation of Obtained Solutions

Interestingly, of the three variables in this problem, all optimized solutions have one common property: the length of the beam (x_3) in all solutions is found to be almost invariant and is close to its lower bound of 15 in. However, optimized solutions obtained by different optimization algorithms find different values for x_1-x_2 combinations, as shown in Figure 4.23.

Scenario 1 optimized solutions correspond to beams with relatively higher heights (x_2) than Scenario 2 optimized solutions. For most Scenario 1 solutions, the x_2 values are set as its upper bound of 10 in, whereas for x_2 values for Scenario 2 trade-off solutions vary from 2 to 10 in. For

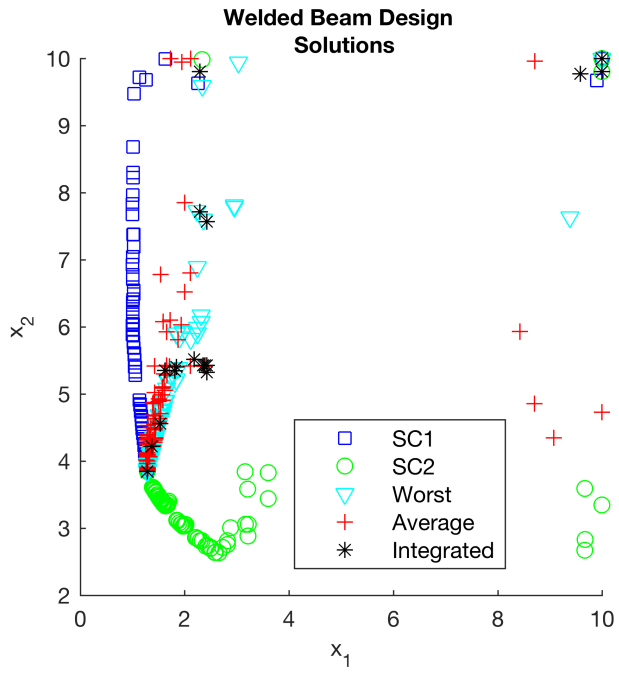

Figure 4.23: Different x_1-x_2 combinations for different multi-objective optimization runs for the cantilever design problem.

negotiating a vertical end load, a longer cross-section around the neutral axis produces a higher moment. On the other hand, to negotiate a torque load, it is better to have a more rounded or square cross-section. It can be seen from the figure that Scenario 2 solutions have almost equal x_1 and x_2 values, thereby confirming our justification of a square cross-section above. The compromised solutions obtained using our three methods produce x_1-x_2 combinations that are in between the two scenarios, revealing the pattern of compromise needed for each case. All our compromise solutions satisfy all constraints, so they are all feasible to constraints of all scenarios, but none of them optimizes any of the two individual scenarios. Individual optimal solutions of one scenario are not efficient in another scenario, but our compromise solutions make a good balance of optimizing both scenarios. While the average and worst aggregation concepts are clear, but they may appear to be biased towards one of the scenarios, if it turns out to be overly dominant, as we have witnessed in the modified ZDT1 problem before. Our integrated method is not entirely intuitive, but attempts to

focus right in the middle (meaning an equal balance of scenarios) of extreme individual trade-off solutions in all scenario spaces and come up with a wide-spread solutions. They are all non-dominated based on our newly defined scenario-space domination principle, but when they are plotted in individual scenario spaces, they may not all appear as non-dominated to each other.

To illustrate how a decision-making task may be performed in a multi-scenario, multi-objective optimization problem, we use an utility function approach. The utility function is defined as a weighted sum of all objectives, by user-defined weights: $U_{\mathbf{W}}(\mathbf{x}) = \sum_{i=1}^{M} w_i f_i(\mathbf{x})$. Since objective functions are different for different scenarios, we identify a solution $\mathbf{x}^p$ from the obtained solution set $\mathcal{S}^*$ of the integrated method that minimizes the sum of utility function of all scenarios:

$$\mathbf{x}^p = \text{argmin}_{\mathbf{x} \in \mathcal{S}^*} \left(\sum_{k=1}^{K} \sum_{i=1}^{M} w_i f_i^{(k)}(\mathbf{x}) \right). \tag{4.16}$$

In the table, we have used three weight vectors: $\mathbf{w} = (1, 0)^T$, $(0.75, 0.25)^T$, $(0.5, 0.5)^T$, $(0.25, 0.75)^T$, and $(0, 1)^T$. The first utility function makes maximum importance to f_1, the third one makes equal importance to both objectives, and the fifth one makes maximum importance to f_2. Table 4.2 presents the design variable vectors and respective objective values of our proposed integrated and weighted methods.

It is clear that all three of our proposed methods have found almost identical extreme solutions, as also matched by individual scenario-wise optimization runs. The utility function based decision-making approach clearly shows the compromises obtained between the scenarios, which are different from those obtained by average and worst-case aggregated methods.

Table 4.2: Optimized design variables for selected solutions of welded beam design problem.

Solution Set	Select Criteria	x_1	x_2	x_3	$f_1^{(1)}$	$f_2^{(1)}$	$f_1^{(2)}$	$f_2^{(2)}$
SC1	$\min(f_1^{(1)})$	1.282	3.847	15.000	21.010	114.831	21.010	172.761
Average	$\min(f_1^{(1)})$	1.282	3.847	15.000	21.010	114.818	21.010	172.766
Worst	$\min(f_1^{(1)})$	1.282	3.847	15.000	21.010	114.830	21.010	172.761
Integrated	$\min(f_1^{(1)})$	1.282	3.847	15.000	21.010	114.831	21.010	172.761
SC1	$\min(f_2^{(1)})$	10.000	10.000	15.000	426.000	0.838	426.000	0.289
Average	$\min(f_2^{(1)})$	10.000	10.000	15.000	426.000	0.838	426.000	0.289
Worst	$\min(f_2^{(1)})$	10.000	10.000	15.000	426.000	0.838	426.000	0.289
Integrated	$\min(f_2^{(1)})$	10.000	10.000	15.000	426.000	0.838	426.000	0.289
SC2	$\min(f_1^{(2)})$	1.282	3.847	15.000	21.010	114.831	21.010	172.761
Average	$\min(f_1^{(2)})$	1.282	3.847	15.000	21.010	114.818	21.010	172.765
Worst	$\min(f_1^{(2)})$	1.282	3.847	15.000	21.010	114.830	21.010	172.761
Integrated	$\min(f_1^{(2)})$	1.282	3.847	15.000	21.010	114.831	21.010	172.761
SC2	$\min(f_2^{(2)})$	10.000	9.804	15.000	417.635	0.889	417.635	0.221
Average	$\min(f_2^{(2)})$	10.000	10.000	15.000	426.000	0.838	426.000	0.289
Worst	$\min(f_2^{(2)})$	10.000	10.000	15.000	426.000	0.838	426.000	0.289
Integrated	$\min(f_2^{(2)})$	10.000	9.804	15.000	417.635	0.889	417.635	0.221
Average	$\min(U_{(1.00,0.00)})$	1.282	3.847	15.000	21.010	114.818	21.010	172.765
Worst	$\min(U_{(1.00,0.00)})$	1.282	3.847	15.000	21.010	114.830	21.010	172.761
Integrated	$\min(U_{(1.00,0.00)})$	1.282	3.847	15.000	21.010	114.831	21.010	172.761
Average	$\min(U_{(0.00,1.00)})$	10.000	10.000	15.000	426.000	0.838	426.000	0.289
Worst	$\min(U_{(0.00,1.00)})$	10.000	10.000	15.000	426.000	0.838	426.000	0.289
Integrated	$\min(U_{(0.00,1.00)})$	10.000	10.000	15.000	426.000	0.838	426.000	0.289
Average	$\min(U_{(0.75,0.25)})$	1.936	6.030	15.000	49.735	19.741	49.735	32.003
Worst	$\min(U_{(0.75,0.25)})$	1.971	5.926	15.000	49.753	20.428	49.753	30.881
Integrated	$\min(U_{(0.75,0.25)})$	1.844	5.408	15.558	44.060	32.063	44.060	47.406
Average	$\min(U_{(0.50,0.50)})$	2.000	7.849	15.000	66.873	8.665	66.873	22.307
Worst	$\min(U_{(0.50,0.50)})$	2.239	7.727	15.000	73.692	8.112	73.692	16.158
Integrated	$\min(U_{(0.50,0.50)})$	2.292	7.711	15.000	75.294	7.973	75.294	15.086
Average	$\min(U_{(0.25,0.75)})$	2.117	10.000	15.000	90.180	3.958	90.180	14.766
Worst	$\min(U_{(0.25,0.75)})$	2.954	7.813	15.000	98.320	5.9470	98.320	7.696
Integrated	$\min(U_{(0.25,0.75)})$	2.294	9.806	15.013	95.920	3.884	95.920	11.839

4.4.4 Truss Bridge Design Problem for Two and Three Scenarios

Next, we consider a truss bridge design problem with eight variables and having two objectives – minimization of weight (f_1), and minimization of maximum displacement (f_2), under multiple different loading conditions. Figure 4.20 shows a sketch of the truss bridge.

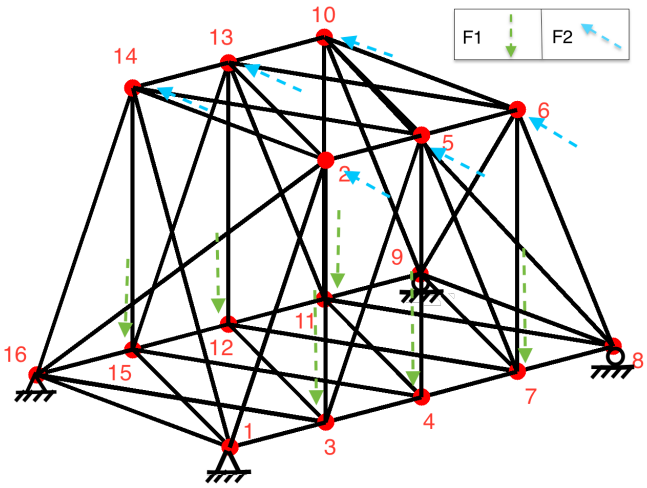

Figure 4.24: Truss bridge design problem is subjected to two scenarios.

The truss bridge problem is subject to two different static loadings – vertical loading ($F1$) simulating a heavy traffic on the bridge and horizontal loading ($F2$) simulating an extreme wind loading condition – which are considered as two different scenarios. Variables are cross-sectional areas of all 46 bars ($\mathbf{A}$) and deflections d_i^j are computed using a finite element software at each of the 12 nodes. The multi-scenario, multi-objective optimization problem is formulated as follows:

$$
\begin{aligned}
\text{Minimize} \quad & f_1^{(1,2)}(\mathbf{A}) = \rho \textstyle\sum_{i=1}^{46} L_i A_i, \\
\text{Minimize} \quad & f_2^{(j)}(\mathbf{A}) = \max(d_1^{(j)}, d_2^{(j)}, \ldots d_{12}^{(j)}),\ j = 1, 2, \\
\text{subject to} \quad & \sigma_i^{(j)} \leq 36 \text{ Ksi}, \text{ for } i = 1, 2, \ldots, 46 \text{ and } j = 1, 2, \\
& d_i^{(j)} \leq 0.2 \text{ in}, \text{for } i = 1, \ldots, 12 \text{ and } j = 1, 2, \\
& 1 \leq A_i \leq 5 \text{ in}^2, \text{ for } i = 1, \ldots, 46.
\end{aligned}
\tag{4.17}
$$

We have used a vertical load of $F1 = 18$ Kips at six lower deck nodes for Scenario 1 ($j = 1$) and a horizontal load of $F2 = 18$ Kips at six upper level deck nodes for Scenario 2 ($j = 2$). Loads $F1$ is absent in Scenario 2 and vice versa.

Although the first objective function (weight) does not depend on any scenario, but due to the hard nature of constraint satisfaction of all scenarios, not all variable combinations will be feasible and hence both objectives are dependent on chosen $F1$ and $F2$ values. The function $\sigma_i^{(j)}(\mathbf{x})$ represents the tensile or compressive stress developed in the i-th bar under j-th scenario loading. The expression for deflection $d_i^{(j)}$ and stress $\sigma_i^{(j)}$ formulations under each loading scenario can be found from the engineering mechanics literature [37]. The developed stress must be smaller than the allowable strength of the chosen material, for the truss to not fail under the loading. The above formulation requires that all stress constraints must be satisfied for a solution $\mathbf{x}$ to be feasible. For the mechanical properties, we use modulus of elasticity, $E = 29,500$ Ksi and density of the material, $\rho = 0.3$ lb/in^3. Identical NSGA-II and MS-NSGA-II parameters to those in previous problems are used here.

4.4.4.1 Two-scenario Results

An $\epsilon = 0.2$ is used for all normalized objectives and scenarios. Figures 4.25 and 4.26 show five non-dominated fronts in individual scenario-wise objective spaces.

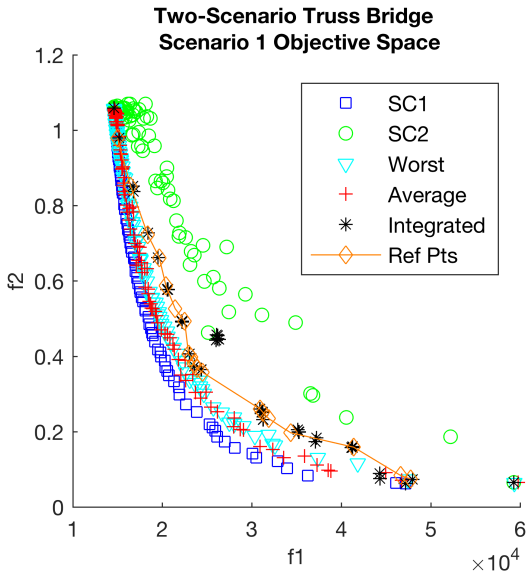

Figure 4.25: Optimized solutions in Scenario 1 objective space for the two-scenario truss bridge design problem.

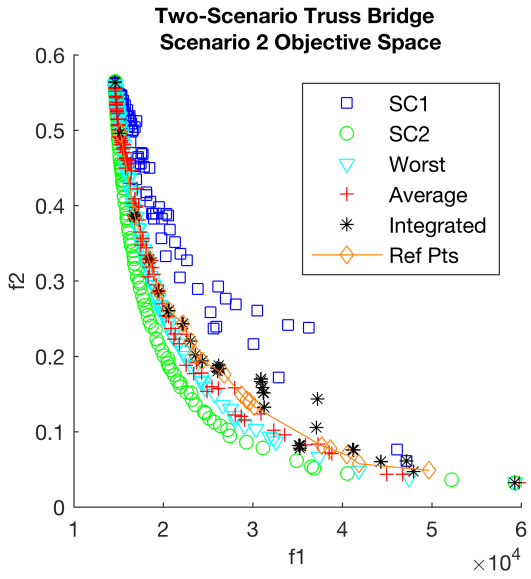

Figure 4.26: Optimized solutions in Scenario 2 objective space for the two-scenario truss bridge design problem.

Figures reveal the fact that two scenarios produce different solutions individually. Thus, when both scenarios are important meaning that in some occasion Scenario 1 is active while in some other occasions Scenario 2 will be active during the life time of the bridge, not only that an optimized truss be safe in both scenarios, it must also make a good compromise between the respective objective values under the two scenarios. If this is not achieved, then the resulting solution is designed to be biased for one of the two scenarios.

In this problem, the usual worst-case scenario solutions (which is a common practice in engineering optimal design studies), the trade-off solutions are somewhat biased towards the first scenario solutions, as evident from Figure 4.25. Average-case and worst-case aggregated solutions are closer to the Scenario 1 solutions.

On the contrary, as evident from both figures, the integrated method is able to find solutions having a better balance between the two scenarios in both scenario spaces.

4.4.4.1.1 Optimized Design Variables and Decision Making

Table 4.3 presents a few selected solutions from the solution set obtained by the integrated method. The best objective solutions of f_1 is tabulated first. This specific solution is obtained by making a compromise between two scenarios for the first objective alone. Thus, if the first objective is infinitely more important than the second objective for a particular decision-making task, the integrated method has found a single compromise solution that is best for f_1 by compromising both scenarios. Scenario-wise best-f_1 solutions are also presented in the table. It is clear that if the problem was solved for individual scenarios, different solutions would be obtained, instead of a single solution, thereby making the decision-making task difficult. A compromise solution for the best second-objective and their scenario-wise variations are also presented in the table.

The trade-off between the two objectives is clear from Table 4.3, which presents solutions from our integrated method. The extreme solutions of our integrated method are exactly identical to the individual scenario-wise extreme solutions. Three other intermediate solutions obtained using the utility function MCDM approach (equation 4.16) are also presented, showing the compromise nature of these solutions. Any other decision-making principle can be applied to the obtained solution set $\mathcal{S}^*$ to choose a single preferred solution after the MS-NSGA-II is completed. Since one solution set is the outcome of the integrated method, the decision-making task becomes identical to that used for single-scenario, multi-objective optimization problems [27, 38, 39].

Table 4.3: Optimized design variables for selected solutions obtained by the integrated method for the two-scenario truss-bridge design problem.

Select Criteria	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	$f_1^{(1)}$	$f_2^{(1)}$	$f_1^{(2)}$	$f_2^{(2)}$
$\min(f_1^{(1)})$ (SC1)	1.231	1.118	1.229	1.000	1.000	1.505	1.000	1.000	14,596	1.059	14,596	0.564
$\min(f_1^{(1)})$ (intg.)	1.231	1.118	1.229	1.000	1.000	1.505	1.000	1.000	14,596	1.059	14,596	0.564
$\min(U_{(1.00,0.00)})$ (intg.)	1.231	1.118	1.229	1.000	1.000	1.505	1.000	1.000	14,596	1.059	14,596	0.564
$\min(f_2^{(2)})$ (SC2)	5.000	5.000	5.000	5.000	5.000	5.000	1.000	1.015	59,287	0.065	59,287	0.032
$\min(f_2^{(2)})$ (intg.)	5.000	5.000	5.000	5.000	5.000	5.000	1.000	1.015	59,287	0.065	59,287	0.032
$\min(U_{(0.00,1.00)})$ (intg.)	5.000	5.000	5.000	5.000	5.000	5.000	1.000	1.015	59,287	0.065	59,287	0.032
$\min(U_{(0.75,0.25)})$ (intg.)	1.373	1.404	1.300	1.292	1.000	1.818	1.003	1.470	16,771	0.837	16,771	0.385
$\min(U_{(0.50,0.50)})$ (intg.)	1.365	1.742	3.017	1.142	1.017	3.160	1.059	1.168	23,574	0.373	23,574	0.202
$\min(U_{(0.25,0.75)})$ (intg.)	4.915	5.000	4.297	1.730	1.464	4.818	1.056	1.046	44,306	0.075	44,306	0.061

4.4.4.2 Three-scenario Results

To demonstrate the working of our proposed methodology for more than two scenarios, we now consider the same truss bridge design problem but introduce a third scenario. Scenarios 1 and 2 remain the same as before, but now we add Scenario 3 which is a combination of these two scenarios. Thus, the third scenario simulates an operating condition in which the bridge has half the vertical load as in Scenario 1 (moderate crowding) and simultaneously has half of the horizontal load considered in Scenario 2 (moderate wind load) resulting in a combined loading of $0.5 * F_1$ and $0.5 * F_2$. As discussed before, all three of our proposed multi-scenario, multi-objective optimization algorithms (aggregated and integrated approaches) extend to more than two scenarios. An $\epsilon = 0.1$ is used for all normalized objectives and for all three scenarios.

Solutions obtained after optimizing the truss bridge for all three scenarios are given in Figures 4.27, 4.28, and 4.29. Each figure shows six sets of solutions (three individual scenario optimizations and three proposed methods) in each scenario-wise objective space.

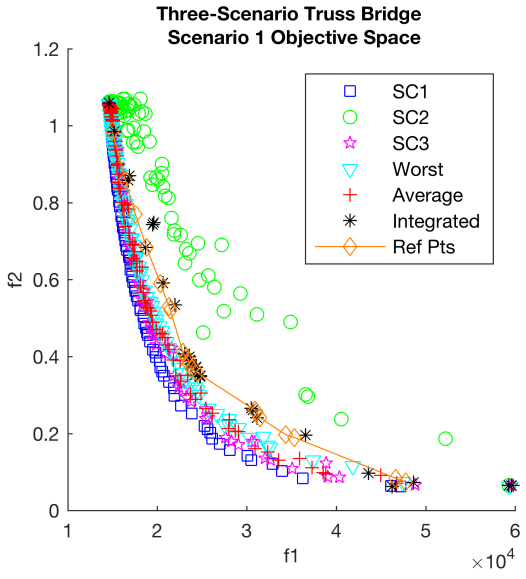

Figure 4.27: Optimized solutions in Scenario 1 objective space for the three-scenario truss bridge design problem.

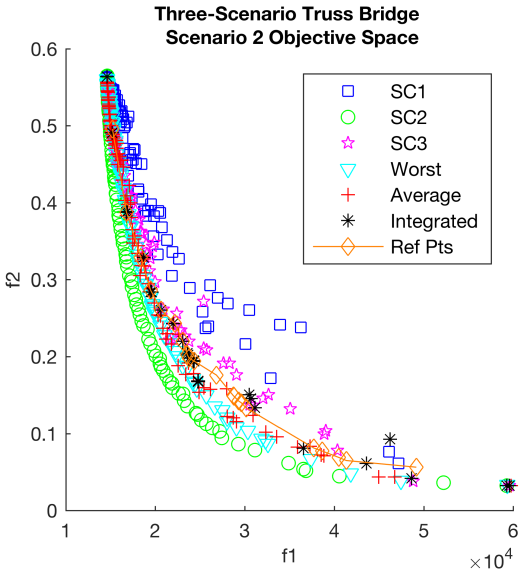

Figure 4.28: Optimized solutions in Scenario 2 objective space for the three-scenario truss bridge design problem.

In all scenarios, the worst-case and average-case solutions lie between scenario-wise best and

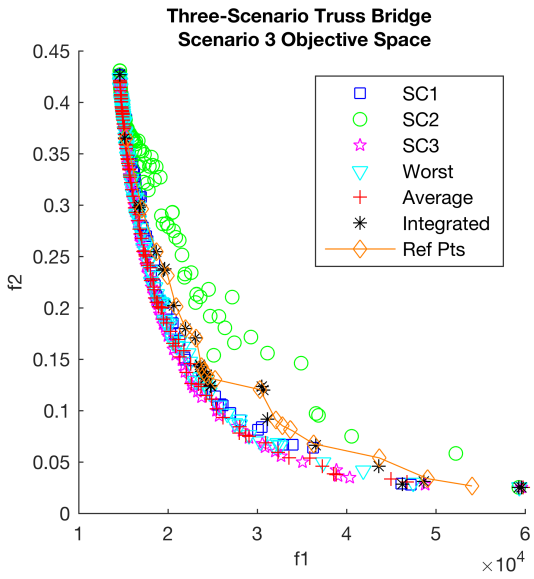

Figure 4.29: Optimized solutions in Scenario 3 objective space for the three-scenario truss bridge design problem.

worst solutions. Figure 4.29 reveals clearly that for this problem Scenario 3 is most dominant and both average and worst-case optimized solutions are almost similar to the Scenario 3 solutions. But if all three scenarios are equally likely to occur in practice, an average-case or worst-case design optimization makes its optimized solutions over-design when other two scenarios occur, thereby making the designs biased for Scenario 3. Figures 4.274.28 make it clear that better balanced designs are possible for all three scenarios. The integrated method is able to find a better compromised set of solutions balancing all three scenarios.

4.4.5 Four-bar Truss Design Problem

Finally, we choose a four-bar truss design problem, which was also studied before [1]. A sketch of the truss structure is shown in Figure 4.30. The four-bar truss problem is subject to three different static loadings applied at the nodes B and D considered as three scenarios, and for each scenario, two objectives – (i) minimization of total volume of bars and (ii) minimization of the displacement at node B – are considered. The design variables are four cross-sectional areas of these four bars:

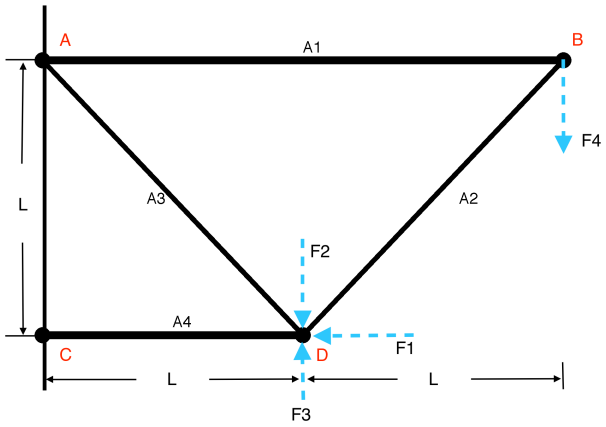

Figure 4.30: Three-scenario, four-bar truss design problem used in [1].

A_1, A_2, A_3, and A_4. The definition of three scenarios is shown below.

$$
\begin{aligned}
\text{Minimize} \quad & f_1^{(1,2,3)}(\mathbf{A}) = L(2A_1 + \sqrt{2}A_2 + \sqrt{2}A_3 + A_4), \\
\text{Minimize} \quad & f_2^{(1)}(\mathbf{A}) = \frac{FL}{E}\left(\frac{2}{A_1} + \frac{2\sqrt{2}}{A_2} + \frac{2\sqrt{2}}{A_3} + \frac{2}{A_4}\right), \\
& f_2^{(2)}(\mathbf{A}) = \frac{FL}{E}\left(\frac{2}{A_1} + \frac{2\sqrt{2}}{A_2} + \frac{4\sqrt{2}}{A_3} + \frac{2}{A_4}\right), \\
& f_2^{(3)}(\mathbf{A}) = \frac{FL}{E}\left(\frac{6\sqrt{2}}{A_3} + \frac{3}{A_4}\right), \\
\text{subject to} \quad & (F/\sigma) \leq (A_1, A_4) \leq 3(F/\sigma), \\
& \sqrt{2}(F/\sigma) \leq (A_2, A_3) \leq 3(F/\sigma).
\end{aligned}
\tag{4.18}
$$

Four forces (F_1, F_2, F_3, F_4) are given for each scenario, as follows:

$$
(F_1, F_2, F_3, F_4) = \begin{cases} (F, 0, 2F, F), & \text{for SC1 } (j = 1), \\ (0, F, 0, F), & \text{for SC2 } (j = 2), \\ (0, 3F, 0, 0), & \text{for SC3 } (j = 3). \end{cases}
$$

The length of the bar $L = 200$ cm, the force $F = 10$ kN, elasticity $E = 2 \times 10^5$ kN/cm^2 , and the stress component $\sigma = 10$ kN/cm^2 are set to be constant. Identical NSGA-II and MS-NSGA-II parameters to those in previous problems are used here. An $\epsilon = 0.2$ is used for all normalized

objectives and scenarios.

4.4.5.1 Results of Four-bar Truss Design Problem

Solutions from all six optimization runs, when plotted in individual objective space of three scenarios is given in Figures 4.31, 4.32 and 4.33, are shown. The effect of each loading scenario is clear from the figures.

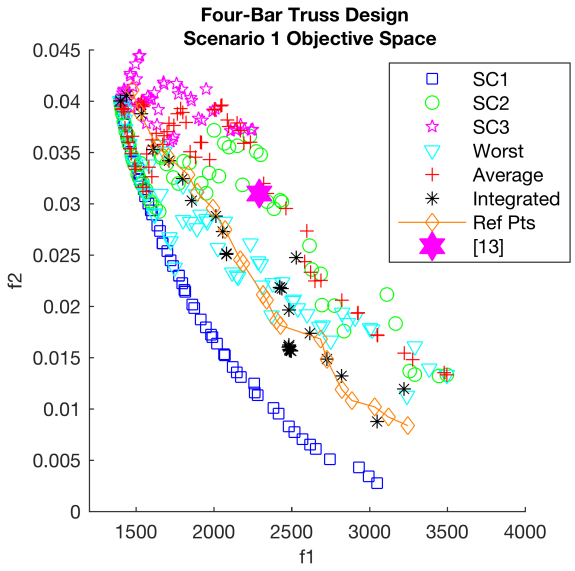

Figure 4.31: Optimized solutions plotted in Scenario 1 objective space for the four-bar truss design problem.

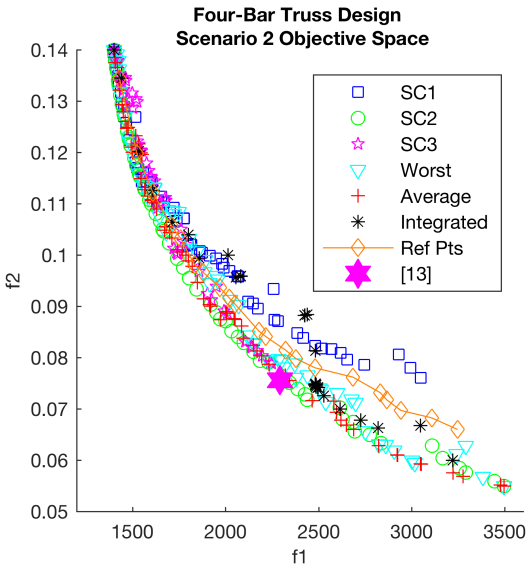

Figure 4.32: Optimized solutions plotted in Scenario 2 objective space for the four-bar truss design problem.

The figures reveal that Scenario 2 is more dominant compared to other two scenarios, as the average and worst-case optimized solutions all appear to be closer to the Scenario 2 solutions. Repeating the need for an integrated method here, an average-case or worst-case optimization of objectives will ignore less dominant scenarios and will find solutions that are similar to the most dominant scenario solutions. However, notice how the integrated method is able to make a good balance among all three scenarios.

This problem was solved by the earlier method and resulted in a single solution, marked with a star in all three figures by using a number of tolerance parameters and intermediate decision-making aids. Clearly, the particular solution is closer to Scenario 2 solution. Although a different

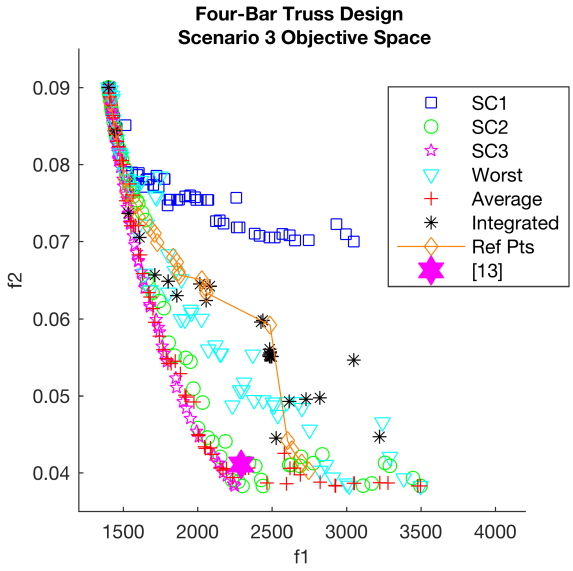

Figure 4.33: Optimized solutions plotted in Scenario 3 objective space for the four-bar truss design problem.

tolerance parameter values and decision-making aids would have found a different final solution, the outcome of such early and intermediate decisions is not known until the whole process is completed. Our integrated method enables an algorithmic approach to finding a set of well-balanced optimized solutions to provide the decision-makers a set of solutions to analyze and make a better informed decision.

4.4.5.2 Optimized Design Variables and Decision Making

Table 4.4 presents a few selected solutions from the solution set of the integrated method for the four-bar truss design problem. Minimum objective-wise solutions are shown in the table. For a comparison, scenario-wise best objective f_1 and f_2 solutions are also presented in the table. Since the the first objective is identical for all scenarios, all scenario-wise best-f_1 solutions are identical. It is clear that scenario-wise optimization produce different solutions for the second objective function, whereas our integrated method (or the aggregated methods) produce a single solution set balancing and compromising all scenarios.

Table 4.4: Optimized design variables for selected solutions from integrated method of the four-bar truss design problem.

Select Criteria	x_1	x_2	x_3	x_4	$f_1^{(1)}$	$f_2^{(1)}$	$f_1^{(2)}$	$f_2^{(2)}$	$f_1^{(3)}$	$f_2^{(3)}$
$\min(f_1^{(1)})$ (SC1)	1.000	1.414	1.414	1.000	1,400	0.040	1,400	0.140	1,400	0.090
$\min(f_1^{(1)})$ (intg.)	1.000	1.414	1.414	1.000	1,400	0.040	1,400	0.140	1,400	0.090
$\min(f_1^{(2)})$ (SC2)	1.000	1.414	1.414	1.000	1,400	0.040	1,400	0.140	1,400	0.090
$\min(f_1^{(2)})$ (intg.)	1.000	1.414	1.414	1.000	1,400	0.040	1,400	0.140	1,400	0.090
$\min(f_1^{(3)})$ (SC3)	1.000	1.414	1.414	1.000	1,400	0.040	1,400	0.140	1,400	0.090
$\min(f_1^{(3)})$ (intg.)	1.000	1.414	1.414	1.000	1,400	0.040	1,400	0.140	1,400	0.090
$\min(f_2^{(1)})$ (SC1)	3.000	3.000	1.414	3.000	3,049	0.003	3,049	0.076	3,049	0.070
$\min(f_2^{(1)})$ (intg.)	2.660	2.993	1.902	2.999	3,048	0.009	3,048	0.067	3,048	0.055
$\min(f_2^{(2)})$ (SC2)	3.000	3.000	3.000	3.000	3,497	0.013	3,497	0.055	3,497	0.038
$\min(f_2^{(2)})$ (intg.)	2.701	2.998	2.447	2.999	3,220	0.012	3,220	0.060	3,220	0.045
$\min(f_2^{(3)})$ (SC3)	1.000	1.414	3.000	3.000	2,249	0.037	2,249	0.079	2,249	0.038
$\min(f_2^{(3)})$ (intg.)	1.003	2.942	2.460	2.995	2,528	0.025	2,528	0.073	2,528	0.045
$\min(U_{(1.00,0.00)})$ (intg.)	1.000	1.414	1.414	1.000	1,400	0.040	1,400	0.140	1,400	0.090
$\min(U_{(0.00,1.00)})$ (intg.)	2.701	2.998	2.447	2.999	3,220	0.012	3,220	0.060	3,220	0.045
$\min(U_{(0.75,0.25)})$ (intg.)	1.000	1.414	1.414	1.000	1,400	0.040	1,400	0.140	1,400	0.090
$\min(U_{(0.50,0.50)})$ (intg.)	1.092	1.414	1.695	1.467	1,610	0.035	1,610	0.113	1,610	0.071
$\min(U_{(0.25,0.75)})$ (intg.)	1.694	2.999	2.154	2.955	2,726	0.050	2,726	0.068	2,726	0.050

Three utility function based solutions are selected from the solution set and are presented in the table as well. It is clear that the integrated method finds a compromise set of solutions in handling multiple scenarios.

4.4.6 Discussion: Effect of ϵ in the Integrated Method

The integrated method involves setting an ϵ-vector for setting up an accepted level of domination of one scenario solutions into another scenario space. In previous sections, we have chosen a reasonable ϵ value (identical for all objectives) to solve each problem. In this section, we perform a parametric study of ϵ on the numerical problem and the truss bridge design problems. Since we normalize each scenario using their ideal and nadir points obtained from their individually optimized trade-off solutions, a single ϵ parameter can be used for each objective and in each scenario space.

Different ϵ values in the range of $[0, 1]$ are tested for the numerical problem (Section 4.4.1) and two-scenario truss bridge design problem (Section 4.4.4). When $\epsilon = 0$, the scenario-based domination becomes exactly the same as the usual domination used in multi-objective domination. When the other extreme end of $\epsilon = 1$ is chosen, it allows all solutions having objective values within best and worst optimized values under all scenarios to be ϵ-non-dominated to each other and the selection process is then mainly relies on the scenario-based crowding distance operator.

We use the hypervolume metric to measure the convergence and diversity of an obtained set of trade-off solutions in a specified space. In each space, we find the worst objective values of obtained individual scenario-wise solutions and set as the reference point for hypervolume computation. After the solutions are obtained for a specified ϵ-value, we compute hypervolume in each individual scenario space. Hypervolumes from different runs are averaged and presented in figures. Figures 4.34 and 4.35 show the average hypervolume calculated for solution sets of dif-

ferent ϵ values for two scenarios. In both figures, when $\epsilon = 0$, the hypervolume is the lowest, as expected. However, as ϵ increases, hypervolume increases rapidly and thereafter stabilizes to an intermediate value. For the two-scenario truss bridge problem, in both scenarios, when $\epsilon = 0.02$, the hypervolume reaches its maximum value and slightly decreases to a stabilized value. The similar observation is also held for the numerical optimization problem, also for a small $\epsilon = 0.07$ value.

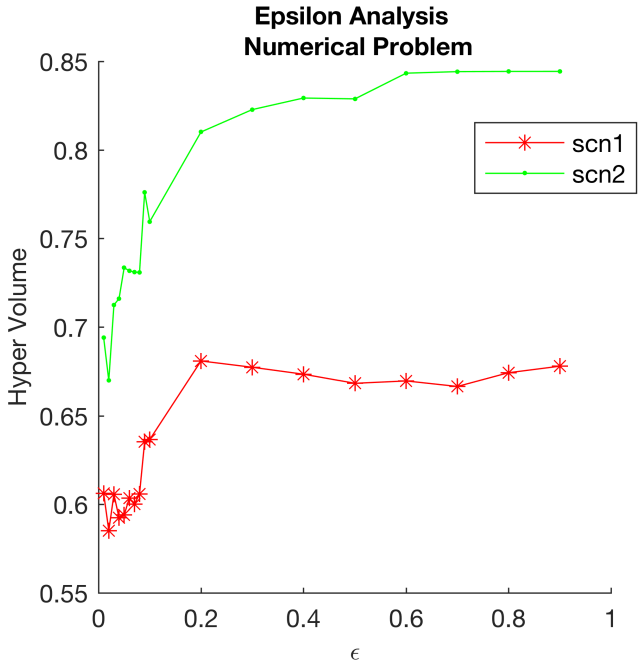

Figure 4.34: Hypervolume of optimized solutions obtained using the integrated method with different ϵ values for the numerical optimization problem.

The performance of the integrated method is shown for two different ϵ values (0, 0.02, and 0.9) in Figures 4.36, 4.37, and 4.38, respectively. In each figure, the final solutions from integrated method for a median run are shown on scenario-wise optimized solutions. For $\epsilon = 0$ run, the integrated method is not able to find a spread of solutions, as without preserving dominated solutions in different scenarios for non-dominated solutions in a specific scenario, multiple compromise solutions cannot be found. Interestingly, when a large $\epsilon = 0.9$ is used, the integrated method struggles to find a more converged set of compromise solutions, as almost any solution is

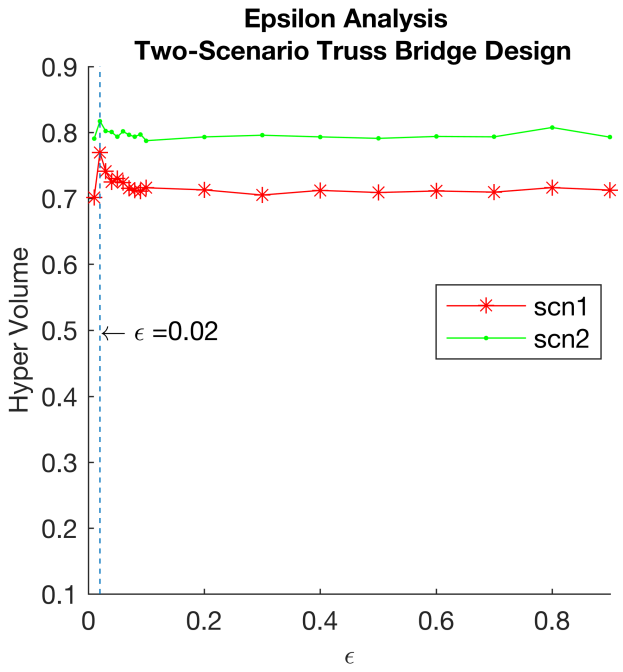

Figure 4.35: Hypervolume of optimized solutions obtained using the integrated method with different ϵ values for the truss-bridge design problem.

scenario-non-dominates to another solution and a scenario-based crowding approach cannot impose adequate selection pressure for a proper convergence. However, when a proper ϵ ($= 0.02$ here) is used, an appropriate selection pressure is established between scenario-based domination and scenario-based niching. Figure 4.34 shows a nicely convergence and well-distributed compromise solutions.

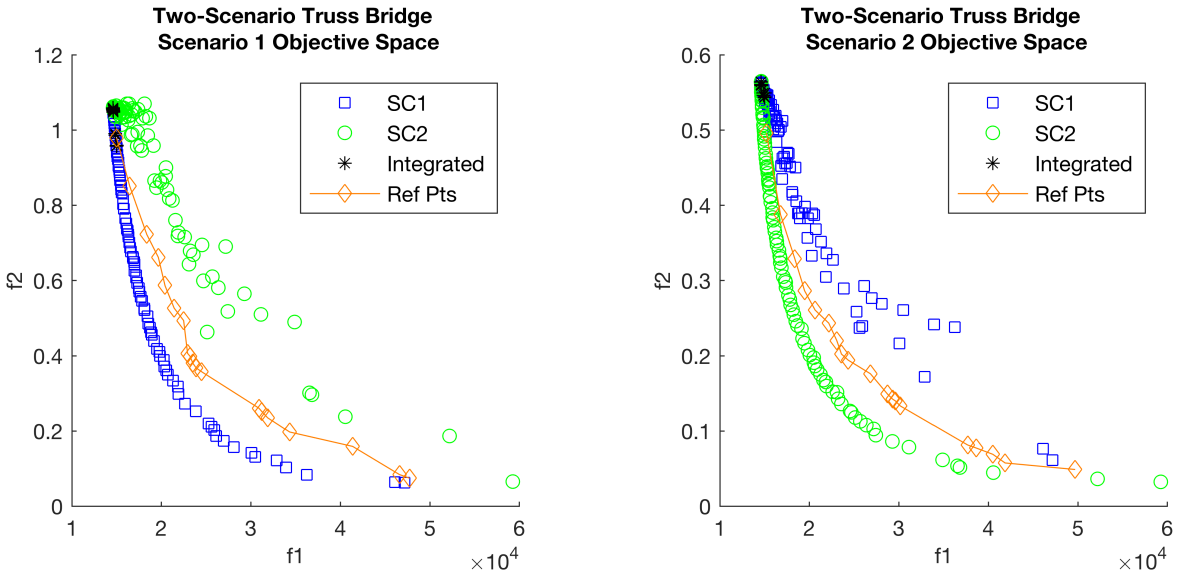

Figure 4.36: Optimized solutions for $\epsilon = 0$ in truss-bridge design problem.

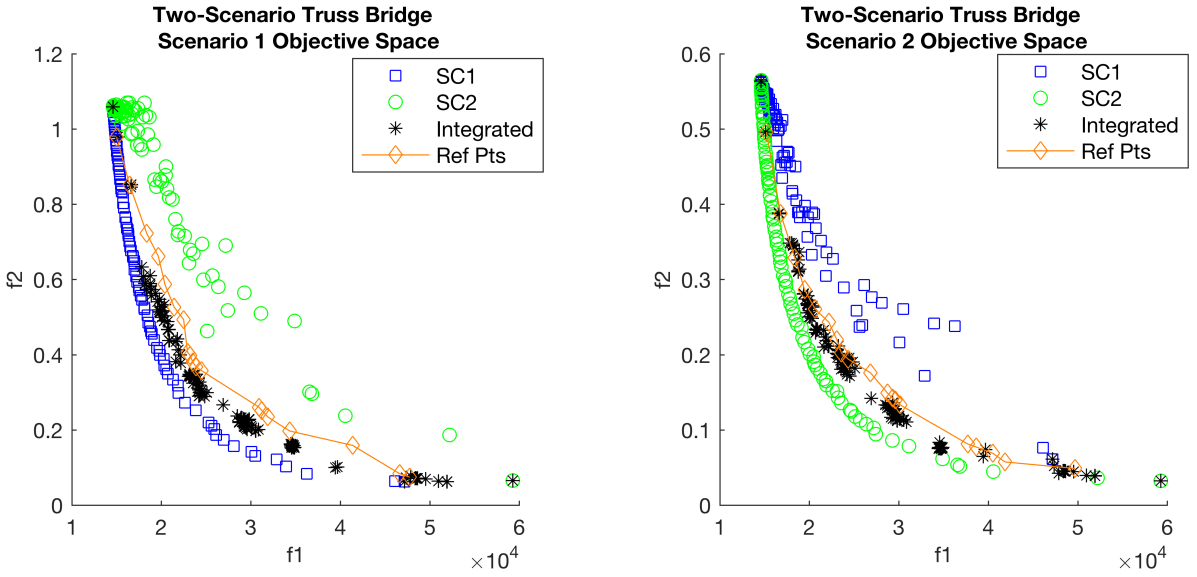

Figure 4.37: Optimized solutions for $\epsilon = 0.02$ in truss-bridge design problem.

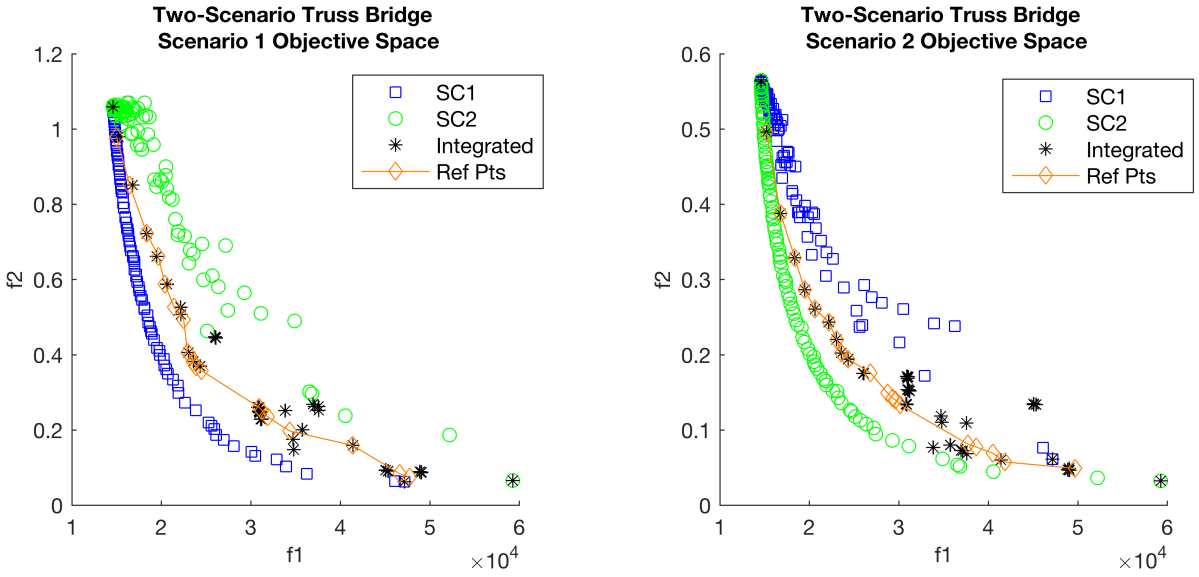

Figure 4.38: Optimized solutions for $\epsilon = 0.9$ in truss-bridge design problem.

Chapter 5

Scenario-wise Approach for Scenario-dependent MSMO Problems

5.1 General Description

For some problems, not all scenarios have the same objectives. For instance, in synthesizing fault-tolerant programs, the objective functions are calculated by program verification, that is, how the program satisfies program requirements. Program requirements for scenario in presence of malicious processes can be different from the ones in absence of the faults. For the scenario in presence of a malicious process, we can not verify the states of all processes, since the malicious process behaves unpredictable. Instead, it needs to verify the normal processes sometime with relaxed requirements. The desired solutions are required to satisfy all different requirements in every scenario. Objective-wise approach has limitation on handling scenarios with different types of objectives or with different number of objectives throughout the scenarios. Alternatively, scenario-wise approach aims to solve multiple scenarios with multiple or many different objectives in each scenario.

For scenario-wise approach, each scenario is considered as a criterion, and the original MSMO problem is converted into multi-scenario, single-objective problem. Figure 5.1 shows the structure of the specific scenario-dependent MSMO problem that can be optimized by scenario-wise

approach. The evaluation of the objective function for a solution requires computation for all K scenarios and then aggregation measure. We combine all objectives in each scenario and consider the combination as a criterion, and optimize all criteria. This approach is especially useful when there are different objectives throughout scenarios, and if the problem do not have optimal solution throughout the scenarios, it can be used to find compromised solutions that provide trade-offs among scenarios.

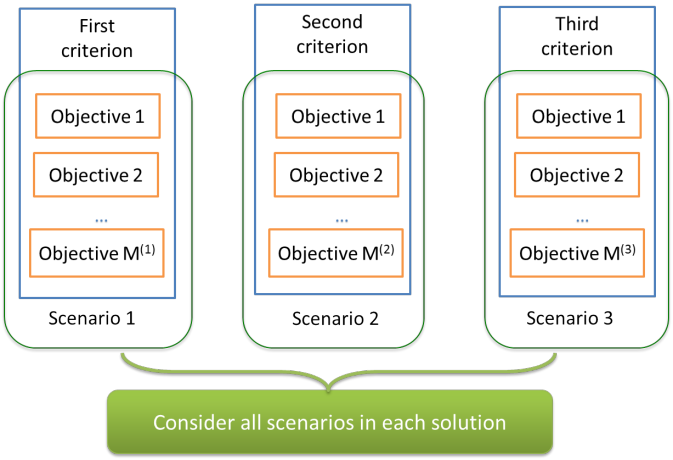

Figure 5.1: Scenario-wise Approach

In this work, the scenario-wise approach uses average aggregation method to combine the objectives within a scenario. In the following sections, we discuss the average aggregated method, followed by case studies to demonstrate effectiveness of the approach.

The general problem description for scenario-wise approach is the same with the one described in Section 3.2. As we use aggregation approach, the resulting problem description is as follows:

$$\begin{aligned} \text{Minimize} \quad & \left\{ \sqcup_{m=1}^{M^{(1)}} f_m^{(1)}(\mathbf{x}), \sqcup_{m=1}^{M^{(2)}} f_m^{(2)}(\mathbf{x}), \ldots, \sqcup_{m=1}^{M^{(K)}} f_m^{(K)}(\mathbf{x}) \right\}, \\ \text{Subject to} \quad & g_j^k(\mathbf{x}) \leq 0, \quad j = 1, 2, ..., J^k, \; k = 1, 2, \ldots, K. \end{aligned} \tag{5.1}$$

The operator $\sqcup$ denotes the aggregate operator. Note that the above formulation allows a different aggregate function. In this section, we mainly focus on the average case aggregation, but other

aggregation methods can be applied such as mean-variance, worst-case, weighted sum, etc. All objective values should be normalized before aggregation. This is especially important, otherwise, the optimization will bias toward objectives with large value.

Average-Case Aggregation An average objective value within a scenarios can be formulated as follows:

$$\sqcup_{m=1}^{M^{(k)}} f_m^{(k)}(\mathbf{x}) = \frac{1}{M^{(k)}} \sum_{m=1}^{M^{(k)}} f_m^{(k)}(\mathbf{x}). \tag{5.2}$$

The average-case scenario is logical particularly when there is not much difference in performance among all objectives within a scenario. When the variance in performance among objectives is large, a more statistically favorable aggregation would be better. Different aggregation methods can be applied depending on the properties of problems.

5.2 Case Study

In this section, we investigate a study on modeling fault-tolerant programs using scenario-wise approach. We consider the problem as scenario-dependent MSMO problem and solve it using scenario-wise approach. First, we describe the detailed synthesis method, then demonstrate the effectiveness of the scenario-wise approach using three case studies and their experimental results.

5.2.1 Modeling Distributed Fault-tolerant Programs

The modeling technique used in this study is Genetic programming (GP). GP is designed to automatically generate computer programs, or model problems. The details of GP mechanism is described in Section 2.3.5. Traditionally, most applications of GP use test-based or simulation-based evaluation on generated programs to calculate objective functions. The generated programs

are usually executed on a set of test cases, and the objective functions are typically aggregation of these calculated executions. This evaluation process is considered as terminating calculation and in most cases it is used for sequential programs that do not have non-determinism. However, this method is not suitable for distributed programs. The evaluation of distributed programs is often complicated due to a large number of concurrent executions. Moreover, different race conditions that result in several possible runs make it difficult to analyze the correctness of distributed programs. The terminating calculation cannot evaluate distributed programs accurately because it can not fully incorporate the specification entirely, and the program execution can be infinite. Hence in GP, using the traditional test-based method is not effective to evolve the distributed programs.

This problem has been solved in practice using one of the formal methods techniques - model checking. In model checking, a program is deemed to be correct only if it satisfies the program specification. It statically checks the entire program state space to verify the program. In this study, we use model checking techniques to evaluate the generated distributed program. Model checking based execution is used to identify the effect of that program on all possible permitted initial states. This symbolic execution is then used to determine whether the specification is satisfied. In the event the specification is satisfied, the desired program is identified. Typical model checking has just two output values, correct or incorrect. However, this does not suffice in GP since not having enough number of objective values impedes the gradual improvements of GP search. To cope with this problem, we identify the number of possible states that result in violation of given properties and assign objective value accordingly. For implementation, we use a model checking technique, Binary Decision Diagrams (BDDs). Using BDD-based evaluation, not only are generated programs accurately evaluated, but also the objective value has more possible values(the number of violated states) which allows GP to search with better gradient.

It is clear that the generated fault-tolerant programs should be correct under all scenarios. The

program is verified based on a number of requirements(similar to objective functions) in each scenario, and the aggregated verification result of a scenario is considered as a criterion. Hence, using the aggregated scenario-wise approach, the scenario-dependent MSMO problem is converted into multi-objective problem. In this study, we use NSGA-II combing with GP to model fault-tolerant programs which satisfy requirements under all given scenarios. We consider two types of typical faults - process crash failures and Byzantine faults. The correctness of fault-tolerant programs is characterized by verification in various scenarios such as in the absence of faults, in the presence of one fault, and in the presence of multiple faults. The detailed GP framework is discussed next.

5.2.2 GP Framework

The overall framework of GP is as follows: User inputs program variables, their domain and the desired fault-tolerance properties. We create a program structure based on the approach described in Section 5.2.2.1, and GP takes these inputs and generates the corresponding fault-tolerant program. To construct guarded commands of the distributed program, we consider linear representation[40], i.e., a linear sequence of statements, and implement evolution by stack-based GP [41, 42]. Stack-based GP represents a program as a list of functions that consuming from its stacks. It is able to directly perform on the linear statements, guarantees the safety of resulting programs not affected by introns. For genome, we use a vector of integers which are decoded into elementary units of a statement. The decoded units comprise into conditional statements by a pair of stacks (an operator stack and an operand stack). These conditional statements are the conditions will be defined in the following sections.

5.2.2.1 Program Representation in GP

The goal of GP is to evolve the given set of programs. In this dissertation, we begin with a set of statements st_1, st_2, $\cdots$, st_n and focus on evolving the guards under which they can be executed. While these statements could be chosen arbitrarily, we select them automatically based on the two criteria: 1) For each variable v, create a statement that assigns v a value based on the specification of the given program; 2) For each variable v, create a statement that assigns v a value from domain of v. If the given program has multiple variables in a given process, the same process is repeated for each variable. However, to avoid explosion of possible number of statements, statements updating different variables are run in parallel. By running in parallel, we mean that that all conditions in the guard are evaluated based on the initial state, i.e., they do not consider the effect of updating one variable in updating other variables. Note that the parallel update of several variables prevents explosive growth in the number of conditions. For example, if we have two statements updating every variable then we would need to consider $2 * n$ possible conditions instead of 2^n possible conditions if the same condition was used to update all variables at once. Thus, the program structure containing variables x, y and z is as shown in Figure 5.2.

Actions for variable x
    if (condition 1) statement 1 for updating x
    elseif (condition 2) statement 2 for updating x
    ...
||Actions for variable y
    ...
||Actions for variable z
    ...

Figure 5.2: Program Structure

We consider some alternative approaches in Section 5.2.6.4 to evaluate performance of GP with different representation.

5.2.2.2 Objective Functions and Scenario-wise Approach

The objective function in a specific scenario computes the extent to which the current program satisfies its specification in that scenario. We consider two approaches for evaluating objective function.

5.2.2.2.1 Simulation-based objective function In simulation-based objective function, we consider different possible runs of the given program for each scenario to identify whether it satisfies its specification in that scenario. For example, if a given specification consists of N properties(objectives), namely $Q_1, Q_2, \cdots, Q_N$ then the aggregated objective value of program P for run e, $F(P)_e$, for this scenario, is given by the following formula:

$$F(P)_e = \frac{1}{n}(f(P \models_e Q_1) + f(P \models_e Q_2) + ... + f(P \models_e Q_N)),$$

where $f(P \models_e Q_i) = 1$ if P satisfies Q_i in run e, and $f(P \models_e Q_i) = 0$ if P does not satisfy Q_i in run e. For the case where multiple runs are utilized (e.g., with different inputs), the aggregated objective value is normalized by taking their average of them. Thus, in this approach the objective function is always between $[0..1]$.

5.2.2.2.2 BDD-based objective function In BDD-based objective function, instead of executing programs GP statically verifies programs using BDD. We obtain a BDD representation of the current program, then utilize reachability techniques to identify reachable states, say R, of the given program. We identify states in R where either safety or liveness requirement is violated. For safety requirement, we obtain $R \cap sv$, where sv identifies states that imply violation of the safety condition. For liveness requirement of the form p leads to q, we obtain (1) states in R that are reachable from p without reaching q and are deadlocked, and (2) states in R that are reachable

from p and from where we could reach a cycle without reaching q. If the current program is correct then the obtained set is the empty set. If not, we identify the number of min-terms in the set. We note that BDD packages provide an efficient approach to count the min-terms without enumerating them. Again, in a scenario, the objective value - the total number of min-terms - is aggregated from the number of min-term of all properties in that scenario.

5.2.2.2.3 Scenario-wise approach A typical fault-tolerant program is required to satisfy the desired behavior in different scenarios: in the absence of faults, in the presence of a single fault, in the presence of multiple faults, etc. Our goal is to find programs that equally work well in all scenarios. The problem is expected to have an optimal solution whose objective value is optimal in all scenarios, which can be referred as single-objective problem. However, we found that the performance of optimization is better in diversity, if we solve the problem in the manner of multi-objectivization. The comparative study is given in section 5.3. By applying scenario-wise approach, we set one objective function for each scenario, and use NSGA-II to evolve the optimal program.

5.2.2.2.4 Reevaluation of objective functions Running GP with the objective function results in the following possible outcomes: (1) we obtain a correct program, (2) we fail to obtain a program with optimum objective value, or (3) we find a program with an optimum objective function although it is incorrect (as can be determined with BDD-based analysis). The last case only occurs in simulation-based approach, as BDD-based reachability analysis is *verifying* whether the properties are satisfied. However, simulation-based approach is only testing a subset of computations. For the second outcome, for programs in maximum-parallelism semantics, we allow the program to consist of several possible programs. This allows the processes to run different programs in dif-

ferent rounds. (Note that this is not necessary for interleaving semantics, as the program does not have a *round number* in which it is executing. Intuitively, this approach allows conditions (from Figure 5.2) to contain round numbers.)For the third outcome, we create a new objective function based on the counterexample that demonstrates the violation of the specification and utilize it in running the program.

In following sections, we model three fault-tolerant distributed programs using the methodology described above - Byzantine agreement program, token ring circulation problem and consensus with failure detector.

5.2.3 Synthesis of Byzantine Agreement Problem

In the Byzantine agreement problem, it consists of one general, g and three non-generals, j, k, and l. First, the general makes a decision and communicates it to the non-generals. After communication with the general and with each other, the non-generals need to finalize their decisions. This communication could be subject to a Byzantine fault where the Byzantine participant (general or non-general) sends incorrect values to other participants. The Byzantine agreement problem requires following specification:

1. Validity: if the general is non-byzantine then the final decision of a non-byzantine non-general must be the same as that of the general.

2. Agreement: the final decision of two non-Byzantine processes must be the same.

3. Termination: all non-Byzantine non-generals must eventually finalize their decision.

Given the structure of the problem, each process maintains a decision variable d. We concatenate the name of the process to denote the variable of that process. Hence, $d.g$ denotes the decision

of process g, $d.j$ denotes the decision of process j and so on. The decision of a non-general can be 1, 0, or $\perp$ (where $\perp$ denotes that the process has not yet received the decision from the general). Additionally, each process maintains a read-only variable b. Thus, $b.j$ denotes that process j is Byzantine. Finally, all non-generals maintain a variable f; $f.j$ denotes j has finalized its decision or whether the decision of j is temporary.

5.2.3.1 Program representation

Consider variable $d.j$ from process j. The domain of $d.j$ is $0, 1, \perp$. Hence, by criterion 2, we need to consider the case where $d.j$ is assigned either 0, 1 or $\perp$. Also, by criterion 1, we need to consider the case where $d.j$ is assigned $d.g$ (due to validity requirement) or $d.j$ is assigned $d.k$ or $d.l$ (due to agreement requirement), where k and l are other non-general processes. Of these, we do not include $d.j = \perp$ since $\perp$ is intended to be used as uninitialized value. Process j has another variable $f.j$. Based on the criteria above, we need to consider the case where $f.j$ assigned either 0 or 1. Based on the specification, i.e., once a process finalizes its decision, it cannot undo it, we only consider the case where f is set to 1. Thus, the modeling of Byzantine agreement is of the form:

Actions for variable $d.j$
- if (condition 1) $d.j = d.g$
- elseif (condition 2) $d.j = d.k$
- elseif (condition 3) $d.j = d.l$
- elseif (condition 4) $d.j = 0$
- elseif (condition 5) $d.j = 1$

||Actions for variable $f.j$
- if (condition 6) $f.j = 1$

Figure 5.3: Byzantine Program Structure

In our experiments, we find that while the statements $d.j = d.k$ and $d.j = d.l$ allow us to

find the desired program, the remaining 4 statements provide optimal performance. Hence, we focus on those four statements in most experiments. We evaluate the effect of other choices in the experimental section.

5.2.3.2 Evaluation of Objective Functions Using Simulations

Based on the description of the specification, we consider three scenarios for the agreement problem: (1) when no process is Byzantine, (2) when some non-general is Byzantine, and (3) when the general is Byzantine. The goal of GP is to evolve programs that work equally well in all these scenarios. The objective $F(P)$ consists of following three objective function.

$$F(P) = [F_{no\_byz}(P), F_{byz\_non\_general}(P), F_{byz\_general}(P)]$$

5.2.3.2.1 Scenario 1: Effectiveness when there is no Byzantine process Objective function is calculated as the average of two distinct cases: the decision of the general $d.g$ is 0; $d.g$ is 1. For each case, the program is evaluated based on the 9 properties as below.

- From validity: $Q_1 = (d.j == d.g), Q_2 = (d.k == d.g), Q_3 = (d.l == d.g)$
- From agreement: $Q_4 = (d.j == d.k), Q_5 = (d.k == d.l), Q_6 = (d.j == d.l)$
- From termination: $Q_7 = (f.j == 1), Q_8 = (f.k == 1), Q_9 = (f, l == 1)$

The objective function for this scenario $F_{no\_byz}(P)$ is now calculated by counting the number of properties that are satisfied and normalizing to 1.

$$F_{no\_byz}(P) = \frac{1}{2} \sum_{d.g=1}^{2} \frac{1}{9} \sum_{i=1}^{9} f(P \models Q_i)$$

5.2.3.2.2 Scenario 2: Effectiveness when one of the non-generals is a Byzantine Let l be the Byzantine process of a non-general. A byzantine process can send different decisions to different participants. If l is the Byzantine process then only Q_1, Q_2, Q_4, Q_7 and Q_8 (denoted as $Q_{relevant_l}$) have to be satisfied. Since we have three non-generals, the objective function for this scenario is defined as follows:

$$F_{byz\_non\_general}(P) = \frac{1}{2}\sum_{d.g=1}^{2}\frac{1}{3}\sum_{byz}^{j,k,l}\frac{1}{5}(f(P \models Q_{relevant_{byz}})$$

5.2.3.2.3 Scenario 3: Effectiveness when the general is a Byzantine If the general is Byzantine, then only $Q_4 - Q_9$ need to be satisfied. This experiment is repeated t times and the objective function for this scenario, hence,

$$F_{byz\_general}(P) = \frac{1}{t}\sum_{m=1}^{t}\frac{1}{6}\sum_{i=4}^{9} f(P \models Q_i);$$

5.2.3.3 Evaluation of Objective Functions using BDDs

We use the same three objectives with BDDs as well. The only difference is that violation of these objectives would be determined based on the number of min-terms (states) that indicate violation of the specification.

5.2.4 Synthesis of Token Ring Problem

This section synthesizes a fault-tolerant binary token ring program. A token ring program can be used to implement mutual exclusion by requiring that a process enters critical section only if it has the token. The token ring program consists of $N + 1$ processes $0, 1, ..., N$. The token circulates along the processes from 0 to N. Each process has one variable x with the domain of $\{0, 1, \perp\}$,

where $\perp$ denotes that the x value has been corrupted by a detectable fault. Process j, $j \neq 0$, has the token if and only if processes j and $j-1$ are not corrupted and $x.j \neq x.(j-1)$. Process 0 has token if and only if $(x.0 == x.N)$ where processes 0 and N are not corrupted. Token ring circulation problem that is subject to *restart* of a process, where a process fails and is restored to a fixed initial state Our goal is to find token ring program that tolerates up to N process (except process 0) corruption.

5.2.4.1 Modeling Program Statements and Objective Functions

The generated token ring program is expected to satisfy following specifications:

1. Safety Specification: There is one token in the ring at any time.

2. Liveness Specification: Token also should be circulated infinitely often.

Based on the specification requirements and the two criteria for designing statements from the beginning of Section 5.2.2.1, we allow $x.j$ value of a process to be updated to either 0 or 1 or a value based on $x.(j-1)$. The values 0 and 1 are based on the domain of $x.j$ and value of $x.(j-1)$ is based on the definition of j having a token. Also, similar to the byzantine agreement program, we consider three scenarios: (1) in the absence of faults and (2) in the presence of one fault, and (3) in the presence of multiple faults. Thus, the objective function is:

$$F(P) = [F_{no\_fault}(P), F_{one\_fault}(P), F_{multiple\_faults}(P)]$$

These objective functions are computed using the approach similar to the Byzantine agreement problem.

5.2.4.1.1 Scenario 1: Effectiveness when no faults occur We do not use fault actions in this scenario. In simulation-based approach, we utilize a parameter num to identify the number of rounds for which the program executes. Subsequently, we analyze this run to determine whether safety is violated or whether some processes fail to obtain the token in that run. We compute $F_{no\_fault}(P)$ based on the fraction of runs that failed to satisfy safety and/or liveness. In BDD-based approach, we compute the reachability states by ignoring the faults. Then, we identify states (minterms) that imply violation of safety or the inability of some process to get the token in the future. Thus, $F_{no\_fault}(P)$ is computed to be the number of such minterms.

5.2.4.1.2 Scenario 2: Effectiveness when one process is corrupted The second objective function considers the case where one process corruption occurs during the execution.

5.2.4.1.3 Scenario 3: Effectiveness when multiple processes are corrupted In this case, multiple processes are corrupted during the execution.

5.2.5 Synthesis of Consensus with Failure Detector S

In this case study, we focus on the problem of consensus using failure detector S. Solving consensus in the presence of the crash fault using failure detector S, that guarantees that (1) there exists a non-failed process that is never suspected by any other process, and (2) if a process fails then it is eventually detected by all non-failed processes. In the consensus protocol, each process starts with a vote. The goal of the protocol is that all processes reach an unanimous decision based on the set of votes. The specification for consensus problem is as follows.

1. Termination: Every correct process eventually decides some value.

2. Uniform integrity: Every process decides at most once.

3. Agreement: No two correct processes decide differently.

4. Uniform validity: If a process decides a value v, then some process proposed v.

5.2.5.1 Modeling Program Statements and Objective Functions

As the goal of this protocol is to share the votes of processes to obtain a uniform view, we include the variable V_p (to denote knowledge about votes of process p) and the variable D_p (to denote the messages sent by p to share votes). Each of these variables is an array and $V_p[q]$ denotes the knowledge that p has about q. Likewise, $D_p[q]$ denotes the vote sent by p about its the vote of q. Subsequently, we identify the statements by which V_p and D_p could be updated. Based on the semantics of V_p, $V_p[p]$ is initialized to 0 or 1 based on the vote of p. And, for $p \neq q$, $V_p[q]$ is initialized to $\perp$ denoting than p is not aware of the vote of q. Finally, $D_p[q]$ is initialized to $\perp$ since no process has received any messages before the first round.

Given the structure of statements of programs from Section 5.2.2.1, we considered the case where $V_p[q]$ is assigned either 0, 1, $\perp$ or the message it received $D_p[q]$. Likewise, possible values for $D_p[q]$ are either 0, 1, $\perp$ or $V_p[q]$. Of these, we omitted the values 0 or 1 since process p should not unilaterally change vote of q, $p \neq q$. (This restriction is not absolutely necessary since protocols where process p changes the vote of q will fail to satisfy safety property.) We used the following constraints to determine whether the specification is satisfied. Of these, constraint Q_1 captures that the set of votes collected is nonempty; this allows us to guarantee termination. Uniform integrity is guaranteed by the program structure since all processes finalize after the given number of rounds. Agreement is checked by constraint Q_2 by making sure that the votes collected by any two processes are identical. And, uniform validity is guaranteed by constraint Q_3.

$Q_1 : \forall p, \exists q \; such \; that \; (V_p[q] \neq \perp)$

$Q_2 : \forall p, \forall q : \forall j, (V_q[j] == V_p[j])$

$Q_3 : \forall p, \forall q, (V_p[q] \neq \perp) : (V_p[q] == V_q[q])$

For this case study, when we considered the program execution in maximum-parallelism semantics, GP was not able to synthesize the desired program in the first step. Based on the discussion about reevaluation of objective function from Section 5.2.2.2, we considered adding multiple programs for different rounds. In simulation-based approach, we could find the program when the number of programs was 3. For objective functions we consider three different scenarios in simulation-based approach described below. BDD-based approach only needs first two scenarios.

5.2.5.1.1 Scenario 1: Effectiveness when there is no message loss and failure In this case, we consider the case where every process is correct and, hence, no messages are lost.

5.2.5.1.2 Scenario 2: Effectiveness when there is one or more failures with different message loss models This case simulates both message loss and process failures. To consider as many different situations as possible, we apply different message loss rates ranging from 20% to 80% combining with various number of failure processes. Programs are tested up to 20 times in the experiments.

5.2.5.1.3 Scenario 3 In simulation-based approach, with the use of these two objectives, we were able to obtain the programs that satisfy objective function. However, it did not meet the specification of consensus. Hence, we defined a objective function that captured the corresponding counter-example as Objective 3.

5.2.6 Experimental Results

In this section, we describe the results of our analysis on the different case studies presented earlier. We performed experiments in both maximum-parallelism and interleaving semantics. We also performed experiments using simulations and using analysis via BDDs.

5.2.6.1 Experimental Setup

In these experiments, we use the population size of 100 (for Byzantine agreement and token ring) and 500 (consensus with S) and each individual (i.e., possible programs) in the population consists of one or more genomes. Within a genome the maximum length of conditions for an action is set to 20 and the maximum length is fixed during the evolution. For genetic operator, mutation has the probability ranging from of 0.05 to 0.1. We set the maximum number of generations to 500.

We partition our experimental results based on the comparison of different semantics considered in the dissertation and based on the different ways to evaluate objective functions. To show the comparison of objective value evolution we scale all objective values to the range of [0, 1] and convert all optimize processes to the maximization problem.

5.2.6.2 Simulation-based Approach vs BDD-based Approach

In this section, we compare the performance of GP using two evaluation approaches: simulation-based and BDD-based reachability analysis. We use all three case studies in this comparison and use maximum-parallelism semantics for them.

5.2.6.2.1 Byzantine agreement Recall that the program for Byzantine agreement is based on the structure in Figure 5.3. We consider one of the variations of this program which uses only conditions 1, 4 and 5. The Figure 5.4 shows the variation of population-average objective value of

each scenario with generation number. It is clear from the figure that all three objectives of BDD-based approach reach the maximum value quickly (before 15th generation), whereas simulation-based approach converges to optimal solution after running 40 generations.

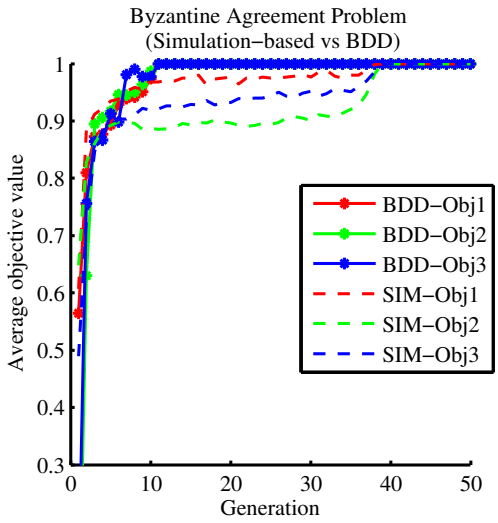

Figure 5.4: Variation of three objectives with generation is shown for the Byzantine agreement problem using two different evaluation methods.

This is achieved in part by the fact that a program may get *lucky* in simulation-based approach because the errors in it are not discovered by the simulation. The objective function in BDD-based approach (states that indicate violation of the specification) is significantly more fine-tuned than that of simulation-based approach.

5.2.6.2.2 Token ring The results for token ring are as shown in Figure 5.5. A similar observation can be made here as well. Though the difference is not as much as Byzantine agreement problem because the simulation-based approach could find the desired program in a few generations, the BDD-based approach still performs better.

5.2.6.2.3 Consensus with failure detector S The results for failure detector are as shown in Figure 5.6. In this figure, the BDD-based approach shows better convergence, and equally em-

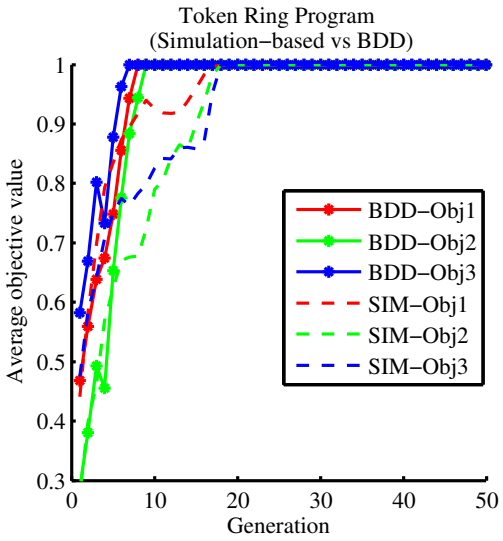

Figure 5.5: Variation of three objectives with generation is shown for the token ring program using two different evaluation methods.

phasizes all three objectives. For simulation-based one the first two objectives could reach the points near to the maximum quickly, however, the third objective fluctuates heavily throughout the evolution and does not converge to the optimal value.

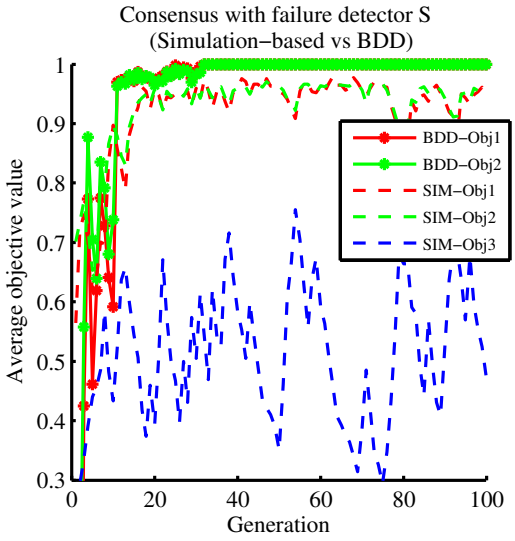

Figure 5.6: Variation of three objectives with generation is shown for the Consensus with failure detector S using two different evaluation methods.

To summarize the outcome of these two approaches of all above case studies, the performance metrics for obtaining an optimum program that also satisfies the desired specification is as shown

in Table 5.1. We tabulate the number of successful runs and best, median and worst number of generations (of up to 25 runs) required to arrive at the optimal solution that optimizes all of the objectives.

Table 5.1: Performance of simulation-based and BDD-based approach using maximum-parallelism semantics

Optimization Problem	Evalutaion Method	Success Rate (%)	# of Gens to Converge Min	Median	Max	Time (Sec) /Gen
Byzantine	BDD	100	3	6	63	0.65
	Simulation	100	5	29	386	3.2
TokenRing	BDD	100	4	6	37	0.4
	Simulation	100	3	15	87	2
Consensus	BDD	96	7	33	404	22
	Simulation	80	8	179	479	69

The table shows that BDD-based approaches are quicker in the most of their runs taking less number of generations to find the solution. In terms of the time, the simulation-based approaches cost much more running time since they execute each generated program total upto 50 times in all scenarios. Meanwhile, BDD-based reachability analysis is more efficient as well as the fact that we need to consider one symbolic run with BDDs as opposed to simulation-based approach. Besides, in consensus case study BDD-based one is even more successful. This is also expected as BDD-based approach provides higher graduation in objective value that allows better diversity, which is preferred by multi-objective optimization.

5.2.6.3 Comparison of Maximum-parallelism Semantics and Interleaving Semantics

In this section, we compare the performance of GP under maximum-parallelism semantics and under interleaving semantics. The analysis performed in this section is based on BDD-based approach.

We use the same program format from previous subsection for analysis of Byzantine agreement. The results are as mentioned in Figure 5.7. From this figure, we find that the GP works better with maximum-parallelism semantics. The objective function converges faster in maximum-parallelism semantics. This is due to the fact that for interleaving programs, processes are required to work well in all possible orders which leads to smaller set of optimal solutions existing in the search space compared to maximum-parallelism. In general, for the same program there are more reachable states in interleaving semantics than maximum-parallelism one that increases the possibility of violated states.

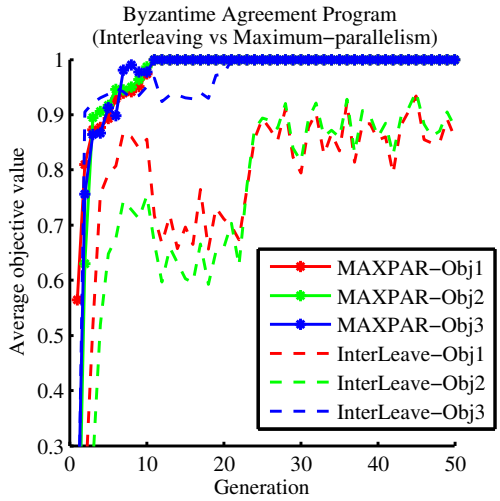

Figure 5.7: Variation of three objectives with generation is shown for the Byzantine agreement problem using two different semantics.

For token ring in Figure 5.8, both approaches work equally well. We expect that this is, in part, due to simplicity of that program.

For the failure detector, we were not able to synthesize the required program in interleaving semantics. However, as mentioned earlier, the program in maximum-parallelism semantics can be implemented in interleaving semantics with addition of round numbers.

Table 5.2 shows the performance metrics of maximum-parallelism semantics and interleaving semantics. As mentioned above in the Byzantine agreement case study, the maximum-parallelism

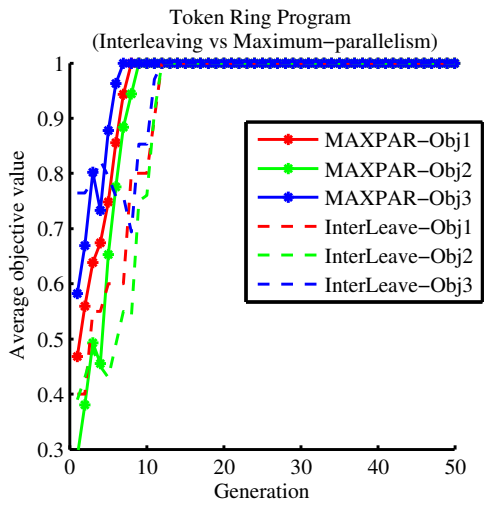

Figure 5.8: Variation of three objectives with generation is shown for the Token ring program using two different semantics.

outperforms in converging speed, and in this table it shows interleaving semantics are also less successful. In terms of the time, there is no difference between two semantics.

Table 5.2: Performance of maximum-parallelism semantics and Interleaving semantics using BDD evaluation

Optimization Problem	Semantics Format	Success Rate (%)	# of Gens to Converge Min	Median	Max	Time (Sec) /Gen
Byzantine	Max-Par	100	3	6	63	0.65
	Interleaving	24	189	309	443	0.65
TokenRing	Max-Par	100	4	6	37	0.4
	Interleaving	100	3	7	39	0.4

5.2.6.4 Effect of Choice of Statements

We also considered the effect of using different statements in the synthesis. For example, Figure 5.3 identifies five possible statements for updating $d.j$ in Byzantine agreement program. We consider different subsets of these five statements to identify their effectiveness. In this experiment we test three different formats:

(1) *4-Statements-1* uses conditions 1, 4 and 5;

(2) *4-Statements-2* uses conditions 1, 2 and 3;

(3) *5-Statements* uses conditions 1, 4, 5 and compliment operation on $d.j$.

The results are presented in Table 5.3 . The table reveals an interesting fact that *4-Statements-2* has worst performance among all three variations of the program in both evaluation types, and the rest of two formats show the similar observation. This implies the program format influences the search process. We also observe that even in different program formats the BDD-based approach performs better in overall.

Table 5.3: Performance of three different choices of statements solving Byzantine agreement problem in Maximum-parallelism semantics

Evaluation Type	Statements Format	Success Rate (%)	# of Gens to Converge Min	Median	Max	Time (Sec) /Gen
BDD	4-Statements-1	100	3	6	63	0.65
	4-Statements-2	96	4	18	222	0.65
	5-Statements	100	4	8	68	0.65
Sim	4-Statements-1	100	5	29	386	3.2
	4-Statements-2	88	6	68	111	3.2
	5-Statements	96	4	31	363	4

5.3 Comparative Study: Single-objective Optimization

In the above section, we use scenario-wise approach to model fault-tolerant programs. However, one may raise a question whether MSMO approach is effective compared to other classic approaches. To validate our proposed methods, in this section, we compare our proposed methods with straight forward single-objective approach. The single-objective approach aggregates all these objectives into one objective and the problem is converted into single-objective optimization problem.

General description of the approach is given below. The leftmost aggregation ($\sqcup$) is denoted as outer-aggregation, others are denoted as inner-aggregation. The inner aggregation is the same with aggregated scenario-wise approach, and the outer aggregation is one more layer above it.

$$\begin{aligned} &\text{Minimize} \quad \sqcup_{k=1}^{K} \left\{ \sqcup_{m=1}^{M^{(k)}} f_m^{(k)}(\mathbf{x}) \right\}, \\ &\text{Subject to} \quad g_j^k(\mathbf{x}) \leq 0, \quad j = 1, 2, ..., J^k, \ k = 1, 2, \ldots, K. \end{aligned} \tag{5.3}$$

Once the aggregation is done, any corresponding optimization method can be applied. For better comparison we use the same case study and experimental setting with the above sections. In the following sections, we describe the comparison results and analysis.

5.3.1 Aggregation Methods Used for Single-objective Optimization

For comparison, we use the same aggregation method with aggregated scenario-wise approach for inner aggregation of single-objective optimization, and carry out experiments on several different outer aggregation methods which are given as follows.

5.3.1.1 Worst-case Aggregation

A common strategy which is followed in practice is to find the worst cost of all scenarios and is used as the objective function of the optimization problem, that is, for minimization problems,

$$\sqcup_{k=1}^{K} F(k, \mathbf{x}) = \max_{k=1}^{K} F(k, \mathbf{x}) \tag{5.4}$$

where $F(k, \mathbf{x}) = \sqcup_{m=1}^{M^{(k)}} f_m^{(k)}(\mathbf{x})$. Thus, for a minimization problem, the overall problem becomes a min-max problem. For maximization problems, the operator $\max$ should be replaced by a $\min$ operator, thereby having a max-min problem. This approach may provide a pessimistic esti-

mate of the objective function, since the worst-case scenario may be an isolated event which may not represent the true performance over all K scenarios. The following approach can be used.

5.3.1.2 Average-case Aggregation

An average objective value of all K scenarios can be used, instead:

$$\sqcup_{k=1}^{K} F(k, \mathbf{x}) = \mu_f^F(k, \mathbf{x}) = \frac{1}{K} \sum_{k=1}^{K} F(k, \mathbf{x}). \tag{5.5}$$

The average-case scenario is logical particularly when there is not much difference in performance among all K scenarios. When the variance in performance among K scenarios is large, a more statistically favorable aggregation would be better, such as the median-case or the following measure.

5.3.1.3 Mean-variance Aggregation

Instead of an average, the mean and standard deviation of the objective function among all K scenarios can be used for minimization problems:

$$\sqcup_{k=1}^{K} F(k, \mathbf{x}) = \mu_f^K(\mathbf{x}) + \kappa \sigma_f^K(\mathbf{x}), \tag{5.6}$$

where $\mu_f^K(\mathbf{x})$ is the mean of $f()$ over K scenarios and $\sigma_f^K(\mathbf{x})$ is the corresponding standard deviation. The parameter κ can be chosen as 1, 2 or 3, depending on the importance of the standard deviation over the mean. Such an aggregate measure will give adequate importance to the distribution of $f()$ for different scenarios. For maximization problems, κ can be considered negative.

5.3.1.4 Weighted-sum Aggregation

In a generic problem setting, different scenarios may have different importance. For example, if p-th scenario happens more often than q-th scenario, then a large weight can be used for the p-th scenario and the following weighted-sum function can be used:

$$\sqcup_{k=1}^{K} F(k, \mathbf{x}) = \frac{1}{\sum_{k=1}^{K} w_k} \sum_{k=1}^{K} w_k F(k, \mathbf{x}), \tag{5.7}$$

Certainly, other aggregate methods are possible, but the above presents the most common strategies.

5.3.2 Experimental Results

To compare with our scenario-wise approach, we use one of case studies used in Section 5.2.6, the byzantine agreement problem to investigate a comparative study. We adopt the same experimental settings for GP, use the simulation-based objective functions and *4-Statements-1* program structure in single-objective optimization.

5.3.2.1 Worst-case Aggregation Results

In the worst-case outer aggregation, the objective function is computed as the worst performance among the three scenarios:

$$F(P) = \min \left(F_{no\_byz}(P), F_{byz\_non\_general}(P), F_{byz\_general}(P) \right).$$

Figure 5.9 shows the variation of population-average objective value of each scenario with the generation number. It is clear from the figure that F_{no_byz} and $F_{byz\_non\_general}$ reach their maxi-

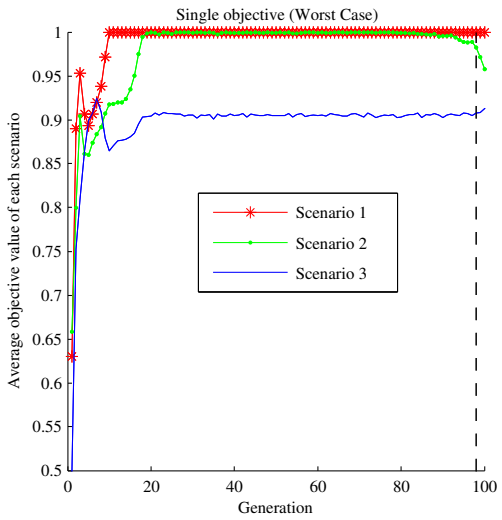

Figure 5.9: Variation of three objectives with generation is shown for the worst-case outer aggregation method. The vertical dashed line shows the generation when the optimal solution for all three objectives (equal to one) is obtained for the first time.

mum values quickly, whereas $F_{byz\_general}$ takes a large number of generations to come close to its maximum value. In this case, the optimal solution $P*$ that maximizes all three objectives is found at generation 98. The corresponding solution is presented in Figure 5.10.

Actions for $d.j$

if	$(d.j == \perp) \wedge (f.j \neq 1) \wedge (d.j \neq 1)$	then	$d.j = d.g$
elseif	$(d.k == d.l) \wedge (d.k == 0)$	then	$d.j = 0$
elseif	$(d.k == d.l)$	then	$d.j = 1$

Actions for $f.j$

if	$(d.j \neq \perp) \wedge ((d.l \neq \perp) \vee (d.l == \perp))$		
	$\wedge((d.l == d.j) \vee (d.k == d.j))$	then	$f.j = 1$

Figure 5.10: One of the Generated Solution for Byzantine Agreement Program

Although the optimal solution was found eventually, due to the unequal emphasis for all three objectives in the worst-case aggregation scheme, it was difficult for the GP to find the optimal solution quickly.

5.3.2.2 Average-case Aggregation Results

The three objective functions are simply averaged here and the average function value is maximized. Figure 5.11 shows the variation of population-average value of all three objectives. A

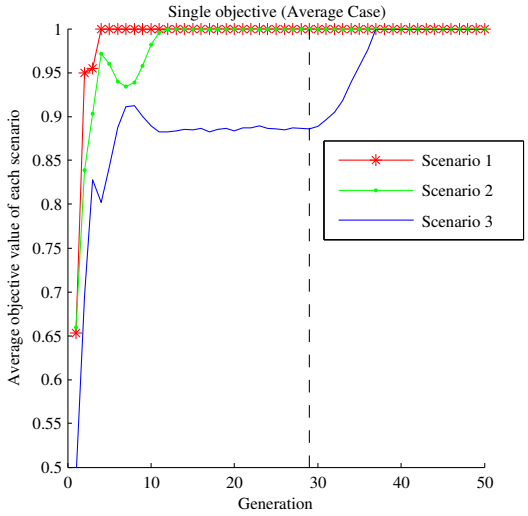

Figure 5.11: Variation of three objectives with generation is shown for the average-case outer-aggregation method.

similar observation can be made here as well, however, the convergence to the true optimal solution is quick. Due to the averaging effect, although all three objectives get emphasized, the process still sets a hierarchical importance to three objectives.

5.3.2.3 Mean-variance Aggregation Results

Next, we use the following aggregate function derived from three objective values:

$$F(P) = \mu\left(F_{no\_byz}(P), F_{byz\_non\_general}(P), F_{byz\_general}(P)\right)$$
$$-2\sigma\left(F_{no\_byz}(P), F_{byz\_non\_general}(P), F_{byz\_general}(P)\right).$$

and maximize $F(P)$. Figure 5.12 shows the variation of population-average objective values. Due to the consideration of both (increasing) mean and (reducing) standard deviation of objective

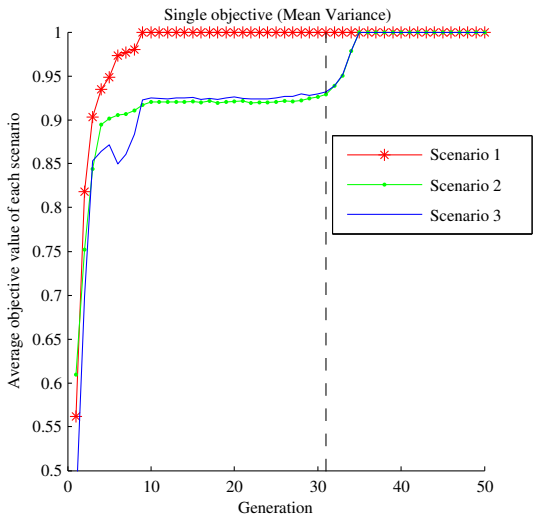

Figure 5.12: Variation of three objectives with generation is shown for the mean-variance outer-aggregation method.

values, all three objectives get emphasized and the optimization method is able to quickly converge to the true optimal solution.

5.3.2.4 Weighted-sum Aggregation Results

The worst-case and average-case aggregation results have shown that it is relatively easy to solve $F_{no\_byz}(P)$ and $F_{byz\_non\_general}(P)$, and in both cases the algorithm waits until it solves the third objective to find the true optimum. If for this reason or for another reason of preferring the third objective more than the first two objectives, we use the following weighted-sum of three objectives as an aggregation scheme:

$$F(P) = 0.25F_{no\_byz}(P) + 0.25F_{byz\_non\_general}(P) \\ +0.5F_{byz\_general}(P),$$

and maximize $F(P)$. Figure 5.13 shows the variation of population-average objective values. Due to the emphasis put on the third objective now, the third objective reaches its maximum quickly,

but the the first two objective takes a long time to converge to their maximum values. Note that for the third objective, there exist a number of optimal solutions. Unfortunately, the algorithm converges to one solution that does not correspond to the maximum of the first two objectives. It takes the algorithm 200 generations to find the optimum that is shared between all three objectives. Therefore, if different weights need to be used for different objectives, the use of a single weighted-sum of objectives ($F(P)$) may not be the right way forward.

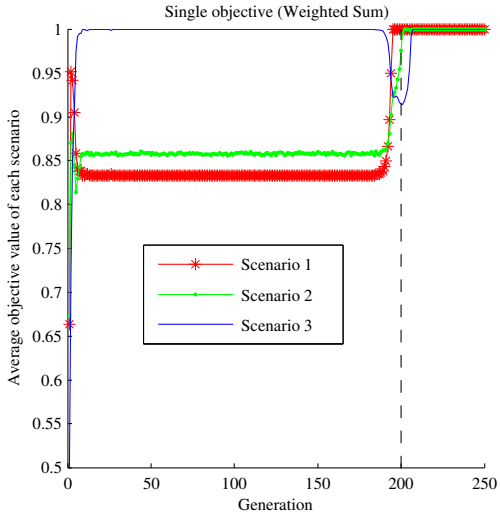

Figure 5.13: Variation of three objectives with generation is shown for the weighted-sum outer-aggregation method.

5.3.2.5 Scenario-wise Approach Results

In this section, we discuss about the result of scenario-wise approach(or multi-objective). Figure 5.14 shows the population-average objective values with generation. A comparison of this figure with that obtained for all the previous single-objective methods reveals an interesting fact. All three objectives increase more or less in a similar manner. Since all objectives are emphasized simultaneously in a multi-objective optimization problem, the population maintains a good diversity of solutions and no objective is ignored or less-emphasized. The optimal solution for all three

objectives is also obtained quickly by this method.

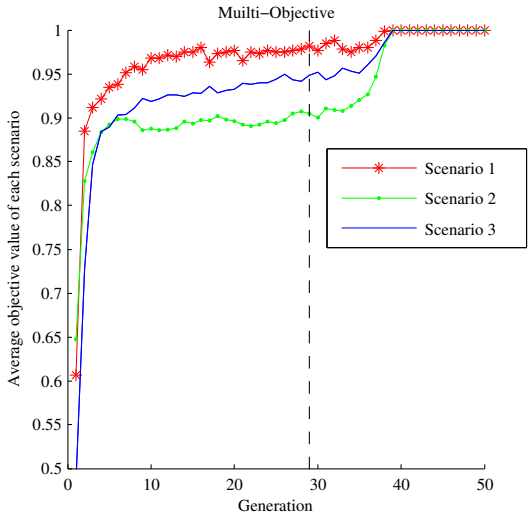

Figure 5.14: Variation of three objectives with generation is shown for the multi-objective method (scenario-wise approach).

5.3.2.6 Specific Bi-objective Aggregation Results

When the number of scenarios is large, the above multi-objective approach will be required to handle many objectives. Unfortunately, NSGA-II or other domination-based evolutionary approaches are not adequate in handling more than three or four objectives. However, recently proposed decomposition-based methods such as NSGA-III [31] or MOEA/D [30] are potential algorithms for handling many scenarios. Another approach to handling many scenarios would be to club a few scenarios together into one class and thereby reduce the number of scenarios for optimization. To demonstrate this method, we merge the first two scenarios into one and compute the combined objective function as $F_1(P) = \left(F_{no\_byz}(P) + F_{byz\_non\_general}(P)\right)/2$ and use the third objective as the second objective for a bi-objective optimization ($F_2(P) = F_{byz\_general}(P)$). NSGA-II is then employed to solve this specific bi-objective problem.

Figure 5.15 shows the variation of the population-average objective value of each of the three

original objectives. The growth of three objective values is closer to each other, meaning that the

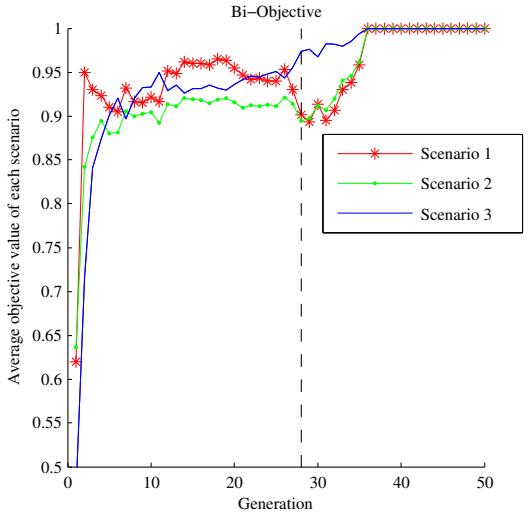

Figure 5.15: Variation of three objectives with generation is shown for the bi-objective method.

optimization algorithm is able to emphasize all three objectives in a similar manner when arriving at the optimal solution.

To summarize the outcome of all six methods, in Table 5.4, we tabulate the number of successful runs and best, median and worst number of generations (of 25 runs) required to arrive at the optimal solution $P*$ that maximizes all three objectives. The best values are shown in bold font. The table shows that scenario-wise MSMO methods are not only more successful, but they

Table 5.4: Performance of six algorithms for solving Byzantine agreement problem.

Optimization Approach	Success Rate (%)	Min # Gens. to Converge	Median # Gens. to Converge	Max # Gens. to Converge
Worst-Case	92	8	98	172
Average-Case	92	6	29	**111**
Mean-Variance	96	6	31	366
Weighted-Sum	48	5	200	421
MSMO	**100**	5	29	386
Bi-objective	96	**4**	**27**	349

are also quicker in most of their runs. Among the single-objective methods, average-case and the

mean-variance methods are better for this problem.

5.3.2.7 Discussion

It is clear that multi-objective (scenario-wise) approach is able to maintain an equal emphasis on each of the three objectives from the start to the end of the optimization process. The reason for this behavior is the natural diversity among optimal solutions for different objectives that multi-objective approach maintains. To investigate this aspect further, we compute a diversity measure as follows.

At every generation, we compute the centroid $\bar{\mathbf{F}}$ of the population members in the objective space. Then, we compute the the distance $d_i = \sqrt{\sum_{i=1}^{3}(f_i - \bar{f_i})^2}$ of each objective vector from the centroid. The diversity measure is then calculated by taking an average of the distance values, or $D = \sum_{i=1}^{N} d_i/N$. When this measure is plotted with generation counter, it will indicate the diversity of the population in the objective space. Figure 5.16 plots this diversity metric for all six methods.

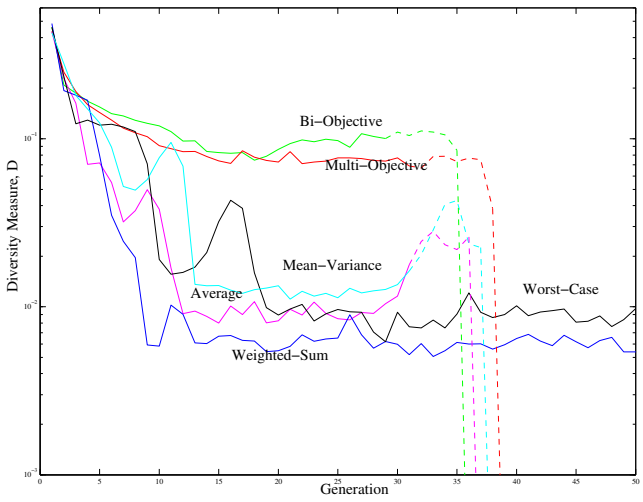

Figure 5.16: Diversity measure D with generation counter for six methods of this study.

It is clear from the plot that single-objective methods lose diversity very quickly, whereas both bi-objective and three-objective method are able to maintain a large diversity all along. For a method, if the optimal solution $P*$ is found, the variation after its occurrence in the population is marked with a dashed line. A reason for poor performance of some of the single-objective methods is their rapid loss of diversity. Once the population gets stuck to the optimum of one of the objectives and the population diversity is small, evolutionary algorithms have difficulties in getting out from there. In multi-scenario problems, an optimal solution for all scenarios is the target, and getting stuck to an optimum for one scenario is often detrimental in arriving at the desired solution. It is clear that due to presence of diversity in the population throughout the entire evolution process, MSMO methods have performed well in synthesizing Byzantine agreement problem.

Chapter 6

Optimization of Scenario-Independent MSMO Problems

6.1 Introduction

There exist fault-tolerant problems that have more than one descriptions (scenarios) and require different optimal solutions for each description of problem. We call such problem scenario-independent. We consider each scenario independently and optimize fault-tolerant problem in single scenario at a time.

Scenario-independent problem is considered as a multi-scenario, multi-objective problem, since optimization in only one scenario is not sufficient. For scenario-independent problem, analysis of optimal solutions from all scenarios help to obtain problem knowledge. Comparison of optimized solutions from different scenarios can also provide a better understanding of the problem. For instance, in designing the randomized self-stabilizing program - a type of fault-tolerant program, if a designer wants to understand potential benefits of using non-uniform probability distributions for randomised algorithms, it is better to investigate different descriptions of the problem. Different descriptions can provide broader knowledge of the problem and help to find the better program.

Figure 6.1 shows the structure of different scenarios of a problem. There are total three scenarios, and each scenario requires to optimize more than one objectives. For the randomized

self-stabilizing program design, the scenarios are the different descriptions of the same problem, and it is expected that optimal solutions from different descriptions can be different.

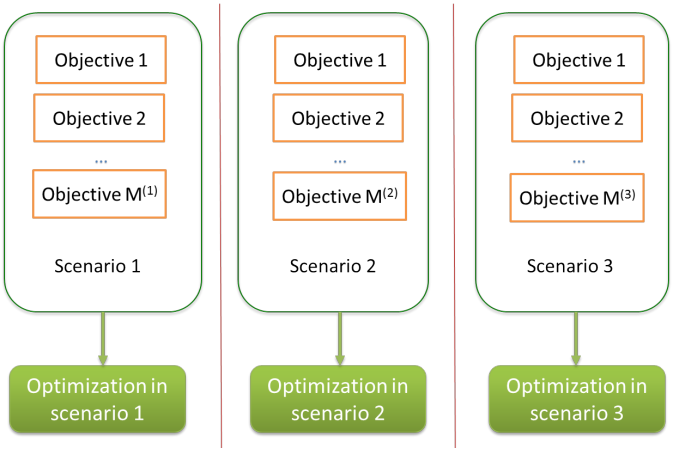

Figure 6.1: Consider each scenario in one optimization

To solve such problem, we optimize each scenario respectively, and analyze the correlation of optimal solutions from different scenarios.

6.2 General Problem Description and Design Approach

The general multi-scenario problem optimization with independent K different scenarios consists of K different optimizations where the optimized solutions shares the same design spaces. Each scenario has its own set of the optimized solution set, and solution sets from all scenarios can be used in analytic study providing better understanding of the problems and scenarios.

Let us denote $f_m^{(k)}(\mathbf{x})$ as the m-th objective function value for the k-th scenario, and $g_j^k(\mathbf{x}) \leq 0$ is the j-th inequality constraint for the k-th scenario. A generic scenario-independent K-scenario,

M-objective, and J-constraint problem can be written as follows:

$$\begin{aligned}
&\text{Minimize} \quad \left\{ f_1^{(1)}(\mathbf{x}), f_2^{(1)}(\mathbf{x}), \ldots, f_{M^{(1)}}^{(1)}(\mathbf{x}) \right\}, \\
&\text{Subject to} \quad g_j^1(\mathbf{x}) \leq 0, \quad j = 1, 2, \ldots, J^{(1)}. \\
&\text{Minimize} \quad \left\{ f_1^{(2)}(\mathbf{x}), f_2^{(2)}(\mathbf{x}), \ldots, f_{M^{(2)}}^{(2)}(\mathbf{x}) \right\}, \\
&\text{Subject to} \quad g_j^2(\mathbf{x}) \leq 0, \quad j = 1, 2, \ldots, J^{(2)}. \\
&\qquad \ldots \\
&\text{Minimize} \quad \left\{ f_1^{(K)}(\mathbf{x}), f_2^{(K)}(\mathbf{x}), \ldots, f_{M^{(K)}}^{(K)}(\mathbf{x}) \right\} \\
&\text{Subject to} \quad g_j^K(\mathbf{x}) \leq 0, \quad j = 1, 2, \ldots, J^{(K)}.
\end{aligned} \tag{6.1}$$

6.3 Case Study

In this section, we carry out a study on refining probabilistic self-stabilizing programs under different program variation. Each variation is considered as a scenario, and evolutionary multi-objective approach is applied to optimize the self-stabilizing token ring program under each scenario. Next, we describe self-stabilizing token ring and its optimization, and the corresponding program variation we optimize in the experiments.

6.3.1 Self-stabilizing Token Ring Problem

Self-stabilization, an approach for designing fault-tolerant distribued systems, is firstly introduced by Dijkstra [9]. [9] proposes K-state self-stabilizing token ring program which ensures ensures that starting from an arbitrary state, the program recovers to its legitimate states. The token ring program can also be used to implement mutual exclusion by requiring that a process enters critical section only if it has the token. This token ring program consists of N processes $0, 1, ..., N-1$. The token circulates along the processes from 0 to $N-1$, and then to 0. Each process has one

variable x with the domain of $\{0, 1, ..., N-1\}$. Process j, $j \neq 0$, has the token if and only if processes j and $j-1$ are not corrupted and $x.j \neq x.(j-1)$. Process 0 has token if and only if $(x.0 == x.(N-1))$ where processes 0 and $N-1$ are not corrupted. The token circulates in one direction from process 0 to $N-1$. Fig 6.2 is the original solution for token ring program. This solution guarantees that starting from an arbitrary state, the program eventually reaches a state where there is a unique token that is circulated among all processes.

Actions for Process 0
if $(x.0 == x.(N-1))$ then $x.0 = (x.0 + 1) mod N$

Actions for Process $j, j \neq 0$
if $(x.j \neq x.(j-1))$ then $x.j = x.(j-1)$

Figure 6.2: Token Ring Program

6.3.2 Probabilistic Self-stabilizing Program

A probabilistic self-stabilizing program, as a self-stabilizing program, eventually converges to legitimate computation with probability 1 starting from any global configuration. In this dissertation, we focus on [18, 19], variants of Dijkstra's token ring program, which are randomized solutions that recover from faults in token ring problem

6.3.3 Optimization in Probabilistic Self-stabilizing Programs

Programs in [18, 19] can have computations that stay outside legitimate states forever. However, by assigning probabilities to individual program actions, we can make this probability negligible. Although a probabilistic program guarantees that with probability 1, it will converge to a legitimate state, the expected convergence time can vary depending upon the probability values assigned to

different actions. Hence, if we want to minimize the expected convergence time, we need to choose these parameters accordingly. Moreover, these parameters can also affect how the protocols behave after convergence. In particular, it is possible that parameters that are ideal for reducing convergence may not be ideal after the program has been restored to legitimate states. This issue is especially challenging if the algorithm of interest uses multiple independent probabilities.

In this dissertation, we utilize GAs to analyze these programs for ideal probability values that will reduce the overall convergence time as well as token circulation time after recovery is complete. Moreover, in cases where there is a trade-off between convergence time and the time for token circulation in legitimate states, we want to identify the trade-off as well.

6.3.3.1 Criteria

One of the criteria is convergence time (or stabilization time). Convergence time evaluates the efficiency of self-stabilizing protocols. This is the time needed for a program recover from the states with an arbitrary number of tokens to legitimate states with one token. In this dissertation, we measure expected convergence time as criterion.

The other criterion is token circulation time (or service time). Token circulation time is usually used to measure the closure, it is the time needed for a token circulates from one processor and all the way back to the same processor. Token circulation time is also important, as it evaluates performance of the self-stabilizing program in the absence of faults and measure the corresponding overhead.

6.3.3.2 Measurements for Analysis

The trade-off of two criteria between different optimal solutions identified by GA are especially important. Besides we utilize another aspect: mean-time-between-failures (MTBF) to help with

analysis of the optimal solutions.

In particular, an algorithm that provides faster convergence time but provides a lower performance in the absence of faults is likely to be desirable in systems where MTBF is small. In this case, the quick convergence would make it possible for the system to operate correctly in an expedited manner. By contrast, if convergence time is high then it would be desirable when MTBF is high. In this case, the cost of the convergence time would occur only infrequently as faults occur far apart. With this motivation, we use GA to analyze the protocols in [19, 18], so that we can identify the optimal solutions in terms of convergence time as well the time for token circulation. We also utilize the generated solutions so that the decision makers can choose the appropriate values based on information such as MTBF.

6.3.4 Scenarios: Different Descriptions of the Program

6.3.4.1 Scenarios in Asymmetric Probabilistic Self-stabilizing Program

In this section, we recall the algorithm in [19] and its different descriptions which we will study in this dissertation. The program in [19] is a token circulation program that is a variant of the classic K-state stabilization program by Dijkstra [9] where a token is circulated along an unidirectional ring. In particular, the program in [19] shows that using probabilistic coin flip, one can obtain probabilistic stabilization with only 3 states per process. By contrast, for a deterministic program, each process needs to have n states. Since the exact structure of the program can affect the probability values, we consider different variations of this program.

Each variation of this program consists of n processes numbered $0..(n-1)$ that are organized in a ring. Each process j maintains the variable $x.j$ whose domain is $\{0, 1, 2\}$. The program is asymmetric in the sense that there is a special process (named 0) that runs a different set of actions

than all other processes. In this program, a non-zero process j has a token iff $x.j \neq x.(j-1)$. If a non-zero process has a token, it copies the value of the predecessor, namely $x.(j-1)$. Process 0 has a token iff $x.0$ equals $x.(n-1)$. We consider variations of this program, using different actions performed by process 0 when it has a token.

6.3.4.1.1 Scenario 1: Program $Asym_{incP}$ ($Asym_{incP_1}$ and $Asym_{incP_2}$): Increment $x.0$ with a random positive number Our first variation, $Asym_{incP}$ increases $x.0$ with a positive number in mod arithmetic when process 0 holds a token. For this variation, we consider two programs that have different domains: $Asym_{incP_1}$ and $Asym_{incP_2}$.

$Asym_{incP_1}$ considers the case where process 0 always changes its value whenever it has a token. Since the domain is $\{0, 1, 2\}$, we consider two actions that keep track of the current value. In particular, it either increases the value by 1 with probability λ and by 2 with probability $1-\lambda$. Thus, the actions are as follows:

$$
\begin{array}{lll}
A_0 :: \; x.0 == x.(n-1) & \longrightarrow & \lambda : \; x.0 := (x.0+1) \bmod 3 \\
 & & + \quad 1-\lambda : \; x.0 := (x.0+2) \bmod 3; \\
A_j \; j \neq 0 :: \; x.j \neq x.(j-1) & \longrightarrow & x.j = x.(j-1);
\end{array}
$$

In the above actions, A_0 is specifically designed for process 0. When the guard condition $x.0 == x.(n-1)$ is satisfied, process 0 is *privileged*, (respectively, holds a token). Then, with probability λ, process 0 increments its value by 1 in modulo 3 arithmetic. Alternatively, with probability $1-\lambda$, it increases the value by 2 in modulo 3 arithmetic. The second action is designed for non-zero processes j, $0 < j < n$. When the guard condition $x.j \neq x.(j-1)$ is satisfied, j is *privileged*. Subsequently, process j copies the value of its predecessor.

Observe that if all x values are initialized to 0 then only process 0 has the token. In this state, process 0 can either change its value to 1 or 2. Without loss of generality, let us consider the case

where process 0 changes its value to 1. In this state, only process 1 has the token. Execution of process 1 changes $x.1$ to 1. In this state, process 2 has the token. This execution continues until all x values change to 1. In this state, process 0 can execute again and change $x.0$. The states reached in the execution of this program are the legitimate states of the program. Additionally, as shown in [19], if the program starts in an arbitrary state, with probability 1, it is guaranteed to reach a legitimate state. In other words, although this program can have computations that stay outside legitimate states forever, the probability of such computations can be made smaller than any ϵ, $\epsilon > 0$. Furthermore, after it reaches a legitimate state, its subsequent execution only includes legitimate states.

As mentioned above, the above program is a variation of the K-state program by Dijkstra [9] in synchronous semantics. Specifically, in [9], the value of λ is always 1, i.e., process 0 always increments its value by 1. Although this program is often studied in interleaving semantics, it is known to be stabilizing in synchronous semantics as well. In this program, any computation is guaranteed to reach a legitimate state and stay there forever.

The above program can also be easily extended to the case where domain of x is slightly increased. Let denote such program as $Asym_{incP_2}$, the domain of x is $\{0, 1, 2, 3\}$, and process 0 increments $x.0$ by 1 in modulo 4 arithmetic with probability λ_1, by 2 with probability λ_2, and by 3 with probability $1 - \lambda_1 - \lambda_2$.

6.3.4.1.2 Scenario 2: Program $Asym_{incNN}$: Increment $x.0$ with random non-negative number The program above ensures the process 0 always changes its value whenever it has a token by increasing $x.0$ with positive number. In this section, we consider the case where the process 0 increases its value with random non-negative number. In other words, with probability λ_1, $x.0$ stays unchanged, with the probability λ_2, process 0 increments its value by 1 in modulo 3 arith-

metic, and, with probability $1 - \lambda_1 - \lambda_2$, it increases the value by 2 in modulo 3 arithmetic. The actions are as follows:

$$
\begin{array}{lll}
A_0 :: \ x.0 == x.(n-1) & \longrightarrow & \lambda_1 : \ x.0 := x.0 \\
& + & \lambda_2 : \ x.0 := (x.0 + 1) \bmod 3; \\
& + & 1 - \lambda_1 - \lambda_2 : \ x.0 := (x.0 + 2) \bmod 3; \\
A_j \ j \neq 0 :: \ x.j \neq x.(j-1) & \longrightarrow & x.j = x.(j-1);
\end{array}
$$

6.3.4.1.3 Scenario 3: Program $Asym_{rand}$: Assign a random value to $x.0$ The two variations mentioned in the above sections increase $x.0$ when process 0 holds a token. However, the third variation we consider in this section directly assigns a random value in the domain of the $x.0$. The actions are as follows:

$$
\begin{array}{lll}
A_0 :: \ x.0 == x.(n-1) & \longrightarrow & \lambda_1 : \ x.0 := 0 \\
& + & \lambda_2 : \ x.0 := 1; \\
& + & 1 - \lambda_1 - \lambda_2 : \ x.0 := 2; \\
A_j \ j \neq 0 :: \ x.j \neq x.(j-1) & \longrightarrow & x.j = x.(j-1);
\end{array}
$$

6.3.4.2 Scenarios in Symmetric Probabilistic Self-stabilizing Program

In this section, we recall the symmetric (anonymous) token ring program in [18]. Similar to the program in Section 6.3.4.1, this program also arranges the processes $0..(n-1)$ in a ring. For this program, n is required to be an odd number. Unlike the asymmetric program in Section, 6.3.4.1, in symmetric programs, all processes execute an identical code and cannot use their ID. In other words, the process IDs are only for understanding the program and not used by processes themselves.

6.3.4.2.1 Scenario 1: Program Sym_{orig} We denote the symmetric probabilistic program as Sym_{orig}, and the actions are as follows:

$$
\begin{aligned}
A_j\text{:}\ \ x.j == x.pre \quad &\longrightarrow \quad \lambda \quad :: x.j := 0 \\
&\quad\ +1-\lambda :: x.j := 1; \\
A_j\text{:}\ \ x.j \neq x.pre \quad &\longrightarrow \quad 1 \quad :: x.j := x.pre
\end{aligned}
$$

where pre denotes the predecessor that comes before process j in the ring. Observe that in the above program, process j is privileged (respectively, process j has the token) iff its x value is equal to its predecessor's. In this case, it randomly chooses to update its value. In particular, with probability λ, it sets x value to 0, with probability $1-\lambda$, it sets it to 1.

Since the number of processes are odd, it follows that the x values of at least two neighboring processes are equal. In other words, there is at least one process holding the token. Now, consider the case where we have five processes and the x values of processes are $\langle 0,1,0,1,0\rangle$. In this state, process 0 has the token, process 1 will have the token after the execution of one program step.

6.3.4.2.2 Scenario 2: Program Sym_{flip}: Flip $x.j$ value In the program Sym_{orig}, if a process has the token, the process probabilistically chooses 0 or 1 by tossing a coin. An informal example given to describe this program from [18] is as follows:

> Imagine seven boys, seated in a circle, each with a coin laying at on one hand. In unison, all boys do the following. Each boy looks at the face of his own coin and that of the boy to his left in the circle; if the two coins show differing faces (head and tail) then he will turn his coin over; otherwise he will toss his coin to obtain a random face. This unison step is repeated ad infinitum. Regardless of the initial faces of the coins, after a finite number of steps (with probability one) only one boy tosses a coin in each step.

In the above program, if x value of process j equals that of $x.(j-1)$ then the new value of $x.j$ does not depend upon the old value of $x.j$. Next, we consider the following variation: When process j tosses a coin then if the coin toss returns *head*, it keeps the old value. On the other hand, if it returns *tail*, it flips the value. In other words, the revised program Sym_{flip} is as follows:

$$
\begin{aligned}
A_j\text{:}\ \ x.j == x.pre \quad &\longrightarrow \quad \lambda \quad :: x.j := x.j \\
&\quad +1-\lambda :: x.j := 1 - x.j; \\
A_j\text{:}\ \ x.j \neq x.pre \quad &\longrightarrow \quad 1 \quad :: x.j := x.pre
\end{aligned}
$$

Thus, Sym_{flip} can be described by the following informal example:

> Imagine seven boys, seated in a circle, each with a coin laying at on one hand. In unison, all boys do the following. Each boy looks at the face of his own coin and that of the boy to his left in the circle; if the two coins show differing faces (head and tail) then he will turn his coin over; **otherwise he will flip his coin with probability** $1-\lambda$**.** This unison step is repeated ad infinitum. Regardless of the initial faces of the coins, after a finite number of steps (with probability one) only one boy is eligible to flip his coin in each step.

6.3.5 Experimental Setup

For the scenario where there is a single objective, we use GA. For the scenario where there are multiple objectives, we use NSGA-II. In all instances, we use a population size of 60. All experiments use simulated recombination operator [43] with $p_c = 0.8$ and index $\eta_c = 10$, and polynomial mutation operator [26] with $p_m = 0.5$ and index $\eta_m = 20$. These are the commonly used values. We set the maximum number of generations to 500. However, GA is stopped when the best solutions survive more than 30 generations, as it is anticipated that further evolution will not lead to a better

program. We run each experiment several times and utilize the best programs identified in all those runs.

6.3.6 Analysis of Asymmetric Probabilistic Self-stabilizing Program

In this section, we describe our experimental results analyzing the different programs in Sections 6.3.4.1. We optimize two objectives - closure and convergence for three different variations (scenarios) of asymmetric probabilistic self-stabilizing program: $Asym_{incP}$, $Asym_{incNN}$ and $Asym_{rand}$. For closure, we measure the token circulation time that the time needed for a token circulates a ring from a legitimate state. For convergence, we measure the average stabilization time that is the time needed for converging to the legitimate states from arbitrary states.

6.3.6.1 Scenario 1: Optimization of $Asym_{incP}$: $Asym_{incP_1}$ and $Asym_{incP_2}$

For $Asym_{incP}$, the token circulation time is fixed to be N. This is due to the fact that every action execution always passes the token. Hence, we only consider optimization of convergence time for $Asym_{incP}$. Specifically, we consider two corresponding programs : $Asym_{incP_1}$ and $Asym_{incP_2}$.

For each variation, we consider different sizes of the ring. The goal of GA is to generate the best programs for each ring, which achieve the best performance in convergence, by evolving the probabilities.

Tables 6.1 and 6.2 show the final probabilities found by GA for the two variations respectively. In Table 6.1, for larger rings ($N \geq 6$), the best λ found by GA is 0.5. Thus, for $Asym_{incP_1}$ program, executing two actions with equal probability is most preferable when the size of ring is greater than or equal to 6. However, for a ring with 5 or less processes, biasing towards one action reduces the convergence time. We note that for the case where we have 4 processes, the best value of λ identified in Table 6.1 is approximately 0.22. GA also found another solution where λ is

around 0.78. This is expected due to symmetry of the solution. In all experiments performed in this section, GA found such symmetric solutions as well. However, for brevity, we only mention one of them.

Table 6.1: Optimal Probabilities for $Asym_{incP_1}$

N	λ	$1-\lambda$
4	0.223833	0.776167
5	0.404007	0.595993
6	0.500000	0.500000
7	0.500000	0.500000
8	0.500000	0.500000
9	0.500000	0.500000
10	0.500000	0.500000
11	0.500000	0.500000
12	0.500000	0.500000
13	0.500000	0.500000

Table 6.2: Optimal Probabilities for $Asym_{incP_2}$

N	λ_1	λ_2	$1-\lambda_1-\lambda_2$
4	0.000001	0.000001	0.999998
5	0.000001	0.1032	0.896799
6	0.000001	0.15721	0.842789
7	0.000001	0.20629	0.793709
8	0.013446	0.24638	0.740174
9	0.087323	0.27439	0.638287
10	0.243172	0.32577	0.431058
11	0.333333	0.333334	0.333333
12	0.333333	0.333334	0.333333

In Table 6.2, for $N < 8$, the best λ_1 is the minimum value 0.000001, whereas $1-\lambda_1-\lambda_2$ is very large from 0.793709 to 0.999998. In this program, the λ_1 and $1-\lambda_1-\lambda_2$ are symmetric, that is the performance of program with $\lambda_1 = 0.1, \lambda_2 = 0.5, 1-\lambda_1-\lambda_2 = 0.4$ is same with the one with $\lambda_1 = 0.4, \lambda_2 = 0.5, 1-\lambda_1-\lambda_2 = 0.1$. For brevity, we only discuss one of the symmetric solutions. In Table 6.2, the solutions that reduce the expected convergence time heavily prefer the third action (where $x.0$ is increased by 3) over the first two actions. As N increases, the three probabilities λ_1, λ_2 and $1-\lambda_1-\lambda_2$ are balanced. Also, the best value for λ_2 also increases when the size of ring increases. In both results in Tables 6.1 and 6.2, we observe that equal probability is preferred as the size of the ring increases, and unequal probabilities are preferred for smaller rings. Also, the size of the ring for which equal probabilities are preferred increases with the size of domain of x.

To analyze performance of GA, in Figure 6.3, we plot the variation of population's average objective value with the generation number for the ring $N = 10$. The figure shows that both cases

reach the minimum value very quickly at generation 5. Once the minimum is found, the whole population converges to it very quickly.

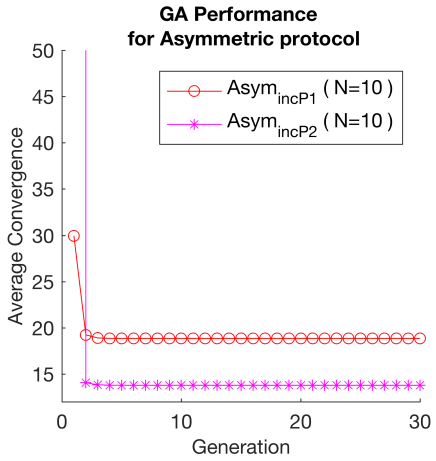

Figure 6.3: Variation of Average Objective with Generation Number for $Asym_{incP}$

6.3.6.2 Scenario 2: Optimization of $Asym_{incNN}$

This section discusses the experimental results of $Asym_{incNN}$ program. In $Asym_{incNN}$ program, $x.0$ either remains the same with the probability λ_1 and increases by 1 or 2 with the probability of λ_2 and $1 - \lambda_1 - \lambda_2$ respectively. We optimize both closure and convergence, and Figure 6.4 shows non-dominated solutions found by GA for each ring.

These solutions provide different trade-off between two objectives. Specifically, there is no single solution performing best in both objectives. In particular, convergence time decrease as the token circulation time increases. The convergence time decreases rapidly when the token circulation time increases upto 10. Subsequently, as the token circulation time increases, convergence time decreases slowly. As an illustration, consider the ring size of 7, the least token circulation time is 7.000001, however, convergence time for this solution is 9.68149. By contrast, the least convergence time is 4.01, however, the token circulation time for this solutions is more than $3 * 10^5$. Both these two extremes are not desirable. Instead, we could choose a solution with token cir-

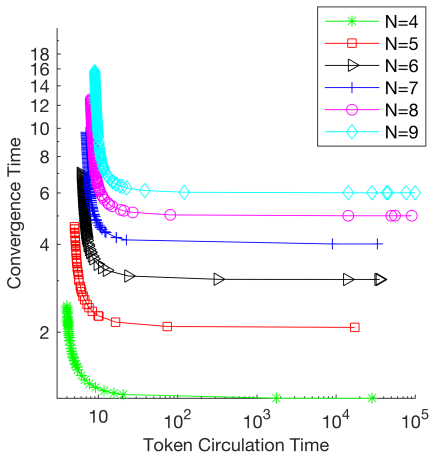

Figure 6.4: Non-dominated Solutions for $Asym_{incNN}$

Table 6.3: Evolved Probabilities for $Asym_{incNN}$ Performing Best on Token Circulation Time

N	λ^1	λ^2	Token Circulation Time
4	0.000001	0.223948	4.000001
5	0.000001	0.403456	5.000001
6	0.000001	0.5	6.000001
7	0.000001	0.5	7.000001
8	0.000001	0.5	8.000001
9	0.000001	0.5	9.000001

Table 6.4: Evolved Probabilities for $Asym_{incNN}$ Performing Best on Convergence

N	λ_1	λ_2	Convergence Time
4	0.999998	0.000001	1.185186
5	0.999998	0.000001	2.074076
6	0.999998	0.000001	3.028810
7	0.999998	0.000001	4.010978
8	0.999998	0.000001	5.004120
9	0.999998	0.000001	6.001530

culation time 7.61290 (increase by 8.76% compared to optimal) where the convergence time is 6.5747 (increased by 63.96% compared to optimal). Or we could choose a solution with token circulation time 8.840917 (increase by 26.3% compared to optimal) where the convergence time is 5.06996 (increased by 26.43% compared to optimal). Among the non-dominated solutions, a user can identify a suitable trade-off.

Table 6.3 shows the evolved probabilities of non-dominated solutions that have best performance respect to token circulation time, and Table 6.4 shows the ones that have the best performance respect to convergence time. It is clear that when λ_1 is close to zero, the program performs best in terms of token circulation time. As λ_1 increases, the token circulation time increases, and

when λ_1 is almost equal to 1, the program has maximum token circulation time. This is anticipated, within the legitimate states, as λ_1 approaches 0, process 0 passes the token almost as soon as it gets one. This is desirable for token circulation. However, it potentially increases the convergence time. By contrast, when λ_1 is high, it takes several steps for process 0 to forward the token to process 1. Within these steps, other processes can stabilize. This helps with the convergence of the processes. Another observation is when λ_1 is the smallest value permitted in our experiments, as λ_2 varies, time for token circulation is unchanged, and GA finds the one with the best convergence time.

From the Table 6.4, we find that when λ_1 is close to 1, the program provides best convergence time. As λ_1 decreases, the convergence time increases, and when λ_1 is almost 0 the convergence time is very high and the token circulation time reaches minimum. The decision maker needs to identify suitable trade-off points so that both convergence time and token circulation time are reasonable. To assist decision-maker, we consider one criteria below.

6.3.6.2.1 Identifying solutions based on analysis on mean time between failures (MTBF)

Since $Asym_{incNN}$ provides trade-off between token circulation time and convergence time, we identify one criteria to choose the value of desired probabilities. Our analysis is based on the overall goal of maximizing the number of token circulations after recovery from a failure. If the token circulation time is high, for a given time the number of token circulation is reduced. Also, if the convergence time is high, it means that the corresponding time is wasted as far as token circulation is concerned. To identify this trade-off, we consider the effect of mean-time-between failures (MTBF), which is often used in measuring the reliability of the system.

For a given MTBF, we analyze evolved solutions by their performance of recovery from a failure and the number of token circulations after recovery. We consider 5 different systems with

MTBF of [10, 50, 500, 1000, 5000] steps. Figures 6.5 and 6.6 mark the solution with the maximum number of token circulation for a given MTBF. Next to the marked solution, the annotation MTBF$-i$ refers to the case MTBF$= i$ steps. The percentage given for each marked solution is computed as follows: within the maximum MTBF (5000 steps), the ratio of the total number of token circulation with a specific MTBF to the total number of token circulation with failure-free system. The higher the percentage, the more efficient the solution is.

For the ring size $N = 4$ and $N = 5$, the best solutions for MTBF> 10 are the same - the solutions with the least token circulation time, the best solution for MTBF$= 10$ is different. This is due to the fact that higher failure rate requires more frequent recoveries and the solution with less convergence time is preferred. For the ring size $N = 6$, the best solutions for all MTBF values are the same. For the large rings $N > 6$, none of the solutions finish the token circulation within 10 steps. Hence, for $N > 6$, we only consider the case where MTBF is more than 10. For the ring size $N > 6$, the best solution for the system with MTBF$= 50$ is the distinct, and the best solution with MTBF> 50 is the same. For the same MTBF, the percentage increases when the ring size increases.

Among the evolved solutions, for a specific MTBF, the solution that provides the smaller convergence time causes an increase in time for the token circulation. For instance, consider the ring size $N = 9$, for 5000 steps and assume MTBF$= 50$. In this case, the solution that provides least convergence time is able to circulate the token zero times , the solution that provides least token circulation time is able to circulate the token 200 times, and the solution that considers both properties is able to circulate the token as much as 300 times.

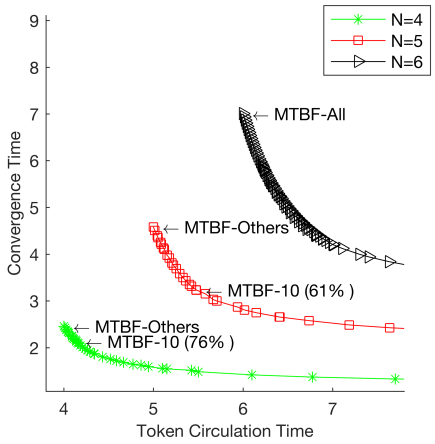

Figure 6.5: MTBF Performance for $Asym_{incNN}$ of $N \leq 6$

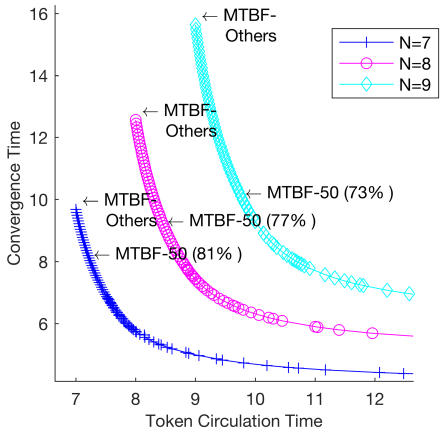

Figure 6.6: MTBF Performance for $Asym_{incNN}$ of $N > 6$

Table 6.5 shows the evolved probabilities of the marked solutions in the above figures that has the best performance for a given MTBF value. It shows that for the large value of MTBF the best solution is the one with best performance in token circulation time (see Table 6.3). However, for the smaller MTBF, since the solution with best performance in token circulation time has large convergence time, they do not have enough time to circulate the token many times.

Table 6.5: Optimal Probabilities for Different MTBF

N	$MTBF$	λ_1	λ_2
4	10	0.122123	0.064721
4	>10	0.000001	0.223948
5	10	0.324489	0.000083
5	>10	0.000001	0.40345
6	All	0.000001	0.500000
7	50	0.16	0.500000
7	>50	0.000001	0.500000
8	50	0.3	0.500000
8	>50	0.000001	0.500000
9	50	0.42	0.500000
9	>50	0.000001	0.500000

6.3.6.3 Scenario 3: Optimization of $Asym_{rand}$

This section discusses the experimental results of $Asym_{rand}$ program. In $Asym_{rand}$ program, $x.0$ is randomly assigned to $0, 1$ or 2 with the probabilities $\lambda_1, \lambda_2, 1 - \lambda_1 - \lambda_2$ respectively. Figure 6.7 shows non-dominated solutions found by GA in each ring.

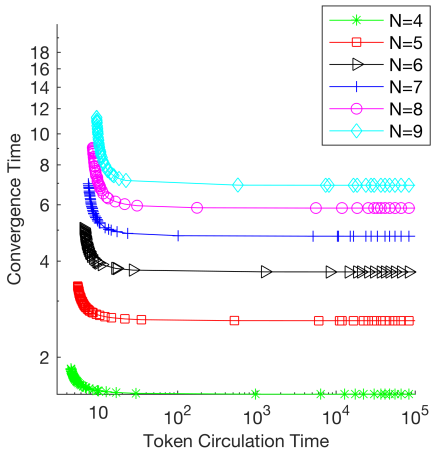

Figure 6.7: Non-dominated Solutions for $Asym_{rand}$

These solutions provide different trade-offs between token circulation time and convergence time. The convergence time decreases rapidly when token circulation time increases from 4 to 13, and after token circulation time increases beyond 13, convergence time decreases slowly. For instance, consider the ring size of 8, the solution with least token circulation time (8.500001) has highest convergence time (9.035), and the solution with least convergence time (5.85095) has the highest token circulation time more than $1.6 * 10^5$. By compromising token circulation time by only 10% from the minimum, we could get the median convergence time around 7.40568. Also, in order to get closer to the minimum convergence time, the token circulation time should increase at least by 200%.

Tables 6.6 shows the evolved probabilities of non-dominated solutions that have best performance with respect to token circulation time, and Table 6.7 shows the ones that have the best

performance with respect to convergence time. From these results, when all probabilities are approximately equal to $\frac{1}{3}$, the program performs best in terms of token circulation time. As both λ_1 and λ_2 decrease, the token circulation time increases. When λ_1 and λ_2 are close to 0, the program performs best in terms of convergence time. This means when the process 0 keeps the same deterministic value the program converges fast, however, in this case, time for token circulation is very high.

Table 6.6: Evolved Probabilities for $Asym_{rand}$ Performing Best on Convergence

N	λ^1	λ^2	Convergence Time
4	0.000001	0.000001	1.530866
5	0.000001	0.000001	2.600826
6	0.000001	0.000001	3.698222
7	0.000001	0.000001	4.784644
8	0.000001	0.000001	5.850949
9	0.000001	0.000001	6.898557

Table 6.7: Evolved Probabilities for $Asym_{rand}$ Performing Best on Token Circulation Time

N	λ^1	λ^2	Token Circulation Time
4	0.333557	0.333523	4.50
5	0.333204	0.333230	5.50
6	0.333255	0.334148	6.500001
7	0.334266	0.333581	7.500003
8	0.333010	0.333089	8.500001
9	0.331886	0.334217	9.500004

6.3.6.3.1 Identifying best solutions based on analysis on MTBF Similar to $Asym_{incNN}$, we evaluate evolved non-dominated solutions for 5 different values of MTBF. In Figure 6.8, we mark the solution with the maximum number of token circulations. In contrast to $Asym_{incNN}$, for all ring sizes, the best solutions for different MTBF values are the same, and it is the one with the least token circulation time. Also, comparing Figure 6.8 with Figures 6.5 and 6.6, the curves in Figure 6.8 are less *steep*, i.e., the convergence time decreases slowly when token circulation time increases.

Table 6.8 shows the evolved probabilities of the marked solutions in Figure 6.8. It shows that for all different values of MTBF the best solution is the one with best performance in token circulation time.

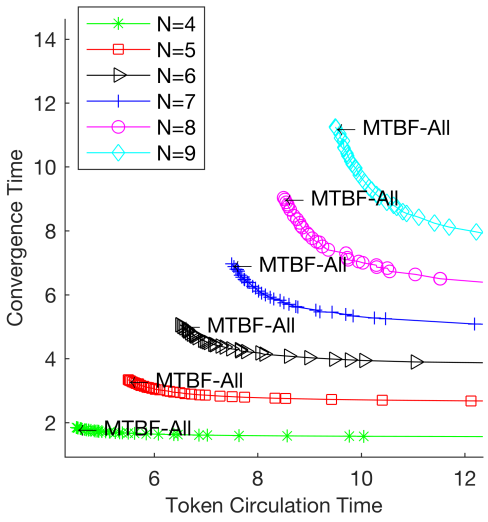

Figure 6.8: MTBF Performance for $Asym_{incNN}$

Table 6.8: Optimal Probabilities for Different MTBF

N	$MTBF$	λ_1	λ_2
4	All	0.333557	0.333523
5	All	0.333204	0.333230
6	All	0.333255	0.334148
7	All	0.334266	0.333581
8	All	0.333010	0.333089
9	All	0.331886	0.334217

6.3.7 Analysis of Symmetric Probabilistic Self-stabilizing Program

This section presents experimental results for identifying the best probabilities evolved by GA for two symmetric probabilistic self-stabilizing token ring programs: Sym_{orig} and Sym_{flip} described in Section 6.3.4.2. Similar to the previous section, we apply NSGA-II to minimize the token circulation time and average convergence time.

6.3.7.1 Scenario 1: Optimization of Sym_{orig}

For the program Sym_{orig}, we consider the case where the ring size is odd number varying from 3 to 13. Figure 6.9 shows non-dominated solutions found by NSGA-II for different ring sizes. In the figure, the knee of the curve is *steep* and, a little decrease in token circulation time increases the convergence time substantially.

Table 6.9 shows the evolved probabilities of non-dominated solutions that have best performance with respect to token circulation time, and Table 6.10 shows the ones that have the best performance respect to convergence time. Similar to the results from previous sections, as token

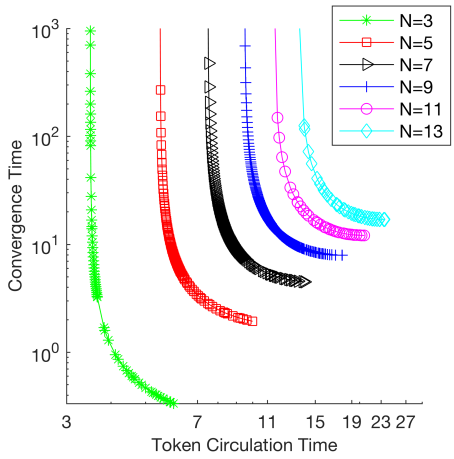

Figure 6.9: Non-dominated Solutions for Sym_{orig}

Table 6.9: Evolved Probabilities for Sym_{orig} Performing Best on Token Circulation Time

N	λ	Token Circulation Time
3	0.000001	3.50000
5	0.000001	5.50000
7	0.000001	7.50001
9	0.000001	9.50001
11	0.000001	11.5000
13	0.000001	13.5000

Table 6.10: Evolved Probabilities for Sym_{orig} Performing Best on Convergence

N	λ	Convergence Time
3	0.499951	0.33333
5	0.500079	1.93333
7	0.500069	4.49331
9	0.542012	7.92099
11	0.63553	12.10203
13	0.669122	16.94901

circulation time decreases, the convergence time increases. For both token circulation time and convergence time, λ and $1-\lambda$ are symmetric. From these two tables, when λ is very close to any two extremes (either 0 or 1), the token circulation time is the minimum, since by holding the same static value, the processes pass the token almost deterministically at most in two steps from any legitimate state. For convergence time, we observe that for smaller size of ring $N < 9$, the best λ found by GA is around 0.5, and for $N \geq 9$, biased to one of the actions is desirable. Moreover, as the ring size increases, the bias increases as well.

6.3.7.1.1 Identifying best solutions based on analysis on MTBF We evaluate the evolved non-dominated solutions on 5 different systems with MTBF of [10, 50, 500, 1000, 5000] steps.

In figures 6.10 - 6.15, we mark the solution with the maximum number of token circulations for different rings $N = 3, 5, 7, 9, 11, 13$.

For different MTBF value, the best solutions are different, the token circulation time of the ideal solution increases when the MTBF decreases. For instance, consider the ring size $N = 5$, and total 5000 steps, when MTBF$= 5000$, the ideal solution has small token circulation time 5.5635 and large convergence time 56.999. As MTBF decreases to 1000, the token circulation time increases to 5.645, and convergence time decreases to 25.5478. When MTBF$= 50$, the token circulation time increases to 6.2078, and convergence time decreases to 6.0169. Similar observation is held for other rings($N \neq 5$).

For the solutions with less/least token circulation time, they have large convergence time and take more time to recover. These solutions are not favorable when the failure rate is high (small MTBF values), since there is not much time left after the recovery. However, for large MTBF values (low failure rate), the solutions with smaller token circulation time perform better.

Similar to the $Asym_{incNN}$, for the same MTBF, the percentage increases when the ring size increases. For brevity, we omit the detailed probabilities value of all marked solutions for different MTBF in this and following sections.

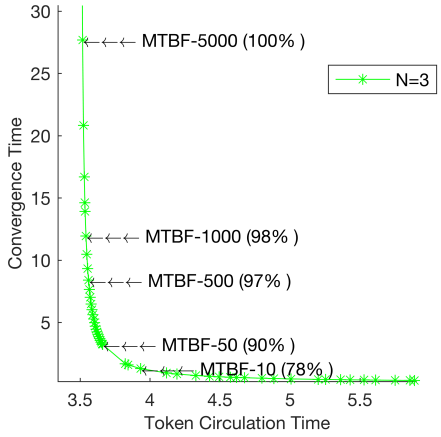

Figure 6.10: MTBF Performance for Sym_{orig} of $N = 3$

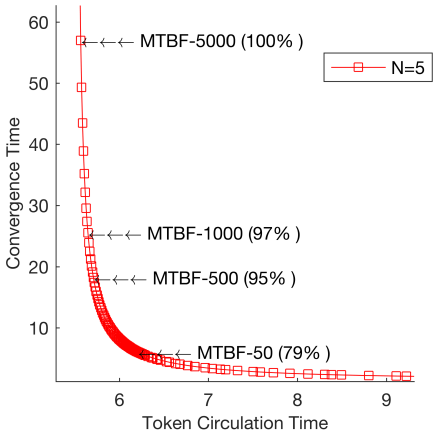

Figure 6.11: MTBF Performance for Sym_{orig} of $N = 5$

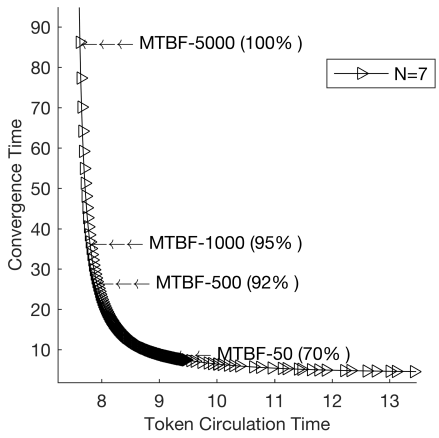

Figure 6.12: MTBF Performance for Sym_{orig} of $N = 7$

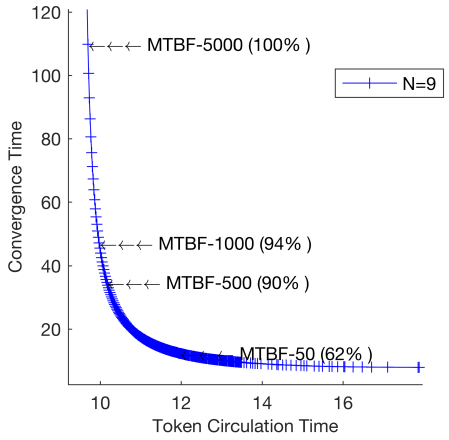

Figure 6.13: MTBF Performance for Sym_{orig} of $N = 9$

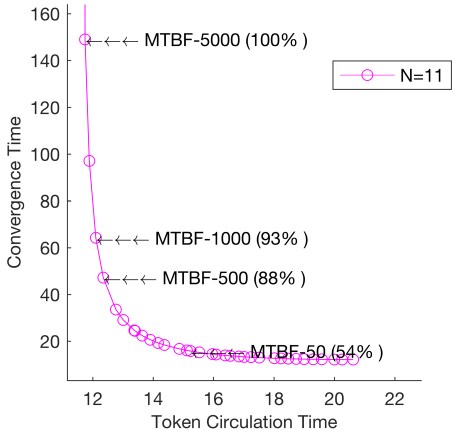

Figure 6.14: MTBF Performance for Sym_{orig} of $N = 11$

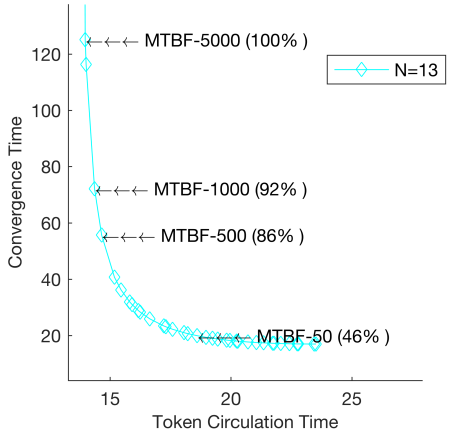

Figure 6.15: MTBF Performance foor Sym_{orig} of $N = 13$

6.3.7.2 Scenario 2: Optimization of Sym_{flip}

For the program Sym_{flip}, we also consider ring sizes to be odd numbers between 3 and 13. Figure 6.16 shows non-dominated solutions evolved by GA. These non-dominated solutions also provide trade-offs between token circulation time and convergence time. Similar to Sym_{orig}, the knee of the curves are also steep.

Table 6.11 shows the evolved probabilities of non-dominated solutions that have best performance respect to token circulation time, and Table 6.12 shows the ones that have the best perfor-

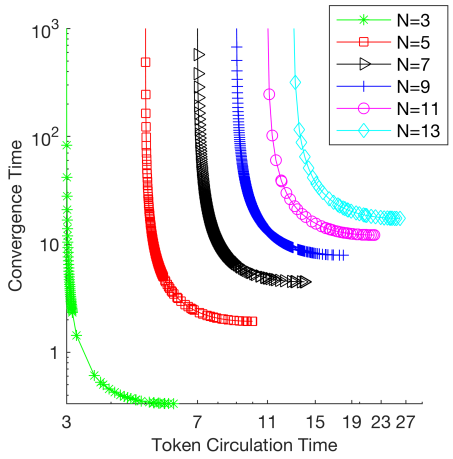

Figure 6.16: Non-dominated Solutions for Sym_{flip}

Table 6.11: Evolved Probabilities for Sym_{flip} Performing Best on Token Circulation Time

N	λ	Token Circulation Time
3	0.999999	3.000003
5	0.999999	5.000005
7	0.999999	7.000007
9	0.999999	9.000009
11	0.999999	11.00001
13	0.999999	13.00001

Table 6.12: Evolved Probabilities for Sym_{flip} Performing Best on Convergence

N	λ	Convergence Time
3	0.499777	0.33333
5	0.500153	1.93333
7	0.499887	4.49331
9	0.499755	7.92156
11	0.499819	12.2059
13	0.500107	17.346

mance respect to convergence time. For convergence time, λ and $1-\lambda$ are symmetric. However, for the token circulation time, λ increases when token circulation decreases. When the probability is close 1, the program has minimum token circulation time. This is due to the fact that by holding the same static value, the processes toss the token almost deterministically from any legitimate state. When the probability is close to 0.5, the evolved programs have the best convergence time. Unlike the Sym_{orig}, the best probability for Sym_{flip} in terms of convergence is the same for different rings.

6.3.7.2.1 Identifying best solutions based on analysis on MTBF Similar to Sym_{orig} we test evolved non-dominated solutions on 5 different systems with MTBF of [10, 50, 500, 1000, 5000]

steps. Figures 6.17 - 6.22 mark the solution with the maximum number of token circulations for the ring size $N = 3, 5, 7, 9, 11, 13$ respectively. Similar to Sym_{orig}, each MTBF has its own best solution, and the token circulation time of the best solution increases when the MTBF decreases. For large MTBF, the programs with smaller token circulation time perform better, and for the small MTBF, solutions with small convergence time are preferred. Meanwhile, the smaller the MTBF time, the smaller the percentage.

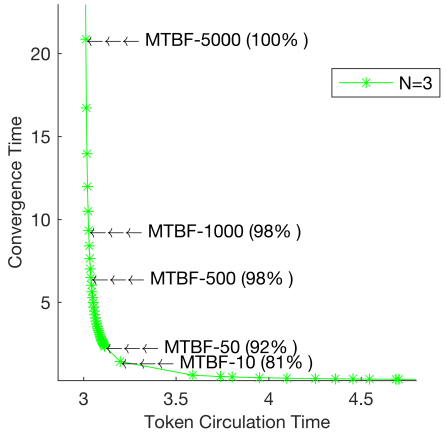

Figure 6.17: MTBF Performance for Sym_{flip} of $N = 3$

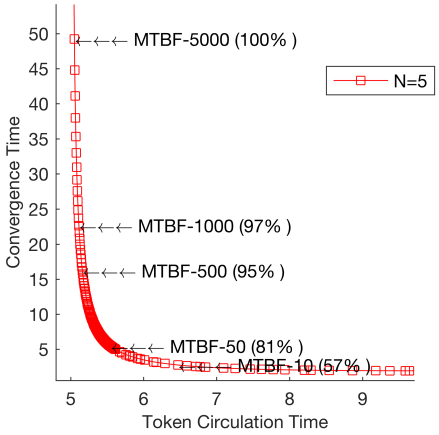

Figure 6.18: MTBF Performance for Sym_{flip} of $N = 5$

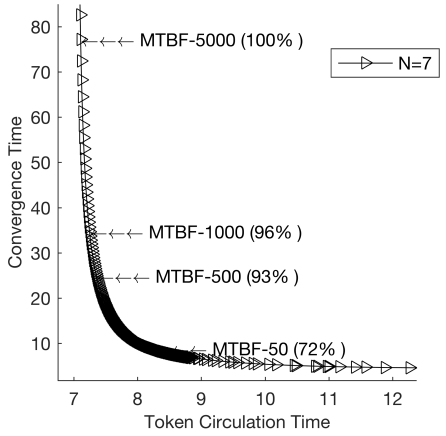

Figure 6.19: MTBF Performance for Sym_{flip} of $N = 7$

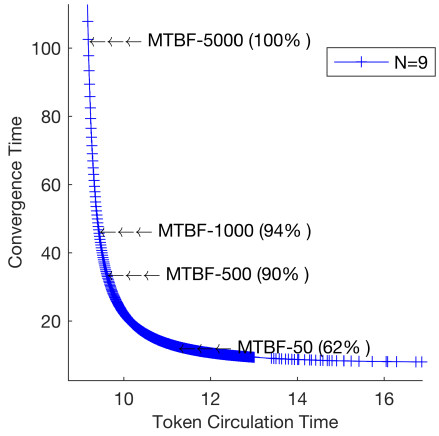

Figure 6.20: MTBF Performance for Sym_{flip} of $N = 9$

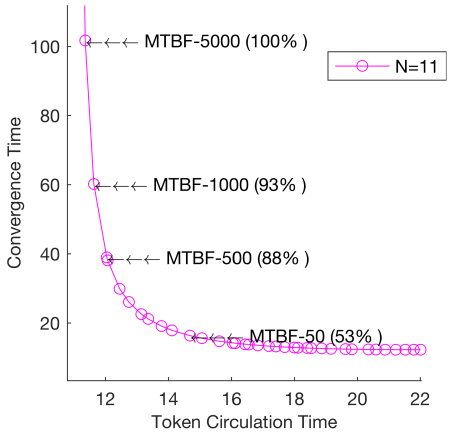

Figure 6.21: MTBF Performance for Sym_{flip} of $N = 11$

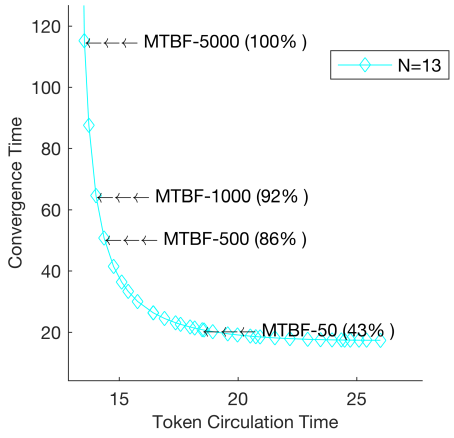

Figure 6.22: MTBF Performance for Sym_{flip} of $N = 13$

6.3.8 Analysis of Symmetric Token Ring Protocols Using Asymmetric Probabilities

In this section, we describe our experimental results for analyzing the symmetric protocol using asymmetric probabilities. For the previous two symmetric token ring protocols Sym_{orig} and Sym_{flip}, the values of λ used by all processes are identical. This captures the intuition of [18] that the processes are anonymous and cannot use their process IDs. In this section, we consider the effect of relaxing this so that each process chooses its λ value independently. We denote λ for process j as λ_j. We consider ring size of 3, 5 and 7. We optimize both token circulation time and convergence using NSGA-II. The non-dominated solutions identified by GA are shown in Figure 6.23. Again, evolved solutions provide different trade-offs between token circulation time and convergence in each ring.

Table 6.13 shows the evolved probabilities of non-dominated solutions that have best performance respect to token circulation time, and Table 6.14 shows the ones that have the best performance respect to convergence time. From the table we could find, for 3-process ring, making one of the processes to be (almost) deterministic is ideal for reducing the convergence time. For 5 pro-

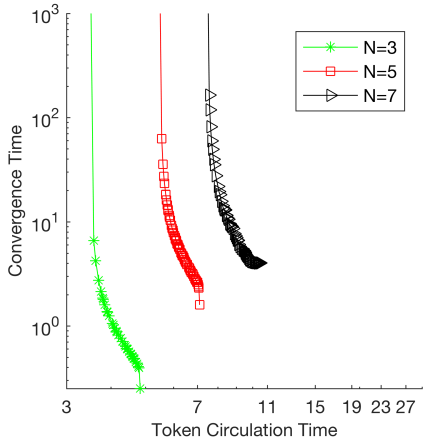

Figure 6.23: Non-dominated Solutions for Sym_{orig} Using Asymmetric Probabilities

Table 6.13: Evolved Probabilities of Sym_{orig} Using Asymmetric Probabilities Performing Best on token circulation time

N	λ^1	λ^2	λ^3	λ^4	λ^5
3	0.000001	0.000001	0.000001		
3	0.999999	0.999999	0.999999		
5	0.000001	0.000001	0.000001	0.000001	0.000001
5	0.999999	0.999999	0.999999	0.999999	0.999999

cesses, when the probability of two consecutive processes are equal to the extreme value(minumum or maximum), the programs perform best in convergence. Obviously, when all the processes have the same extreme probability value(almost deterministically chosing the same value), the program provides the least token circulation time.

Table 6.14: Evolved Probabilities of Sym_{orig} Using Asymmetric Probabilities Performing Best on Convergence

N	λ^1	λ^2	λ^3	λ^4	λ^5
3	0.000001	0.000001	0.999999		
3	0.000001	0.999999	0.000001		
3	0.999999	0.000001	0.000001		
5	0.000001	0.999999	0.999999	0.999999	0.000001
5	0.000001	0.000001	0.999999	0.999999	0.999999
5	0.999999	0.000001	0.000001	0.999999	0.999999
5	0.999999	0.999999	0.000001	0.000001	0.999999
5	0.999999	0.999999	0.999999	0.000001	0.000001

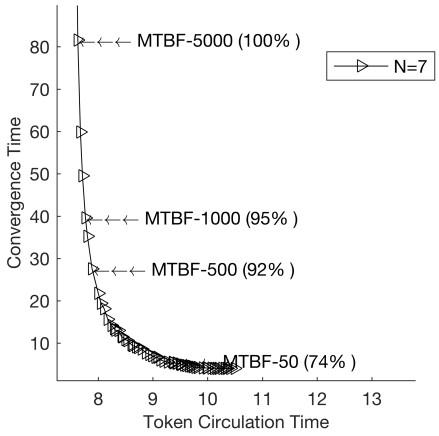

Figure 6.26: MTBF Performance for Sym_{orig} using asymmetric probabilities of $N = 7$

6.3.8.1 Identifying best solutions Based on Analysis on MTBF

We evaluated evolved non-dominated solutions on 5 different systems with MTBF of [10, 50, 500, 1000, 5000] steps. In Figures 6.24- 6.26, we identify the solution with the maximum number of token circulations for the ring size $N = 3, 5, 7$. Similar to Sym_{orig} and Sym_{flip}, each MTBF has its own best solution, and the token circulation time of the best solutions increases when the MTBF decreases, and the smaller the MTBF time, the smaller the percentage. For the similar detailed analysis, please see Section 6.3.7.1

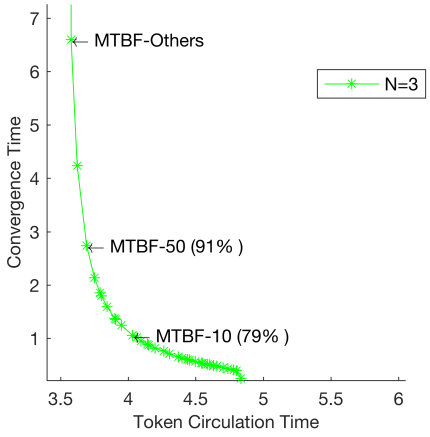

Figure 6.24: MTBF Performance for Sym_{orig} using asymmetric probabilities of $N = 3$

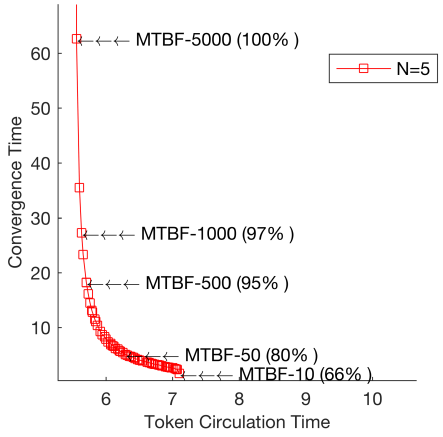

Figure 6.25: MTBF Performance for Sym_{orig} using asymmetric probabilities of $N = 5$

Chapter 7

Related Literature

This section discusses existing approaches related to this dissertation. The existing studies can be divided into two areas: optimization and problem modeling (synthesis). The former can be further classified into two parts: optimization of multi-scenario multi-objective engineering problem and optimization of fault-tolerant program. As far as we are concerned, there are few studies optimizing fault-tolerant problems in a multi-scenario, multi-objective manner. In the next sections, we discuss existing studies of both parts.

7.1 Optimization of Multi-scenario Multi-objective Problem

Many engineering design problems are required to consider various scenarios, optimal structural shapes for truss structures design, optimization of aircraft components or aerospace structures, and vehicles design, etc.

In the structural optimization, most studies dealing with multiple scenarios consider the existence of different scenarios as multiple loading conditions [44, 45, 46, 47, 48]. [49] proposes a formulation of shape optimization of structures for multiple loading conditions using a homogenization method. Authors uses a weighted sum to aggregate all loading conditions. Instead of treating loading conditions and other constraints deterministically, robust active research field called robust optimization treats these uncertainties theoretically. [47] focuses on topology optimization under loading uncertainties in multi-loading formulation consisting of various load patterns including

probability function. [48] formulates design problem to minimize a robust compliance induced by the worst load case of an uncertain load sets.

In other engineering areas, multiple loading conditions or scenarios are considered in the optimization. In vehicle design, [6] focuses on the optimization of groups of vehicle performance indices (criteria) in different operating scenarios. [50] performs a multi-scenario multi-objective optimization in the area of power management. Authors construct a multi-objective problem by applying a weighted-sum of all objectives for a scenario and each scenario is then considered as an objective value. In the area of System-On-Chip (SOC) applications design, [51] aggregates scenarios and considers each objective as an criteria. However, only a geometric mean of all possible scenarios is suggested as an aggregate function.

Most of the above studies focus on the application to optimization problems involving multiple loading conditions or scenarios. Only a few recent studies [8, 7, 1] systematically solve a multi-scenario optimization problem. In [8], authors propose a scenario-oriented approach in which an individual trade-off objective set ($\mathcal{F}^{(k)}$) is first found for each scenario (say, k-th one) by optimizing all M objective functions, as given below:

$$\begin{aligned} &\text{Minimize} \quad \left\{ f_1^{(k)}(\mathbf{x}), f_2^{(k)}(\mathbf{x}), \ldots, f_M^{(k)}(\mathbf{x}) \right\}, \\ &\text{Subject to} \quad g_j^{(k)}(\mathbf{x}) \leq 0, \quad j = 1, 2, ..., J^k, \; k = 1, 2, \ldots, K. \end{aligned} \tag{7.1}$$

Then, every pair of trade-off sets (say, $\mathcal{F}^{(k)}$ and $\mathcal{F}^{(l)}$) are considered and the shortest normalized Euclidean distance of every solution in one trade-off set from all solutions of the other set is computed. The normalized distances are then computed from second trade-off set from former set. Then the pair of solutions from each trade-off set which is closest to a pre-specified threshold value is identified. By this process, the authors attempted to locate a pair of solutions that are a specific

distance from each other in the objective space. Thereafter, from the pair, the one which is furthest from the objective-wise extreme solutions is declared as the final solution to the multi-scenario, multi-objective optimization problem.

One criticism of the above approach is that it completely eliminates the decision-making procedure which is often followed in the solution process of a multi-objective optimization problem. Authors have integrated a pre-defined decision-making process which may not be agreeable to decision-makers, in general. Moreover, this philosophy does not allow decision-makers to have any understanding of trade-off solutions to facilitate them to choose a preferred solution in an informed manner [26]. Although, the decision-making through preference information takes place only in one of the trade-off efficient fronts for a particular scenario, for other trade-off frontiers, the decision-making is geometry based, which may be highly questionable in practice.

Realizing the importance of decision-making, in their subsequent approaches [7, 1], authors suggest a more sophisticated method. They suggest to solve first the optimization problem for the first scenario (say, $\mathcal{F}^{(1)}$). These solutions are then presented to the decision-makers and a single preferred solution (say, $\mathbf{x}^1$) is identified. After that, $\mathbf{x}^1$ is evaluated for all other scenarios. If $\mathbf{x}^1$ is found acceptable as a preferred solution for all other scenarios, then $\mathbf{x}^1$ is the final solution to the multi-scenario problem. Otherwise, a new optimization problem if formulated by using an ϵ-dominance concept from $\mathbf{x}^1$ to solve the second scenario problem. The ϵ-vector is specified by the user as a tolerance parameter. Once the second trade-off set ($\mathcal{F}^{(2)}$) is found, the decision maker then chooses a preferred solution (say, $\mathbf{x}^2$) from the second optimized set. This solution is then evaluated for all other scenarios including Scenario 1, which has already been considered before, and the process continues iteratively until an accepted solution is found. One of the drawbacks of this approach is that the decision maker needs to be involved many times during the optimization process, which can be too demanding for decision-makers. Also decision-makers are expected to

look at multiple trade-off solutions after each scenario problem is solved. Moreover, the outcome of the procedure largely depends on the sequence of scenarios optimized during the optimization process and the chosen tolerance parameters. It would be ideal if a new optimization procedure can be developed to consider all scenarios simultaneously and find a single trade-off solution set balancing all scenarios so that decision-makers can analyze them to choose a single preferred solution. Not only that this method is practical, it is also in tune with the single-scenario, multi-objective optimization problem solving tasks. Standard multi-criterion decision-making (MCDM) methods can then be applied to the trade-off set for choosing a preferred solution. Our proposed objective-wise integrated method (Section 4.3) is one such methodology using evolutionary multi-objective optimization algorithms.

7.2 Optimization of Fault-tolerant Programs

Self-stabilization [9, 11] is one of the fault-tolerant techniques in distributed systems. As mentioned in Section 2, the network is anonymous (symmetric) if processes do not have identity and execute the same code. A number of studies [52, 53, 54] focus on anonymous self-stabilizing ring and provide the calculus of convergence (stabilization) time - the time taken from arbitrary states to legitimate states. [53, 54] shows the expected convergence time is of order n^3, and [52] proposes an optimal protocol with convergence time of $\theta(n)$ when the memory size is known.

Among the anonymous self-stablizing programs, Herman's work [18] is most widely studied in many work [55, 56, 57, 58, 59, 60, 61]. These studies focus on expected convergence time and proposed an upper bound. [55, 59] states that expected convergence time should be minimized by the equidistant configuration.

Our technique differs from these studies, since it infers optimizing probabilities for different

variations of a randomized stabilizing program with respect to both closure and convergence. We consider each variation as a scenario, and for each variation we apply evolutionary optimization to find solutions of different trade-offs between closure and convergence. We also apply a probabilistic model checking technique (PRISM) [62] to evaluate the closure and convergence. [60] applies PRISM and exhaustively analyzes the worst-case convergence time. Instead of exhaustive search, our technique is meta-heuristic optimization, and the key insight underlying our approach is using evolution-based technique.

Similarily, [63] uses multi-objective GA and PRISM to find approximate Pareto-optimal probabilistic model sets associated with the QoS requirements of a software system. This work focuses on QoS requirements. [64] investigates the trade-offs of token circulation time and convergence time of Herman's token ring program using multi-objective GA and PRISM. However, our work considers multiple variations of both asymmetric and symmetric self-stabilizing programs, and shows many interesting observations. Moreover, we also analyze the evolved solutions based on the systems with different MTBF values. Recently, authors in [65] focus on optimizing Dijkstra's self-stabilizing program [9], and they identify trade-offs between closure and convergence properties. Our work focuses on the approach of identifying optimal probabilities values.

7.3 Modeling of Fault-tolerant Programs

There are several approaches to model programs (also known as program synthesis) [66]: formal methods, machine learning based techniques (such as genetic programming), brute-force search, version space algebra, etc.

From the perspective of formal methods, some of the related studies include program sketching [67], heuristic-based addition of fault-tolerance [68, 69], controller synthesis [70] and Game-

theoretic approaches [71]. In program sketching [67, 72], authors begin with a sketch of the program similar to ours and utilize tools such as SMT solvers to identify the conditions that are unspecified in the sketch. Literature on addition of fault-tolerance focuses on revising an existing program to add fault-tolerance with the help of heuristics. These heuristics attempt to manage the complexity of addition of fault-tolerance. However, if the heuristics fail then addition of fault-tolerance fails. By contrast, GP uses evolutionary techniques to design the fault-tolerant program and, hence, does not use heuristics, and we expect that genetic approach will be more generic in that. It does not depend on pre-defined set of heuristics but rather on the principle of evolution. Work on controller synthesis focuses on adding a controller to an existing program (plant) so that it satisfies the desired specification. Game theoretic approaches for synthesizing controllers generally utilize the model of two-player games [73].

From the evolutionary perspective, GP automatically generates programs targeted toward a particular task. The GP can be used to search large problem spaces and find approximated solutions when it is allowed. GP could be an alternative method when other conventional techniques fail. It has a certain flexibility in aspect to enumerating complex programs and providing global perspective for the search. Due to the probabilistic feature of genetic operators GP has less chance of getting stuck. Most of the previous studies focus on synthesizing the sequential program. In [74, 75], authors propose the idea of using GP to automatically repair software bugs. In [75] authors use tree-based program representation and employ test suite method to evaluate the candidate program by running on different test inputs. These studies do not consider evolving distributed programs. In [76, 77], authors propose the approach of using rule-based GP to generate distributed programs. But their work does not consider designing fault-tolerant programs and also does not apply model checking techniques. Several studies [78, 79, 80] use model checking based fitness measurement for GP, however, these studies do not consider the unpredictable environments.

One of the other search techniques for program synthesis is Brute-force, an exhaustive search method, is applied to find new programs in [81]. Compared to brute-force enumeration of programs, GP is a guided search process directed by given objectives. Version space algebra, introduced by [82], is as a search technique which discovers a function that maps attributes to the binary sets from a set of hypotheses [82]. [83] use version space algebra to synthesize repetitive robot programs and python programs.

Chapter 8

Conclusion and Future Research

We addressed an important practical aspect of applying evolutionary multi-scenario, multi-objective (MSMO) approach to modeling and optimization of fault-tolerant problems in this dissertation. In most cases, in fault-tolerant design, a solution is evaluated for various scenarios without faults, with single or multiple faults of the same or different types, under different operational conditions, and multiple problem variations, etc. We classified MSMO problems into two categories: scenario-dependent and scenario-independent problems. The former type of problem requires a solution to be optimized in all scenarios at the same time, while the latter requires a solution to be optimized in independent scenario. For the scenario-dependent problem we obtained desired solutions in one optimization run. In order to obtain well-balanced compromised solutions, it is crucial to handle multiple scenarios and multiple objectives simultaneously, particularly when some scenarios conflict. For the scenario-independent problems, we obtained one optimal solution set for each scenario, and carried out comparative analysis to gain better understanding of the problem. In this dissertation we adopted different methodologies for these two types.

8.1 Scenario-dependent MSMO problem

In scenario-dependent MSMO problem, we handled scenarios in two different approaches: objective-wise MSMO approach and scenario-wise MSMO approach.

For objective-wise approach, we suggested two systematic methods: an aggregated method and

an integrated method. The aggregated approach combines all scenarios for each objective function , and the integrated MSMO approach considers convergence and diversity of evolving population members in each scenario space and makes a careful balance between them. For integrated MSMO, we defined a new scenario-based domination principle and a scenario-based crowding distance. We then used them to modify the NSGA-II procedure. In two and three-scenario problems, we presented the results of our proposed integrated approach and compared them with an average-case, a worst-case, scenario-wise individual optimization approaches, and an existing single-point approach. Two numerical problems and a number of engineering design problems were used to demonstrate the usefulness of the integrated approach and discuss its difference with other approaches. Results showed that our methods find a widely distributed set of solutions compromised among the respective objective values under all scenarios. The previous methods that finds a single preferred solution based on a number of preferences for tolerances and intermediate solutions, lack of a diverse set of solutions at the end of the optimization procedure, and may not provide decision-makers with other trade-off solutions to make a more informed decision-making task. Instead, our proposed approaches can find solutions that are not only widely spread in all objectives, but also well compromised among different scenarios. They also provide more control to the decision-maker in setting up importance of different scenarios before and after the optimization task is performed.

For scenario-wise approach, each scenario is considered as new criteria and it converts a MSMO problem to a multi-objective problem. For this approach, we focused on an application of modeling fault-tolerant program, and evaluated the effectiveness of MSMO approach. We used genetic programming (GP) to synthesize fault-tolerant distributed programs. We adopted scenario-wise MSMO approach to consider program performance in different scenarios. We demonstrated the effectiveness of MSMO-based GP with a classic distributed problem - Byzantine agreement

problem, token ring problem and consensus problem using failure detector S. Our analysis showed that GP was more effective under a maximum-parallelism semantics (see 2.1.1) using model checking techniques for program evaluation. This suggests one way to improve effectiveness of GP in synthesizing fault-tolerant programs. Furthermore, we carried out a comparative study using single objective optimization method on Byzantine agreement program synthesis. Compared to single objective optimization method, our scenario-wise MSMO approach is able to maintain better diversity of the solutions during evolution. It also emphasizes each objective function arising from every scenario uniformly, and eventually find the desired optimum with less probability of getting stuck in the search space.

8.2 Scenario-independent MSMO problem

In scenario-independent problem, we optimized objectives in each scenario independently, and compared all the solutions to gain the deeper understanding of the problem. For a case study, we focused on optimizing a special type of fault-tolerant programs, the probabilistic self-stabilizing programs[18, 19]. Although it is well known that randomized algorithms can reduce the state space required to achieve stabilization as well as solve several problems that cannot be solved in a deterministic setting, identifying the optimal probability values for optimum results is often difficult. We evaluated these solutions in each scenario in terms of two objectives (1) convergence time and (2) time for token circulation in the absence of faults. While the protocol in [18] has been studied in [60] for analyzing convergence time, the trade-off between time for convergence and time for token circulation has not been analyzed. It also only focuses on one specific scenario(program variation). In our work, we not only validated previously known result, but also found a surprising observation. We also demonstrated that the algorithm that reduces convergence time may not be

the most suitable in practice. This is due to the fact that the time required for circulating the token is large for that algorithm. Hence, different probability values are preferred to provide the suitable trade-off.

8.3 Future Research

8.3.1 Scenario-dependent MSMO Problem

This study needs to be extended for many-objective problems and in problems having a large number of scenarios. The objective-wise integrated principle should extend to existing evolutionary many-objective optimization algorithms (such as NSGA-III or MOEA/D) with scenario-based domination and niching methods. However, it is apparent that when a large number of scenarios (say 5+ or 10+) are present, coordination among many such scenario spaces may become computationally challenging and average-case (or its more generic weighted-average cases) or worst-case optimization may be more practical, but as it has been discovered here such aggregation-based methods may not produce a nicely-balanced compromise solutions when all scenarios are not equi-dominant. Similarly, scenario-wise approach needs to systematically handle many scenarios as well.

Most of this study focused on finding the MSMO fault-tolerant solutions. Intuitively, the scenario (also worst-scenario) considering occurrence of faults is likely to provide over-design solutions for the remaining scenarios. If the worst-scenario solution is grossly over-designed for the rest of the scenarios and if particularly the specific worst-scenario is a low-probability event, designers are better off finding a new and contingent design concept to mitigate the worst scenario exclusively. We will develop contingent design scheme for mitigating worst-scenario.

8.3.2 Scenario-independent MSMO Problem

In the application of optimizing fault-tolerant problem, we used a probablisitic model checker - PRISM [84] - to analyze individual programs. Hence, one bottleneck for the use of MSMO in this manner is any bottleneck (e.g., state space explosion) associated with PRISM. One future work in this application is to develop algorithms for objective functions that provide a rough estimate of the desired property more efficiently. This will allow GA to identify almost optimal solutions quickly. One future work in this area is to identify whether this approach can improve the performance of GA. Another future work in this area is to use parallelism. Since GA provides easy opportunities for parallelism, we anticipate that this will be especially valuable for large programs.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Alexander Engau and Margaret M Wiecek. 2d decision-making for multicriteria design optimization. *Structural and Multidisciplinary Optimization*, 34(4):301–315, 2007.

[2] Sukumar Ghosh. *Distributed systems: an algorithmic approach*. CRC press, 2006.

[3] Borzoo Bonakdarpour, Sandeep S Kulkarni, and Fuad Abujarad. Symbolic synthesis of masking fault-tolerant distributed programs. *Distributed Computing*, 25(1):83–108, 2012.

[4] John R Koza. *Genetic Programming: vol. 1, On the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[5] John R Koza, Forrest H Bennett III, and Oscar Stiffelman. *Genetic programming as a Darwinian invention machine*. Springer, 1999.

[6] Georges Fadel, Imtiaz Haque, Vincent Blouin, and Margaret Wiecek. Multi-criteria multi-scenario approaches in the design of vehicles, 2005.

[7] Margaret M Wiecek, Vincent Y Blouin, Georges M Fadel, Alexander Engau, Brian J Hunt, and Vijay Singh. Multi-scenario multi-objective optimization with applications in engineering design. In *Multiobjective Programming and Goal Programming*, pages 283–298. Springer, 2009.

[8] Margaret M Wiecek, Vijay Singh, and Vincent Blouin. Multi-scenario multi-criteria optimization in engineering design. Technical report, DTIC Document, 2007.

[9] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, November 1974.

[10] Anish Arora and Mohamed Gouda. Closure and convergence: A foundation of fault-tolerant computing. *IEEE Trans. Softw. Eng.*, 19(11), November 1993.

[11] Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.

[12] Anish Arora and Mohamed Gouda. Closure and convergence: A foundation of fault-tolerant computing. *IEEE Trans. Software Engineering*, 19:1015–1027, 1993.

[13] Anish Arora and Mohamed G. Gouda. Distributed reset. *IEEE Trans. Computers*, 43(9):1026–1038, 1994.

[14] Hermann Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, Jan 2003.

[15] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[16] M. Gouda and F. Haddix. The linear alternator. *In Proceedings of the Third Workshop on Self-stabilizing Systems*, pages 31–47, 1997.

[17] M. Gouda and F. Haddix. The alternator. *In Proceedings of the Fourth Workshop on Self-stabilizing Systems*, pages 48–53, 1999.

[18] Ted Herman. Probabilistic self-stabilization. *Inf. Process. Lett.*, 35(2):63–67, 1990.

[19] Ted Herman. Self-stabilization: Randomness to reduce space. *Distributed Computing*, 6(2):95–98, 1992.

[20] Jaap-Henk Hoepman. Uniform deterministic self-stabilizing ring-orientation on odd-length rings. In *Proceedings of the 8th International Workshop on Distributed Algorithms*, WDAG '94, pages 265–279, 1994.

[21] Shing-Tsaan Huang. Leader election in uniform rings. *ACM Trans. Program. Lang. Syst.*, 15(3), July 1993.

[22] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[23] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[24] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[25] K. Deb L. Zhu and S. S. Kulkarni. Multi-scenario optimization using multi-criterion methods: A case study on byzantine agreement problem. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, July 2014.

[26] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[27] K. Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer, Boston, 1999.

[28] A. P. Wierzbicki. The use of reference objectives in multiobjective optimization. In G. Fandel and T. Gal, editors, *Multiple Criteria Decision Making Theory and Applications*, pages 468–486. Berlin: Springer-Verlag, 1980.

[29] I. Das and J.E. Dennis. Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems. *SIAM Journal of Optimization*, 8(3):631–657, 1998.

[30] Q. Zhang and H. Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *Evolutionary Computation, IEEE Transactions on*, 11(6):712–731, 2007.

[31] K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point based non-dominated sorting approach, Part I: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014.

[32] K. Deb, M. Mohan, and S. Mishra. Towards a quick computation of well-spread Pareto-optimal solutions. In *Proceedings of the Second Evolutionary Multi-Criterion Optimization (EMO-03) Conference (LNCS 2632)*, pages 222–236, 2003.

[33] K. Deb, M. Mohan, and S. Mishra. Evaluating the ϵ-domination based multi-objective evolutionary algorithm for a quick computation of Pareto-optimal solutions. *Evolutionary Computation Journal*, 13(4):501–525, 2005.

[34] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2–4):311–338, 2000.

[35] Kalyanmoy Deb and Ram B Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9(2):115–148, 1995.

[36] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.

[37] Stephen H Crandall, Norman C Dahl, and Thomas J Lardner. An introduction to the mechanics of solids, 1978. *McGrawHill International Book Company*.

[38] K. Deb and A. Kumar. Interactive evolutionary multi-objective optimization and decision-making using reference direction method. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2007)*, pages 781–788. New York: The Association of Computing Machinery (ACM), 2007.

[39] K. Deb and A. Kumar. Light beam search based multi-objective optimization using evolutionary algorithms. In *Proceedings of the Congress on Evolutionary Computation (CEC-07)*, pages 2125–2132, 2007.

[40] M. Oltean, C. Grosan, L. Diosan, and C. Mihaila. Genetic programming with linear representation: a survey. *International Journal on Artificial Intelligence Tools*, 18(2):197–238, 2009.

[41] L. Spector and A. Robinson. Genetic programming and autoconstructive evolution with the push programming language. In *Genetic Programming and Evolvable Machines*, pages 7–40, 2002.

[42] Timothy Perkis. Stack-based genetic programming. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 148–153. IEEE, 1994.

[43] Kalyanmoy Deb and Ram B Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9(3):1–15, 1994.

[44] Panos Kouvelis and Gang Yu. *Robust discrete optimization and its applications*, volume 14. Springer Science & Business Media, 2013.

[45] MC Campi and G Calafiore. Decision making in an uncertain environment: the scenario-based optimization approach. *Multiple Participant Decision Making*, pages 99–111, 2004.

[46] Hans-Georg Beyer and Bernhard Sendhoff. Robust optimization–a comprehensive survey. *Computer methods in applied mechanics and engineering*, 196(33):3190–3218, 2007.

[47] James K Guest and Takeru Igusa. Structural optimization under uncertain loads and nodal locations. *Computer Methods in Applied Mechanics and Engineering*, 198(1):116–124, 2008.

[48] Akihiro Takezawa, Satoru Nii, Mitsuru Kitamura, and Nozomu Kogiso. Topology optimization for worst load conditions based on the eigenvalue analysis of an aggregated linear system. *Computer Methods in Applied Mechanics and Engineering*, 200(2528):2268 – 2281, 2011.

[49] AR Diaz and MPe Bendsøe. Shape optimization of structures for multiple loading conditions using a homogenization method. *Structural Optimization*, 4(1):17–22, 1992.

[50] Cheng-Shan Wang, Bo Yu, Jun Xiao, and Li Guo. Multi-scenario, multi-objective optimization of grid-parallel microgrid. In *Electric Utility Deregulation and Restructuring and Power Technologies (DRPT), 2011 4th International Conference on*, pages 1638–1646. IEEE, 2011.

[51] Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. Robust optimization of soc architectures: A multi-scenario approach. In *2008 IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia*, pages 7–12. IEEE, 2008.

[52] Philippe Duchon, Nicolas Hanusse, and Sébastien Tixeuil. Optimal randomized self-stabilizing mutual exclusion on synchronous rings. In *International Symposium on Distributed Computing*, pages 216–229. Springer, 2004.

[53] Ajoy Kumar Datta, Maria Gradinariu, and Sébastien Tixeuil. Self-stabilizing mutual exclusion using unfair distributed scheduler. In *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, pages 465–470. IEEE, 2000.

[54] Colette Johnen. Service time optimal self-stabilizing token circulation protocol on anonymous undirectional rings. In *Reliable Distributed Systems, 2002. Proceedings. 21st IEEE Symposium on*, pages 80–89. IEEE, 2002.

[55] Carroll Morgan Annabelle Mcovera. An elementary proof that herman's ring is (n2). *Information Processing Letters*, 94(2):79–84, 2005.

[56] Yuan Feng and Lijun Zhang. A tighter bound for the self-stabilization time in herman's algorithm. *Inf. Process. Lett.*, 113(13):486–488, 2013.

[57] Laurent Fribourg, Stéphane Messika, and Claudine Picaronny. Coupling and self-stabilization. *Distributed Computing*, 18(3):221–232, 2006.

[58] John Haslegrave. Bounds on herman's algorithm. *CoRR*, abs/1405.5209, 2014.

[59] Stefan Kiefer, Andrzej S. Murawski, Joël Ouaknine, Björn Wachter, and James Worrell. Three tokens in herman's algorithm. *Formal Asp. Comput.*, 24(4-6):671–678, 2012.

[60] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Probabilistic verification of herman's self-stabilisation algorithm. *Formal Asp. Comput.*, 24(4-6):661–670, 2012.

[61] Toshio Nakata. On the expected time for herman's probabilistic self-stabilizing algorithm. *Theor. Comput. Sci.*, 349(3):475–483, 2005.

[62] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *International Conference on Computer Aided Verification*, pages 585–591. Springer, 2011.

[63] Simos Gerasimou, Giordano Tamburrelli, and Radu Calinescu. Search-based synthesis of probabilistic models for quality-of-service software engineering (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 319–330. IEEE, 2015.

[64] AlanG. Millard, DavidR. White, and JohnA. Clark. Searching for pareto-optimal randomised algorithms. In Gordon Fraser and Jerffeson Teixeira de Souza, editors, *Search Based Software Engineering*, volume 7515 of *Lecture Notes in Computer Science*, pages 183–197. Springer Berlin Heidelberg, 2012.

[65] Ling Zhu and Sandeep Kulkarni. Using genetic programming to identify tradeoffs in self-stabilizing programs: A case study. In *Proceedings of the 2015 IEEE 35th International Conference on Distributed Computing Systems Workshops*, ICDCSW '15, Columbus, OH, USA, 2015.

[66] Sumit Gulwani. Dimensions in program synthesis. In *Proceedings of the 12th international ACM SIGPLAN symposium on Principles and practice of declarative programming*, pages 13–24. ACM, 2010.

[67] A. Solar-Lezama. *Program Synthesis by Sketching*. PhD thesis, University of California, Berkeley, 2008.

[68] B. Bonakdarpour, S. S. Kulkarni, and F. Abujarad. Symbolic synthesis of masking fault-tolerant distributed programs. *Distributed Computing*, 25(1):83–108, 2012.

[69] B. Jobstmann, A. Griesmayer, and R. Bloem. Program repair as a game. In *Computer Aided Verification*, pages 226–238, 2005.

[70] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[71] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Principles of Programming Languages*, pages 179–190, 1989.

[72] Armando Solar-Lezama, Rodric Rabbah, Rastislav Bodík, and Kemal Ebcioğlu. Programming by sketching for bit-streaming programs. In *ACM SIGPLAN Notices*, volume 40, pages 281–294. ACM, 2005.

[73] W. Thomas. On the synthesis of strategies in infinite games. In *Theoretical Aspects of Computer Science*, pages 1–13, 1995.

[74] A. Arcuri and X. Yao. A novel co-evolutionary approach to automatic software bug fixing. In *Evolutionary Computation, 2008. IEEE World Congress on Computational Intelligence*, pages 162–168, 2008.

[75] S. Forrest, T. Nguyen, W. Weimer, and C. Le Goues. A genetic programming approach to automated software repair. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, New York, NY, USA, 2009.

[76] T. Weise, M. Zapf, and K. Geihs. Rule-based genetic programming. In *Bio-Inspired Models of Network, Information and Computing Systems, 2007. Bionetics 2007. 2nd*, pages 8–15, 2007.

[77] T. Weise and K. Tang. Evolving distributed algorithms with genetic programming. *IEEE Transactions on Evolutionary Computation*, 16(2):242–265, 2012.

[78] Gal Katz and Doron Peled. Genetic programming and model checking: Synthesizing new mutual exclusion algorithms. In *Automated Technology for Verification and Analysis*, pages 33–47. Springer, 2008.

[79] Colin G Johnson. Genetic programming with fitness based on model checking. In *Genetic Programming*, pages 114–124. Springer, 2007.

[80] Gal Katz and Doron Peled. Synthesizing, correcting and improving code, using model checking-based genetic programming. In *Hardware and Software: Verification and Testing*, pages 246–261. Springer, 2013.

[81] Yoah Bar-David and Gadi Taubenfeld. Automatic discovery of mutual exclusion algorithms. In *Distributed Computing*, pages 136–150. Springer, 2003.

[82] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203 – 226, 1982.

[83] Tessa Lau, Pedro Domingos, and Daniel S. Weld. Learning programs from traces using version space algebra. In *Proceedings of the 2Nd International Conference on Knowledge Capture*, K-CAP '03, pages 36–43, New York, NY, USA, 2003. ACM.

[84] Andrew Hinton, Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: A tool for automatic verification of probabilistic systems. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS06), volume 3920 of LNCS*, pages 441–444. Springer, 2006.