MULTIOBJECTIVE PARTICLE SWARM OPTIMIZATION: INTEGRATION OF DYNAMIC POPULATION AND MULTIPLE-SWARM CONCEPTS AND CONSTRAINT HANDLING

By

WEN FUNG LEONG

Bachelor of Science in Electrical Engineering
Oklahoma State University
Stillwater, Oklahoma
2000

Master of Science in Electrical Engineering
Oklahoma State University
Stillwater, Oklahoma
2002

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
December, 2008

MULTIOBJECTIVE PARTICLE SWARM

OPTIMIZATION: INTEGRATION OF DYNAMIC

POPULATION AND MULTIPLE-SWARM CONCEPTS

AND CONSTRAINT HANDLING

Dissertation Approved:

Dr. Gary G. Yen
Dissertation Adviser

Dr. Guoliang Fan

Dr. Carl D. Latino

Dr. R. Russell Rhinehart

Dr. A. Gordon Emslie
Dean of the Graduate College

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Professor Gary G. Yen. Over the course of this study, he has provided his insightful guidance, continued motivation and unlimited patience in guiding my writing progress. Furthermore, he has also given me other opportunities including conference and professional experiences, and financial assistance.

My heartfelt appreciation to my committee members, Professor Guoliang Fan, Professor Carl D. Latino, and Professor R. Russell Rhinehart, for their time, valuable feedback, and constructive feedback.

Many thanks to Dr. Huantong Geng, from Nanjing University of Information Science and Technology, for sharing the source code of his published journal [164].

To my past and present colleagues of Intelligent Systems and Control Laboratory (ISCL), Sangameswar Venkatraman, Daghan Acay, Pedro Gerbase de Lima, Michel Goldstein, Monica Wu Zheng, Xiaochen Hu, Yonas G. Woldesenbet, Biruk G. Tessema, Kumlachew Woldemariam, Moayed Daneshyari, Ashwin Kadkol, Yared Nesrane, and Nardos Zewde, I thank you all for the constructive discussions, the brainstorming sessions, friendship and help. I have had the pleasure of working with Xin Zhang and I thank her for valuable inputs and collaborative work.

I am forever thankful to my parents (W.H. Leong and K.M. Yim) and siblings

(Chew, Bun, Ting, and Zhou) for being patience, giving me their unconditional love, financial and moral supports. Finally, my special thanks to my husband, Edmond J.O. Poh for his encouragement, love, and giving emotional supports.

TABLE OF CONTENTS

Chapter Page

Chapter Page

Chapter Page

LIST OF TABLES

Table Page

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Motivation

In our daily lives, we encounter problems that demand us to search for the best possible solutions. These problems such as planning one's day or monthly expenditure can be formulated as optimization problems. These problems are described by a mathematical model and objective function. An optimization problem with only one objective function is known as the single objective optimization problem (SOP). A best solution is usually obtained via either minimizing or maximizing a single objective function. However, many optimization problems encountered in the real world technical disciplines involve more than one objective. Usually, these objectives are conflicting with each other, e.g., maximizing return while minimizing risk measures in financial portfolio management. In this case, finding solutions by optimizing each objective independently is not the best way to do. If the optimum solution is found for one of the objectives it may lead to a compromise in achieving lower quality solutions by the other objectives. The optimization problems with more than one objective are referred to as multiobjective optimization problems (MOPs). An example of realistic MOPs is the aircraft design, in which the objectives comprise of fuel efficiency, payload, range, performance, speed and many other design considerations. Additionally, most real world MOPs are limited by a set of constraints. To optimize these so called constrained MOPs (CMOPs) are much

difficult since the set of optimum solutions (or the Pareto optimal set) are not only taken into consideration with trade-offs between the conflicting objectives but also must satisfy the constraints that impose upon the MOPs.

1.2 Objective

Various methods are available to tackle MOPs. The common choice is to employ the conventional methods (e.g., weighted sum method, goal programming, linear programming, min-max optimum, and etc.) or aggregating approach [1,2]. Most of these methods used to solve for MOPs follow the same design principle where all the objectives are combined together into one function by any means and optimize the new function as if it is a single objective optimization problem. These methods are not efficient in dealing with MOPs since they are designed to solve for one solution at a time instead of finding multiple solutions at once.

Heuristic methods, on the other hand, are favored in this case because they reduce the computational cost for high-dimensional optimization problems. Some of the heuristic methods, such as simulated annealing [3] and tabu search [4], face difficulty in solving MOPs, regardless of their stochastic nature, because they are not designed to find multiple solutions; while other heuristic methods, metaheuristics type, are better tools to solve MOPs. Evolutionary algorithms (EAs) are popular among the metaheuristics approaches [1,2,5-7]. They are population-based approach where multiple individuals search for a set of potential solutions in parallel and in a single run. Their design mechanisms reinforce their ability and flexibility in handling various types of problems with problem characteristics such as continuous, discontinuity, and multimodality.

Recently, a new metaheuristic design emerged from the field of swarm intelligence. This metaheuristic approach is called particle swarm optimization (PSO) [50]. It has shown great potential in solving single objective optimization problems [64-99] and has been modified necessarily to solve for MOPs [105-122]. Similar to EA, PSO also incorporates population-based approach and exhibits ability to deal with problems with different problem characteristics. The difference between PSOs and EAs is the fundamental mechanism design. EAs mimic the mechanism in biological evolution while the mechanism in PSO is inspired by the behavior of a bird flock. PSO presents two advantages over EA. PSO possesses faster convergence speed than EA and offers simplicity in implementation. Therefore, PSO is rapidly gaining attention among researchers. The advantages of PSO motivates this work in developing multiobjective optimization particle swarm optimization (MOPSO). The following discussion will relate to MOPSO unless specified otherwise.

Years of research has identified the desired attributes of a Pareto optimal set (solutions) that a multiobjective algorithm should achieve. A quality Pareto optimal set means the solutions are well extended, uniformly distributed, and near-optimal. Achieving such Pareto optimal set is challenging since it involves two compelling goals: to minimize the distance of the resulted solutions (Pareto optimal set or Pareto front) to the true Pareto set (or true Pareto front) and maximize the diversity of the resulted solutions [8]. Existing MOPSOs are designed with Pareto ranking schemes, archive maintenance strategies, and techniques to preserve the diversity, which guide the search towards a well extended, uniformly distributed, and near-optimal Pareto front.

However, to enhance the efficiency of a multiobjective optimization algorithm is not limited to develop ways to improve the convergence and techniques to promote diversity. In fact, the number of particles, i.e., swarm population size, to explore the search space in order to discover possible better solutions indirectly contributes to the efficiency improvement of an algorithm. The issue of determining an appropriate swarm population size is still at question. The easiest approach is to choose a larger population size since this would increase the chance for any MOPSOs to find the true Pareto front. A large population size, however, inevitably results in undesirable and high computational cost. Conversely, an insufficient swarm population size may result in premature convergence in MOPSO. Therefore, estimate an optimal population size requires many trial-and-error, especially for those MOPs with complicated landscape and unknown. One approach to address this disadvantage is to dynamically adjust the population size during the optimization process. Only few existing works under this research line are published, and they are all applied to MOEAs. Another approach to improve the performance of MOPSO is to employ the subpopulation concept. The reason is the swarm-like characteristic renders PSO aptness to adopt the subpopulation concept often referred to as multiple-swarm concept. Most publications in multiple swarms PSO are for single objective optimization and only a few apply this concept in multiobjective optimization. Therefore, the goal of this research is to study the dynamic population size and multiple-swarm concepts of the existing works, and develop state-of-the-art MOPSOs that fuse both elements to exploit possible improvement in efficiency and performance of existing MOPSOs.

The above discussion mainly focuses on multiobjective optimization algorithm to solve for unconstrained MOPs. Since in the real world application, many optimization problems involve a set of constraints (functions). Hence, an optimization tool must be able to handle these constraints, and also solve for the optimum solution for constrained optimization problems (COPs) or Pareto optimal set for constrained multiobjective optimization problems (CMOPs). Most EAs that are designed to solve for unconstrained MOPs lack a mechanism to handle constraints. In the past decade, many constraint handling techniques for EAs have been proposed. All these EAs are mainly aimed to solve for COPs and there are relatively less publications on MOEAs to solve for CMOPs. Since PSO is still a relatively new optimization algorithm, there is little work on applying PSO for COPs and applying MOPSO to solve for CMOPs. Thus, the second research goal is to design a MOPSO to solve for CMOPs. In order to develop the proposed MOPSO, it is essential to develop a PSO to handle constraint in COPs first and then extend the technique to design a MOPSO for CMOPs.

1.3 Contributions

The contributions of this thesis are summarized below.

- Develop a MOPSO that incorporates dynamic population and multiple swarm, in which the particles are grouped according to a user-defined number of swarms, for multiobjective optimization. This algorithm design involves dynamic swarm population strategy and adaptive local archives.
- Develop a framework for a MOPSO that dynamically adjust the number of swarms needed where under certain conditions new swarms may be added or

some existing swarms may be eliminated. Additional designs included in this algorithm are modified PSO update mechanism and objective space compression and expansion strategy.

- Develop a constrained PSO with design elements that exploit the key mechanisms to handle constraints as well as optimization of the objective function. The designs include updating personal best, maintaining feasible and infeasible global archive, adaptive acceleration constants in PSO, and mutation operators. These designs are also extended into a MOPSO to solve for CMOPs.

1.4 Outline of the Dissertation

This dissertation comprises of nine chapters and these chapters are organized as follows.

Chapter 2 provides the essential background of multiobjective optimization. Basic concepts of multiobjective optimization problem formulation and Pareto optimization are presented. Optimization methods and main topics related to multiobjective evolutionary algorithms, including test functions and performance metrics, are briefly reviewed.

Chapter 3 presents the background of the swarm intelligence field. The main objective is to understand swarm behavior, its unique benefits and the fundamental concept that render such behavior. Significant works of modeling the behavior of bird flock are reviewed since particle swarm optimization (PSO) is developed based on the principle of the social behavior of a bird flock.

In Chapter 4, history of particle swarm optimization (PSO) was discussed is presented. Then, the standard PSO equations and generic algorithm are introduced. Finally, we review the major modifications and advancements for improving the performance of original PSO. Related topics include the parameter settings, modification of the standard PSO equations, neighborhood topology, and incorporation of multiple-swarm concept into PSO.

Current works of multiobjective particle swarm optimizations (MOPSOs) that are relevant to this study are reviewed in Chapter 5. First, rationale of applying PSO for multiobjective optimization is discussed. Afterwards, a general framework of MOPSO along with the main themes related to the modification of MOPSOs is discussed.

Chapter 6 elaborates the first proposed MOPSO, namely dynamic multiobjective particle swarm optimization (DMOPSO). The chapter starts by discussing the role of population size when searching for potential solutions for a MOP. Two main concepts are incorporated: dynamic population and multiple swarms. Strategies to support the two concepts and to further improve the performance of the algorithm are detailed. Comparative study on the performance and computational cost of the DMOPSO against selected MOPSOs are analyzed.

Chapter 7 outlines the second MOPSO, i.e., dynamic multiple swarms in multiobjective particle swarm optimization (DSMOPSO). In this work, dynamic population concept is applied to regulate the number of swarms, which is different from DMOPSO in Chapter 6. Here, the number of particles in each swarm is fixed but the number of swarms is dynamically varied according to each contribution in searching for potential solutions during the search process. The development of the algorithm and key

design elements are described. Experiments to evaluate the performance and computational cost of the DSMOPSO are conducted. The chapter finishes with the sensitivity analysis and provides recommendation on the parameters settings.

In Chapter 8, a PSO and MOPSO are proposed to solve for constrained optimization problems. In this study, the multiobjective constraint handling formulation is applied. Design elements are proposed with the goal of guiding the particles towards feasible regions and leading them to the global optimum solution or the Pareto optimal set. Experiments are conducted on the benchmark functions to evaluate the performance of the proposed approaches.

Conclusions are discussed in Chapter 8. Summary of the main contributions of this thesis are reviewed. Limitations of the proposed works are identified and possible future research directions related to this study are recommended.

CHAPTER 2

MULTIOBJECTIVE OPTIMIZATION

Multiobjective optimization problems (MOPs) emerge in many fields. Difficulties arise when the MOPs involve multiple, conflicting objectives since the solution of the problems are more than one. Many conventional methods can be used to solve these MOPs but they are limited in certain aspects. Recent metaheuristics have brought the possibility of approaching MOPs in much simplistic and efficient ways. This chapter presents the basic concept of multiobjective optimization. In the following section, the background of selected optimization methods such as conventional algorithms, aggregating approaches and multiobjective evolutionary algorithms (MOEA) are elaborated. Finally, validation methodologies for MOEAs that are commonly used in many publications are presented.

2.1 Definition

Consider a minimization problem; the general form of the multiobjective optimization problem (MOPs) with k objective functions is given as follows [1,2]:

$$\min_{\mathbf{x} \in \Re^n} \mathbf{F}(\mathbf{x}) = [F_1(\mathbf{x}), F_2(\mathbf{x}), \ldots, F_k(\mathbf{x})], \tag{2.1}$$

subject to the m inequality constraints:

$$g_j(\mathbf{x}) \leq 0, \quad j = 1, 2, \ldots, m; \tag{2.2}$$

the $p - m$ equality constraints:

$$h_j(\mathbf{x}) = 0, \quad j = m+1, \ldots, p; \tag{2.3}$$

and the n decision variable bounds:

$$x_i^L \le x_i \le x_i^U, \quad i = 1, 2, \ldots, n. \tag{2.4}$$

where $\mathbf{x} = [x_1, x_2, \ldots, x_n] \in \Re^n$. (2.5)

The function F_i is known as the objective function or fitness function, and $F_i(\mathbf{x})$ is called the fitness or fitness value of F_i. $\mathbf{x}$ represents a decision vector of n decision variables, where each decision variable is bounded by a lower x_i^L, and an upper x_i^U bound. The n variable bounds constitute a *decision space* or *search space*, $S \subseteq \Re^n$, and the k objective functions constitute a *objective space*, Z. Decision vectors that minimize $\mathbf{F}(\mathbf{x})$ are also referred as solutions. $g_j(\mathbf{x})$ represents the *j*th inequality constraint while $h_j(\mathbf{x})$ represents the *j*th equality constraint. The inequality constraints that are equal to zero, i.e., $g_j(\mathbf{x}^*) = 0$, at the global optimum ($\mathbf{x}^*$) of a given problem are called *active constraints*. The feasible region ($F \subseteq S$) is defined by satisfying all constraints (Equations (2.2)-(2.4)). A solution in the feasible region ($\mathbf{x} \in F$) is called a *feasible solution*, otherwise it is considered an infeasible solution. All the solutions that lie on the feasible region is called the feasible set, Φ. Equation 2.1 presents the case of minimizing all the objective functions. By duality principles, any objective function can be converted from minimization form to maximization form or vice versa, which is given below [5]:

$$\max F_i(\mathbf{x}) = \min(-F_i(\mathbf{x})) \tag{2.5}$$

$$\min F_i(\mathbf{x}) = \max(-F_i(\mathbf{x})) \tag{2.6}$$

2.1.1 Pareto Optimization

For single objective optimization, the aim is to search for the best possible solution available, or the global optimum [6]. However, for MOPs, provided that the objectives functions are conflicting to each other, there is not just a single optimum solution but a set of optimal solutions. To obtain the set of optimum solutions, the concepts of *Pareto dominance* and *Pareto optimality* are adopted. The following discussion presents the key definitions that related to the concepts [1,2,7]:

Definition 2.1 (*Concept of Pareto Dominance*)

Consider a minimization problem, a decision vector $\mathbf{x}_a$ is said to dominate another decision vector $\mathbf{x}_b$, denoted by $\mathbf{x}_a \prec \mathbf{x}_b$, *iff*

1. $F_i(\mathbf{x}_a) \leq F_i(\mathbf{x}_b)$ for all $i = 1, 2, \ldots, k$ and
2. $F_j(\mathbf{x}_a) < F_j(\mathbf{x}_b)$ for at least one $j \in (1, 2, \ldots, k)$

Definition 2.2 (*Nondominated Set*)

Let P represent the set of decision vectors in the feasible region, $\mathrm{P} \subseteq \Phi$, the nondominated set are those decision vectors in P that are not dominated by any members of the set P, (i.e. all individuals in the nondominated set are feasible).

Definition 2.3 (*Pareto Optimal Set*)

A feasible decision vector $\mathbf{x}^*$ is Pareto optimal if there exist no feasible decision vector $\mathbf{x}_i$ for which $\mathbf{F}(\mathbf{x}_i)$ dominates $\mathbf{F}(\mathbf{x}^*)$. The collection of such decision vectors

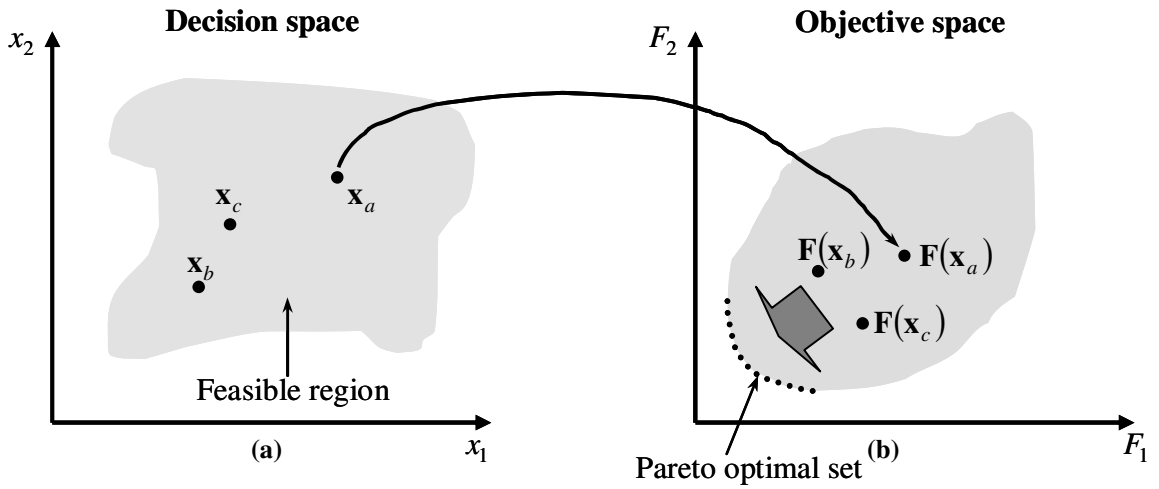


Figure 2.1 The decision vectors $\mathbf{x}_a$, $\mathbf{x}_b$, and $\mathbf{x}_c$ in the feasible region in decision space and their corresponding fitness $\mathbf{F}(\mathbf{x}_a)$, $\mathbf{F}(\mathbf{x}_b)$, and $\mathbf{F}(\mathbf{x}_c)$ in the objective space.

that are Pareto optimal is known as the *Pareto optimal set*. This means that each solution in this set holds equal importance and is a good compromise among the trade-off objectives. The resulted tradeoff curve in the objective space that obtained from Pareto optimal set is called the *Pareto front*.

2.1.2 Example

Consider a minimization problem; Figure 2.1 presents a representation of the feasible region in the decision space and the corresponding feasible objective space. Referring to Figure 2.1, the decision vectors $\mathbf{x}_a$, $\mathbf{x}_b$, and $\mathbf{x}_c$ in the decision space are mapped to the three fitness, i.e., $\mathbf{F}(\mathbf{x}_a)$, $\mathbf{F}(\mathbf{x}_b)$, and $\mathbf{F}(\mathbf{x}_c)$ respectively in the objective space. Observe Figure 2.1(b), the solution $\mathbf{x}_b$ dominates solution $\mathbf{x}_a$, since the objective $F_1(\mathbf{x}_b) < F_1(\mathbf{x}_a)$ and $F_2(\mathbf{x}_b) < F_2(\mathbf{x}_a)$, which satisfies the two conditions of Definition 2.1. Apply the same definition, solution $\mathbf{x}_c$ is also found to dominate solution $\mathbf{x}_a$. For solutions $\mathbf{x}_b$ and $\mathbf{x}_c$, both do not dominate each other because Condition 2 of Definition

2.1 is violated. In addition, solutions $\mathbf{x}_b$ and $\mathbf{x}_c$ are not dominated by another solution; hence according to Definition 2.2, $\mathbf{x}_b$ and $\mathbf{x}_c$ belong to the nondominated set. The Pareto optimal set, also the Pareto front or the tradeoff curve, is illustrated in Figure 2.1(b).

2.2 Optimization Methods

After the invention of the computer, research in optimization field been active ever since. Various optimization methods are designed and created to solve for optimization problems. There are two main classes: the conventional methods and the modern heuristics.

Table 2.1 Examples of optimization methods under the two main classes.

Conventional Methods	Modern Heuristics
Branch and Bound	Tabu Search
Dynamic Programming	Simulated Annealing
Linear Programming	Differential Algorithm
Min-max Optimum	Evolutionary Algorithms
Newton's Method	Cultural Algorithm
Divide and Conquer	Particle Swarm Optimization
Goal Programming etc…	Ant Colony Optimization etc…

Conventional methods adopt the deterministic approach. During the optimization process, any solutions found are assumed to be exact and the computation for next set of solutions completely depends on the previous solutions found. That's why conventional methods are also known as deterministic optimization methods. In addition, these methods involve certain assumptions about the formulation of the objective functions and constraint functions. Conventional methods include algorithms such as branch and bound, dynamic programming, linear programming, min-max optimum, and those listed in Table 2.1. There is a subclass under modern heuristics, which is called the stochastic based

methods. The algorithms that are categorized as stochastic based methods include simulated annealing, evolutionary algorithms, differential algorithm, cultural algorithm, and particle swarm optimization. These algorithms possess the stochastic nature while searching for possible solutions for a problem. In the following, elaboration on conventional algorithms, aggregating approach, and evolutionary algorithms are presented.

2.2.1 Conventional Algorithms

Conventional algorithms or classical methods have been around for at least four decades [1]. They possess the deterministic and predictable behavior, in which the techniques are designed to find the same solution if the same input sample and stopping criteria are applies. The search process will be much efficient and quicker if the input is located within some defined finite search space provided that the search space is not overly large. Publications have shown the success of employing these algorithms in solving a wide variety of problems [9-11], but not for problems that are high dimensional, multi-modal or NP-complete problems.

Conventional algorithm can solve MOPs. These techniques used for handling MOPs share a similar spirit, which is to convert the MOPs into a single objective optimization problem and find a preferred Pareto optimal solution [1]. Refer to the classification of algorithms given by Hwang and Masud [12], these algorithms are under the class of *priori* preference [7]. The best represented algorithms include weighted-sum method, the Goal programming method, and the min-max optimum.

Weighted-sum method [1,2,7,13] – The aggregating function is derived by pre-multiplied the multiple objectives functions with the corresponding predefined weights. Mathematically, the aggregating function is in the form:

$$F(\mathbf{x}) = \sum_{i=1}^{k} w_i F_i(\mathbf{x}), \tag{2.7}$$

where w_i are the weighting coefficients, within the range of $[0,1]$. These weighting coefficients represent the relative significances of the objective functions. To maintain the same order of scale among the objective functions, the objective functions are normalized first before applying Equation (2.7). In addition, the weighting coefficients are chosen such that the sum of these weighting coefficients is one, i.e.,

$$\sum_{i=1}^{k} w_i = 1, \tag{2.8}$$

This method has its disadvantages. It is sensitive to the weighting coefficients chosen heuristically, so prior knowledge is needed to predetermine the weights. In addition, it fails to find solutions that locate on the concave portions of the Pareto front [7].

Goal Programming Method – This method is introduced by Charnes and Cooper [14,15] in 1960s and due to its simplicity, is has applied to various fields [16,17]. The main idea is to find solutions that attain a set of predefined goals for the corresponding objective functions [1]. The general steps to find solutions by using this method are given below:

Step 1: For MOPs with k objective functions, pre-specify a set goal, t_i, where $i = 1, 2, \ldots, k$

Step 2: Setup k generic constraint equations based on the given goals, types of goal criteria, and the corresponding k objective functions. For example, the constraint equations for four different types of goal criteria are given as follows [1]:

1. Less-than-equal-to, $F(\mathbf{x}) \le t$:

Generic constraint equation: $F(\mathbf{x}) - p \le t$; (2.9)

2. Greater-than-equal-to, $F(\mathbf{x}) \ge t$

Generic constraint equation: $F(\mathbf{x}) + n \ge t$; (2.10)

3. Equal-to, $F(\mathbf{x}) = t$

Generic constraint equation: $F(\mathbf{x}) - p + n = t$; (2.11)

4. Range, $F(\mathbf{x}) \in \left[t^L, t^U\right]$

Generic constraint equation: $F(\mathbf{x}) - p \le t^L$ and $F(\mathbf{x}) + n \ge t^U$; (2.12)

The two new variable(s) appeared in Equations (2.9) to (2.12), i.e., p and n, are called the deviational variables. The aim of adding the variable(s) is to measure the difference between the goal and the achieved levels of the corresponding objective function. Detail on how Equations (2.9) to (2.12) are obtained is given in [1,14,15].

Step 3: Once the constraint equations are set, optimization technique is applied to optimize all the deviational variables as a weighed sum single objective function that subject to k constraint equations (given in Step 2). If it is a minimization problem, then all the deviational variables are to be minimized. There are many techniques available [17,18]. Among them, the common ones are the weighted goal programming (WGP) and the lexicographic goal programming (LGP).

The disadvantage of this method is need of prior knowledge to set the predefined goals for their corresponding objective functions.

Min-max Optimum – This approach is one of the techniques used in the field of game theory. Due to its design to deal with conflicting situation, it has been employed in solving the MOPs [19]. In this method, the set of solutions found will have the minimum deviation between the solutions and the individual objective function. The "min-max" criteria are used to compare relative deviation of the current best points and the individual objective function at every iterations until the set of solution is found. Detailed procedure of this method can be found in [19]. This method is capable of discovering all optimum solutions for a given the MOPs regardless if the problem is convex or nonconvex [7]. The disadvantage of min-max optimum is applied to each of the objective functions individually.

In solving the MOPs, the goal is to find the Pareto optimal set. In this case, conventional algorithm can only find one solution in one run with a fixed parameter setting. Note that a single run means that an algorithm continues its process to search for solutions until it meets the stopping criteria. Hence, to find the Pareto optimal set, multiple runs with different parameter settings for every individual objective function are required. In addition, some of these algorithms such as weighted-sum method may require prior knowledge of the problem to predetermine some of the fixed parameters; while some algorithms have difficulty in solving MOPs that have convex Pareto front [2].

2.2.2 Aggregating Approaches

In aggregating approaches, techniques are employed to combine multiple objective functions into a single objective function using either addition, multiplication, or any other combination of arithmetical operations [2]. The techniques are also known as aggregating functions and can be either linear or nonlinear. A simple example of an aggregating approach is the weighted-sum method. In general, many have known that aggregating function poses a well-known limitation, which is the difficulty in finding the concave portion of the Pareto front. However, this limitation does not necessarily hold if a nonlinear aggregating function is adopted [6]. Hence, the limitations of the aggregating approach depend on the technique employed. Although an aggregating approach may be able to find an optimum solution at each run, many runs are needed to obtain the complete optimal Pareto front for a given MOPs.

2.2.3 Multiobjective Evolutionary Algorithms (MOEAs)

Since the groundbreaking work of computer simulation of evolution in 1954 [20], along with various researchers' contributions in developing new computer simulations that merge evolution theory with computational methods, the new field of evolutionary computation has arisen. In evolutionary computation, the algorithms are population based. The population undergoes processes that iteratively guide it to achieve the desired goal. The processes can be inspired by concepts that are different from the mathematical or computer field, such as biological mechanisms of evolution or social behaviors. Among the computational techniques in evolutionary computation, evolutionary algorithms (EAs) adopted mechanism that inspired by the principle of biological

evolution [21]. EAs comprise of some well-known techniques [21-23], for instance, genetic algorithm, evolutionary programming, evolutionary strategy, and genetic programming where each employs the mechanisms of evolution yet differ in implementation.

The main disadvantage of using conventional algorithms and other mathematical programming techniques to solve MOPs are most of them are designed to solve for specific problems only and they find only one, at most, optimum solution in a single run, multiple runs are necessary to complete the Pareto front. EAs can overcome this disadvantage. Research in developing evolutionary algorithms to solve MOPs have

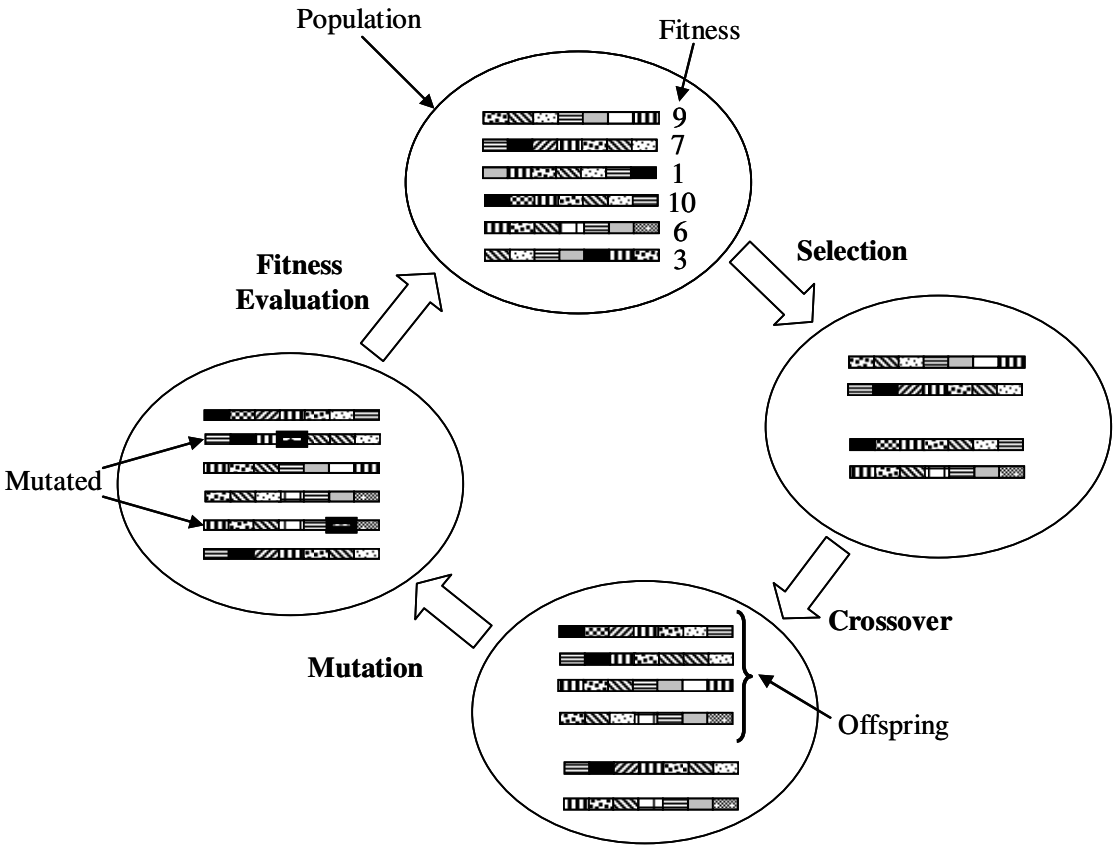


Figure 2.2 Main procedure of an evolutionary algorithm for single generation.

gained much attention for over 20 years and these algorithms are called multiobjective evolutionary algorithm (MOEA).

2.2.3.1 General Concept

The main idea of evolutionary algorithm (EA) is to model the fundamental mechanisms of evolution and utilizes evolution concept to perform optimization process. Five main mechanisms mimicked and incorporated into an EA are reproduction, natural selection, survival of the fittest, crossover, and mutation. In EA, a candidate solution, denoted as an individual, is encoded as genes in the chromosomes. A set of candidate solutions are referred to as population. During a series of iterations, or called generations, the individuals are evaluated to determine their fitness value. Based on their fitness value, those that are considered the fitter ones are selected by the selection operator because they have higher probabilities to produce "fitter" individuals (offsprings). Hence, two of the selected individuals that are randomly chosen are denoted as parents. Next, crossover operation and occasionally followed by mutation operator are applied to the parents to produce new individuals or offsprings. This reproduction process is applied to all the selected individuals. Figure 2.2 illustrates the main procedure of an evolutionary algorithm for single generation.

2.2.3.2 A Brief Tour of MOEAs

Various designs of MOEAs have been developed since the 1980s. The pioneering work of MOEA is called vector evaluated genetic algorithm (VEGA), designed by Shaffer [24]. At each generation, the whole population is divided into subpopulations of equal size. The number of subpopulations depends on the number of objective functions

in a MOPs. These subpopulations are combined and shuffled together. Crossover and mutation operators are applied to the shuffled population to obtain new population. Advantage of VEGA is its simplicity to implement and its disadvantage is the tendency to generate good solutions for one of the objective but not for all of the objectives because the selection operator would incline to select a subpopulation with better fitness values than the others.

The mark of significant contribution to MOEAs development is after David E. Goldberg's proposal of the concept of Pareto optimality [25]. His idea is to assign ranks to the individuals based on their relative Pareto dominance. Hence, the selection process is based on these rank values of the individuals. Selection pressure is imposed to guide the population towards the direction of the Pareto front. Goldberg's ranking scheme is known as the nondominated sorting (Figure 2.3 (a)) and have sparked the interest of designing Pareto based MOEAs. Several MOEAs have adopted his scheme. Among those are niched Pareto genetic algorithm (NPGA) [186] and nondominated sorting genetic algorithm (NSGA)[187]. Improved versions of Goldberg's ranking scheme are introduced in several publications. Figure 2.3 shows different Pareto ranking schemes of [25-28]. There are Fonseca's Pareto ranking scheme where the rank of an individual is corresponding to the number of other individuals that dominate it [26] (in Figure 2.3 (b)); ranking scheme proposed by SPEA [27] (refer to Figure 2.3 (c)) where fitness assignment strategy is modified to determine the "strength" of each individual, instead of rank; and automatic accumulated ranking scheme by [28] where individual's rank is corresponding to the accumulated rank of those individual that dominate it, as shown in Figure 2.3(d).

Second significant advancement in the MOEA research area is the introduction of elitism or archiving concept. Purpose of archive is to store the good solutions (i.e., nondominated solutions) found thus far from the search process. Issue of adopting archiving is what strategy to maintain the archive. The most popular of incorporation of elitism concept is introduced by Zitzler and Thiele [27]. They adopted two populations in their proposed MOEA, called strength Pareto evolutionary algorithm (SPEA). One population contains the individuals that search for solutions while the other is an external population or archive that stores limited nondominated soutions found at every generation. To maintain the archive, strength values are assigned to the solutions in the archive. These strength values will play a role in computing fitness of the current

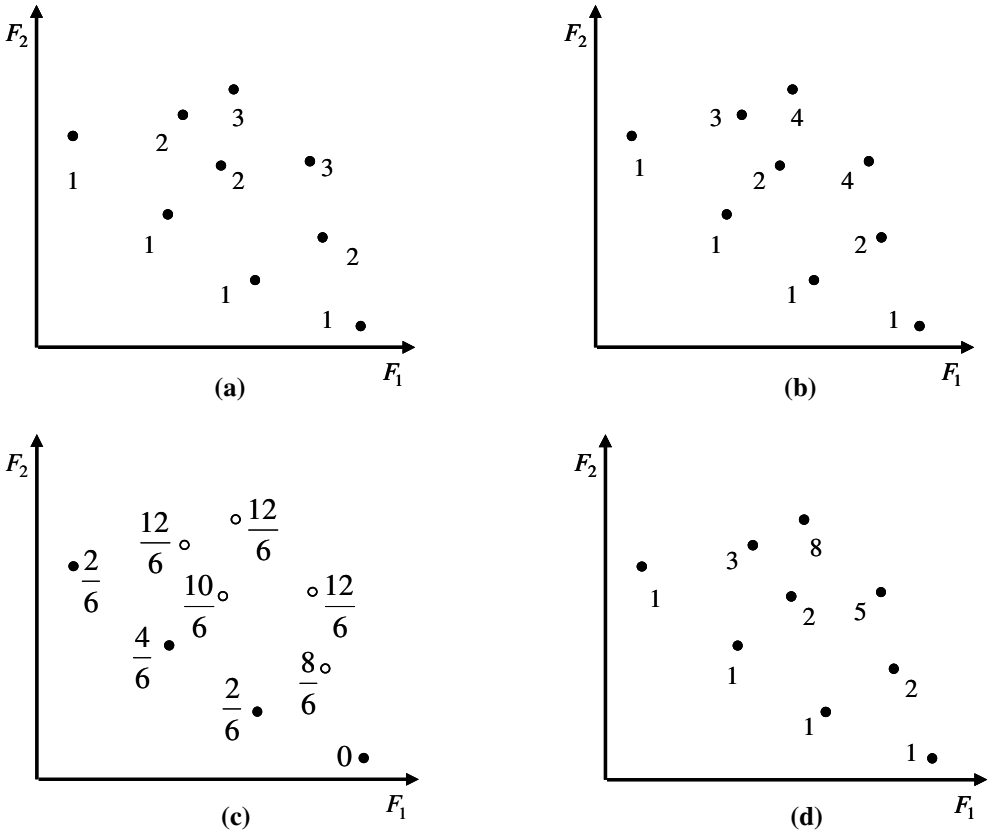
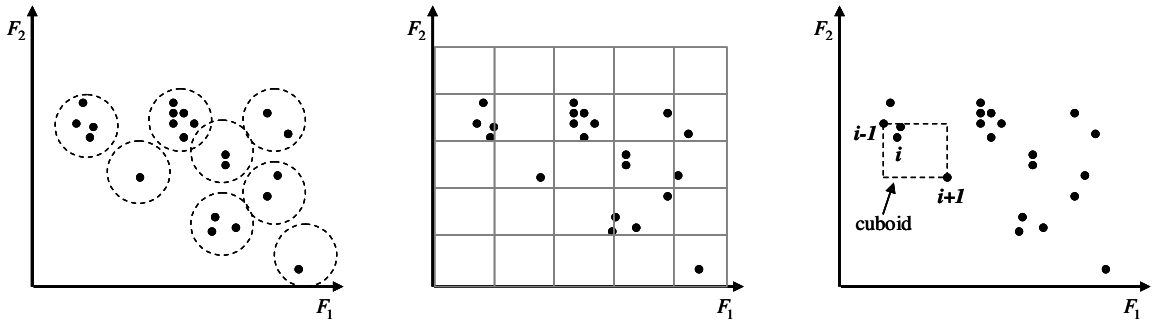


Figure 2.3 Above shows different kind of ranking schemes. (a) Goldberg's nondominated sorting [25], (b) Fonseca's ranking method [26], (c) Ranking scheme adopted in SPEA [27], and (d) Automatic accumulated ranking scheme proposed by [28].



(a) Fitness sharing **(b) Grid** **(c) Crowding distance**

Figure 2.4 Three diversity techniques proposed in [25,30,31] and used in various MOEAs. (a) In fitness sharing technique, fitness of an individual that share the same niche (dashed circles) with other individuals is reduced. (b) Grid approach is usually applied in archive for two purposes: diversity and archive maintenances. The grid regions represent a region. Individuals reside in crowded grid region have less chance to be selected. (c) In crowding distance scheme, distance of the individual, *i* and its two neighboring individuals (i.e. individuals of index *i-1* and *i+1*) in each objective function are computed.

population, which indirectly place preference to individuals that are least dominated and those that located in less populated region in the objective space. Clustering algorithm is applied to maintain the archive size and to promote diversity. Zitzler and Thiele's [27] elitism concept brought interest to many researchers to incorporate this concept to their new MOEAs. Significant work with new elitism strategies include SPEA2 [29], PAES [30], NSGA-II [31], PESA [32], and micro-GA [33].

Diversity maintenance is essential to prevent the "genetic drift" effect that causes the loss of diversity in the population. A number of works have proposed various techniques to encourage diversity. Among them, the established diversity techniques include fitness sharing [25], grid [30], and crowding distance [31], as illustrated in Figure 2.4. Please note the numbers shown next to individuals are the assigned rank values according to each specific design.

2.3 Test Functions

It is a common practice that after a MOEA is proposed, its performance is validated from the standard simulated testing process. Applying the new algorithm to a set of test functions or benchmark problems is part of the simulated testing process to show the efficiency in solving the problems. Usually, the benchmark problems are selected from a variety of available standard test functions, which they all have their own representations, difficulties and properties (i.e., multifrontality, discontinuity, and convexity in the Pareto optimal front [2]).

The earlier test functions designs for MOEA are simpler and often with two objectives [34-37]. In the last several years, several researchers have developed sets of test functions that have become the standard benchmark functions in many MOEA research publications [38-40]. Introduction of toolkits to design test functions facilitates constructing desired test suites [38-42]. Recall that the two key tasks that a MOEA should accomplish are to converge towards the optimal Pareto front and to maintain the diverse distribution of the optimal Pareto front solutions. Hence, in test problem design, these two tasks are the criteria to determine the difficulty level of the test problem. In [38], method to construct a test function is based on two basic functions, i.e. function h and function g. Consider a two objectives test function, the first objective, $F_1(\mathbf{x})$, influences the level of distribution of the Pareto optimal solutions, while the second objective, $F_2(\mathbf{x})$, is designed from the combinations of the two basic functions and $F_1(\mathbf{x})$. Function h designs the shape of the Pareto front in the objective space and function g controls the level of difficulty to converge towards the Pareto front. Following [38],

Zitzler *et al.* [39] and [40] produced standard and practical benchmark problems that are currently used for performance testing of MOEAs in many publications.

Another work by Okade *et al.* steered away from the method proposed by [38] and proposed their own method that can construct various benchmark functions with arbitrary, customized Pareto front in objective and in decision spaces [41]. They also proposed a distribution indicator to measure the difficulty of the benchmark functions based on the mapping from the decision space to the objective space. In [42], the limitations of the existing benchmark functions are analyzed. They recommended that a test suite should contain test problems with a wide range of possible features [42]. In addition, a toolkit (i.e. WFG Toolkit) to construct unconstrained test problems is proposed to provide control and flexibility to the users to design test suites that meet the desired features. Using WFG Toolkit, the authors designed a test suite that consists of test problems with various features. This new test suite may be considered complete at this time. Once this new test suite gains popularity, it would become one of those standard test functions in the MOEA field. As indicated in [6], future test functions are expected to be more complex and cover wider range of aspects and features that are closer to the real-world problems.

2.4 Performance Metrics

Metrics are applied to assess the performance of a MOEA after it finds the optimal Pareto front for a chosen test function. Performance metrics for MOEA are different from algorithms that deal with SOPs. Since the optimal solution for MOPs is a set of solutions, the designed metrics have to measure multiple solutions. In addition,

convergence and distribution of the optimal Pareto solutions produced by an algorithm are the key features to be measured. Due to the stochastic nature in all MOEAs, 30 to 50 runs are often required to assess their performance using statistical analysis. In general, performance metrics do not just measure the quality of a new MOEA but also for comparison of results produced by other MOEAs. There are two types of comparison methods: one type is the quantitative assessment and the other is the qualitative assessment.

For quantitative assessment, the metrics are based on certain equations or theories as the measuring tool. Each metric is designed to measure one feature of the overall performance. In early MOEAs publications, performance comparisons often rely on qualitative assessment. The common metrics used are aimed to measure three features as given in [2,39]:

1) Measure how close to the optimal Pareto front produced by a MOEA with respect to the true Pareto front, assuming that we know the true solutions of a MOP. An example metric is the generational distance (GD) [43] that calculates the distance between the solutions in a set of the Pareto front found and those in the true Pareto front.
2) Measure how well distributed of the solutions found by a MOEA since we want to find the Pareto front that is well extended and uniform distributed. A popular metric for this measure is called Spacing (SP) [44], which measures the distance variance of the neighboring solutions lie along the resulted optimal Pareto front.

3) Measure the number of solutions found in the optimal Pareto set belongs to the true Pareto set. The error ratio (ER) metric computes the percentage of the solutions found that belongs to the true Pareto set [45].

The above only presents a few from a wide variety of metrics. These metrics are also under the category known as the unary indicators [46,47]. There are limitations for unary indicators. The users must know true Pareto front of a MOP in order to calculate the metrics. This is not practical since in real application, one does not always have access of the true solutions of a problem. As analyzed in [46], unary indicators are restricted because when comparing two MOEAs, unary indicators are able to indicate which algorithm is better but fail to express how much better. Recent research works introduce performance metrics that consider two algorithms instated of one. These metrics are referred to as the binary indicators in [46]. Binary indicators are different from unary indicators since the performance measure is based on the dominance relationship between two MOEAs. Hence, binary indicators are able to indicate how much better of an algorithm compared to the other one. Another advantage over unary indicators is no true Pareto front is required to apply binary indicators. Examples binary indicators include multiplicative epsilon indicator [46], additive epsilon indicator [46], two set coverage [27,47], and binary hypervolume indictor [46].

Another alternative of comparing the performance of several MOEAs are through graphical representation, or qualitative assessment. Basically, all the optimal Pareto fronts produced by the MOEAs for the given same initial populatopn are presented graphically. Those MOEAs with good Pareto fronts are identified through visual judgment. This may only works for those results that are significant difference from each

other since they are easier to identify. For those MOEAs that produce good Pareto fronts, quantitative assessment can complement the difficulty in identify which MOEA is better.

CHAPTER 3

SWARM INTELLIGENCE

The study of particle swarm optimization belongs to the field of swarm intelligence. It is an optimization method which bears two similar characteristics that can be found in evolutionary computation techniques, i.e., stochastic search and population based design. This chapter starts by introducing swarm intelligence and providing the background to understand the fundamental concept behind this field. Later, a brief history in modeling flocking behavior is elaborated. The aim is to familiarize with the pioneering works in modeling flocking behavior, which bears some connection in the history of particle swarm optimization.

3.1 Introducing Swarm Intelligence

In the evolutionary computation field, the unorthodox optimization techniques are designed by modeling or incorporating the theories or concepts extracted from areas such as the natural science, psychology, or even sociology. As introduced in previous chapter, an evolutionary algorithm mimics the biological evolution by employing the mechanics of natural selection and genetics, such as mutation and recombination operations, selection pressure, and survival of the fitness, to find optimal solutions for the optimization problems. Algorithms that follow these similar mechanics [21-23] are genetic algorithm, evolution strategy, genetic programming, and evolutionary

programming.

Swarm intelligence (SI), on the other hand, is a computational intelligence technique inspired by the swarming behavior of the social insects or social animals in the nature. Swarming behavior is how the social insects or social animals interact to accomplish simple goal. They stay in groups and collaborate to find food, to provide warning and collaborative defense against any predators. Such cooperative behavior can bring benefits. Benefits include increasing foraging efficiency, reducing the probability of each individual from becoming a prey, and acquiring information about the environment quickly from each other; all to allow these social insects or animals to achieve a high survival advantage [48]. One can observe swarming behavior in nature. For example: a school of fish travels together and reacts in unison when face any external threat; a flock of birds fly across the sky; or a swarm of African termites builds their huge termite tower.

3.1.1 Fundamental Concepts

A general term to refer a social insect or social animal is known as a simple *agent*. Hence, we can say a swarm is formed when a number of agents groups and cooperates to achieve their collective purpose or some goal. Interestingly, there is neither leader nor centralized control to dictate the behavior of these agents in a swarm. Without centralized control, how do the agents gather and collectively establish some kind of complex yet functional system to collaborate towards a common goal?

One of the underlying concepts is the ability of the population of agents to undergo self-organization process. Each agent plays its role by interacting with its environment to gather the latest information, constantly making decision based on some

simple local rules and information received, and interact locally with other agents. When a population of agents groups together, each agent plays its own role, and eventually, this results in self-organization process. Consequently, the process leads to the production of a global behavior, or a swarming behavior.

Another underlying concept is the division of labor. In nature, different groups of agents within a swarm have their own specializations and specialties to carry on certain tasks. These different groups of agents cooperate and perform their own tasks simultaneously. Tasks can range from foraging, nest building, to defending their nest. The typical examples are ants, bees, wasps, and termites. With division of labor, task performance of each specialized agent can reach its highest efficiency. If there are environment changes, the agents will adjust themselves to optimize their performance.

3.1.2 Example Algorithms

Understanding the principles of swarm intelligence has brought great relevance in different disciplines such as engineering [188], robotics [189], and telecommunication fields [190]. In recent years, the research area of swarm intelligence is growing rapidly. Two areas that have shown significant applications in solving optimization problems are the "swarm-like" algorithms — ant colony optimization (ACO) [49] and particle swarm optimization (PSO) [50].

Ant colony optimization (ACO) [49] is inspired by the behavior of how natural ants find the shortest path from their nest to the food source. In nature, ants are able to find the shorter path to the food source by exchanging communication through the

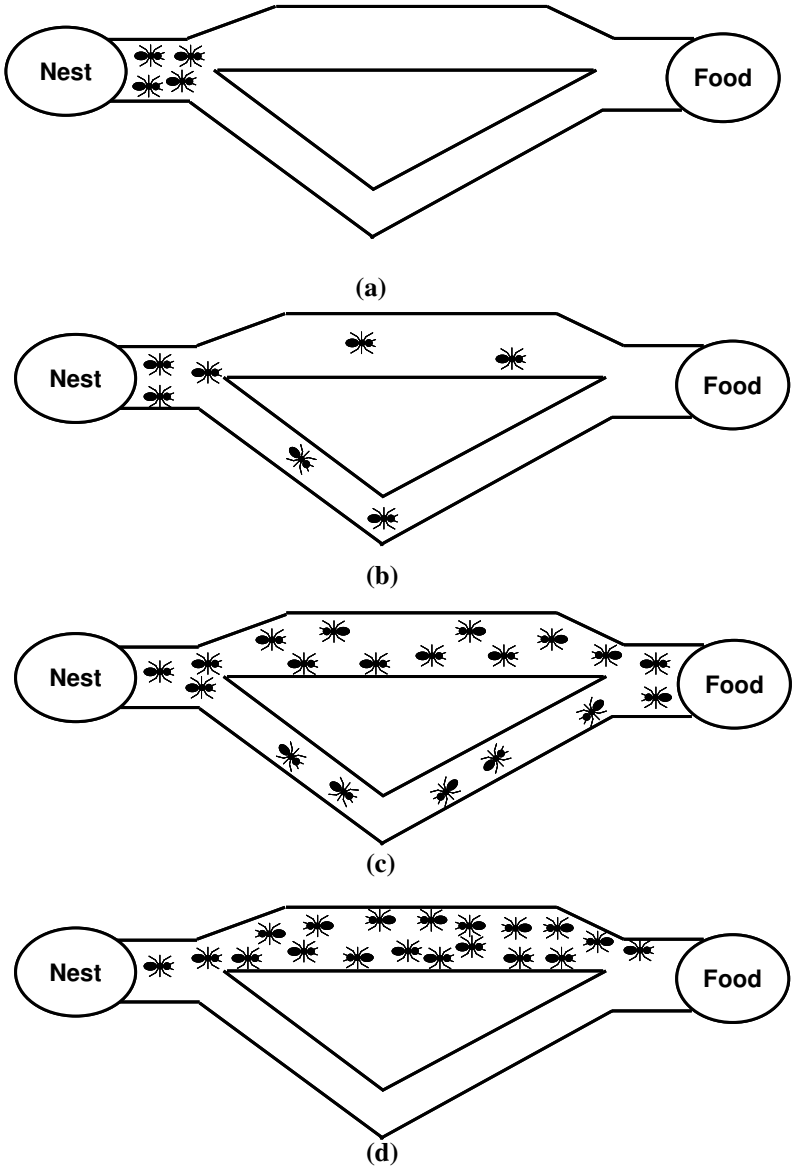


Figure 3.1 Ants' foraging behavior in finding the shortest paths from their nest to the food source. (a) Ants are at the junction of the two paths that can lead to the food source from their nest. (b) The ants choose the path randomly. (c) Ants leave the pheromone trail while returning to their nest after find food. Shorter path (upper path) has higher pheromone concentration than longer path (lower path), which attracts more ants to choose the shorter path. (c) Eventually, All ants will end up using the shorter path.

chemical message called pheromone, which is deposited by ants on the ground. The ants' foraging behavior is illustrated in Figure 3.1. In Figure 3.1a, there are two paths that can lead to the food source from the ants nest. The ants' goal is to go to the food source and have no idea which path is the better one. Hence, they choose the path randomly, as

shown in Figure 3.1b. Some ants choose the upper path while others choose the lower path. When the ant find the food, it will leave the pheromone trail along the ground while returning to its nest using the same path it has chosen earlier. With the pheromone trail, other ants from the nest will likely to follow the trail to the food source. Again, if they find food, in returning to their nest, the ants will leave the pheromone trail. This will increase the concentration of the pheromone scent and will attract more ants to follow the trail with higher concentration. The pheromone trail will dissipate over time, so this will reduce the concentration of the pheromone scent. As the ants travel the shorter path (the upper path), the concentration level of the pheromone will tend to become denser since travel duration is shorter and the pheromone concentration still remains. On the other hand, at longer path (lower path), the longer travel duration allow more time for the pheromone concentration to dissipate. Hence, by comparing the pheromone concentration between two paths, the ants are likely to choose the shorter path, which is illustrated in Figure 3.1c. As time progresses, more ants will choose the shorter path and eventually, all ants will end up using the shorter path (refer to Figure 3.1d). This process of how ants find the shortest path is the core concept of the ant colony optimization. Ant colony optimization has shown effective in finding optimum path for optimization problems with graph related such as network routing. Publications have shown ant colony optimization effectiveness in optimizing combinatorial optimization problems such as traveling salesman problems (TSP) [51] and quadratic assignment problems [52].

Particle swarm optimization (PSO) is developed based on the principle of the social behavior of a bird flock. The inventor of this algorithm is Kennedy and Eberhart [50]. They are inspired by one of the pioneers who had researched the bird flocking

behavior and developed rules to model a flock. In recent years, publications on PSO have shown promise in solving optimization problems and many applications [188, 191-196]. Since the existence of PSO bears relation to modeling bird flock in the nature, the background of related publications on modeling flocking behavior is briefly introduced in the following section.

3.2 Modeling the Behavior of Bird Flock

You may have enjoyed the sight of a flock of western sandpipers over a bay stay close together to avoid and to confuse possible predators. When one of western sandpipers in the flock turns to a different direction, the rest behind this western sandpiper will follow and turn to the same direction in unison. Or you may have observed the beautiful sight of a flock of common cranes fly in a "V" formation, migrate to warmer regions. Flying in "V" formation is to gain social advantages and to reduce energy expenditure [53]. It is amazing of how the flock, without a leader, can stay or fly close enough and not collide into each other, change direction in unison, and group together after being separated into several smaller flocks by an obstacle. In recent years, publications show continuing research and modeling of this flocking behavior [57-60]. The two significant works in modeling the flocking behavior via computer simulation are published by [54,55] and [56] during the 1980s.

In 1986, Craig Reynolds proposed an approach to simulate the flock motion [54,55]. In his approach, the resulting flock motion is contributed by the interaction between the behaviors of individual birds. Since the simulation is intended to model the flocking behavior of simple, generic agents, he referred these agents as *boids*. Each boid

has its own coordinate system and applies geometric flight model to support its flight movement. Geometric flight model includes translation and flight dynamic parameters of yaw, pitch, and banking (roll). Additional three steering behaviors (local rules) are incorporated in each boid. These three steering behaviors model the underlying concept of flocking, in which each boid desires to stay close within the flock and to avoid collision with others boids of the flock. Each boid has its own local neighborhood, which is similar to the limited perception range of birds or fishes in the nature. The neighborhood is determined by the distance from the center of a boid (green) and the the boid's heading direction is determined by the angle as shown in Figure 3.2. The flockmates within a boid's neighborhood are referred to as the local flockmates.

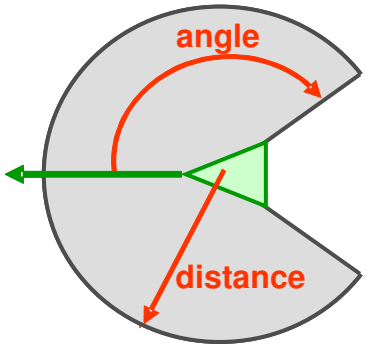


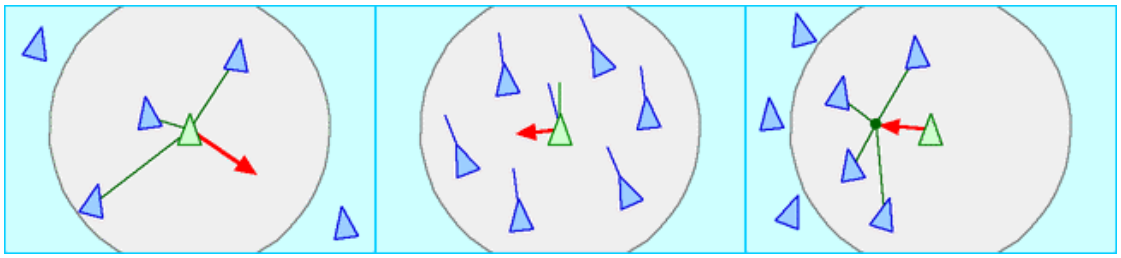
Figure 3.2 A boid's neighborhood (in grey) and the triangular symbol (marked green) represents a boid [54,55].

The three steering behaviors describe the maneuverability of an individual boid [54,55]:

(1) *Separation* behavior for collision avoidance (Figure 3.3(a)): The relative positions of a boid (green) and the local flockmates are compared. If the relative positions of a boid and the local flockmates are less than the minimum required separation distance, it indicates that the boid is too near and may collides to those local flockmates.The velocity and position of the boid is adjusted to steer away in order to avoid collision (red arrow).

(2) *Alignment* behavior for velocity matching (Figure 3.3 (b)): The velocity of a boid (green) is compared with the local flockmates' velocities. Then, the boid will adjust its velocity to match the velocity of the local flockmates and steer towards the average direction (red arrow) of the local flockmates (blue).

(3) *Cohesion* behavior or flock centering: Each boid attempts to steer towards the average position of its local flockmates. This is to ensure that the boid stays within the flock and doesn't fly away from its flock. Please note the boid (green) and its steering direction (red arrow) in Figure 3.3(c).



(a) **(b)** **(c)**

Figure 3.3 The illustrations of the three steering behaviors of the boids. (The color in the illustrations are indicated as follow: the boid (in green and is attached with an red arrow), its neighborhood (in grey) and its local flockmates (in blue) [55].

With the three steering behaviors, Reynolds successfully developed a program that the simulated flock shows realistic flocking behavior and if there is an obstacle, the flock will bifurcate to avoid or turn away from the obstacle and later, the separated flocks will rejoin together. His pioneering work became the stepping stone for many researchers to explore in this field including a computer graphic area known as the behavior animation which is implemented in movies such as The Lion King (1994) and Lord of the Ring trilogy (2001-2003).

About the same time, zoologist Frank H. Heppner studied bird flock from the movies taken by him. Through his observation, he concluded that there is no leader in the flock but interestingly, the flock can maintain a stage of dynamic equilibrium [56]. Later,

he collaborated with applied mathematician Ulf Grenander and undergraduate computer wiz Daniel Potter to develop a program that simulates an artificial bird flocks. Through their research and observation from the movies, Heppner realized that chaotic theory can be used to explain the emergent behavior in flock. Hence, they designed four simple rules to model an individual bird's behavior. The four rules include: 1) The attractive force is to allow the birds to attract each other (stay close together) and repulsive forces is to prevent the birds to fly too close to each other; 2) Each bird maintains the same velocity as its neighboring birds; 3) Occasionally, the birds' flight path can be altered by a random input (craziness); 4) Any birds are attracted to a roost and the attraction increases as the birds are flying closer to the roost. The concept of Heppner's roost idea is similar to the foraging behavior of a flock of birds. Firstly, a flock of birds are flying freely around. When one of the birds in the flock spots a roost area, it is attracted to the roost and flies towards the roost until it is finally landed on the roost. At the same time, with the "attraction force" rule, its nearest neighbors are being pulled towards the roost area. As these neighbors land on the roost, they will "attract" their nearest neighbors towards the roost area. This similar behavior is continued until the entire flock lands on the roost. By incorporating these rules in the program, the resulted simulation displays a global behavior of a group of the artificial birds, which is similar to the behavior of a flock observed in the movies.

The pioneering works of Reynolds and Heppner have brought a closer understanding of the flocking behavior using computer simulation. Following their works publications in modeling bird flock have appeared. There is [57] that simulates the Reynolds's bird flocking model via a process-parallel approach in which each processor

will simulate a fixed number of birds in a portion of simulated world. Each processor is known as bird processor. All results produced by the bird processors are gathered into a central system, the drawing process, and the results are visualized via Silicon Graphic Machine. Another approach is the used of detail mathematic derivations and theories to model and analyze the collective coherent motion of a large number of self-propelled organisms [58]. In another publication by Spector *et al.* [59], the authors developed two systems. The first system (SwarmEvolve 1.0) aims to observe how different species evolve their behavior in order to achieve their respective goal. The species and motion control formula (from classic flocking algorithm) are hard-coded. On the other hand, for the second system (SwarmEvolve 2.0) the behavior of individual agent, instead of species, is controlled by evolved computer programs. Unlike the behavior observed in the first system, the experiments show that the agents, in the second system, emerge to food-sharing behaviors in both stable and dynamic, unstable environments. Recently, a publication inspired by Reynolds's work proposed four local control laws for the flocking agents, i.e., global alignment of velocity vector, same convergence speed, collision avoidance, and local minimization of agents' artificial potential energy [60]. The authors applied the algebraic graph theory to support the topology interaction and interconnections between the agents. Simulation results showed the flocking behavior of the agents is maintained while robust to any arbitrary changes in the topology interconnections between the agents.

The motives of the discussed publications include study and understanding the distinct behavior and characteristic of flocking; development of model to explain coordinated movements of flock; and realizing the model via computer programming to

experiment and to simulate the artificial bird flocks. In terms of application domain, flocking simulation shows useful in animation or in developing multi-agent modeling for autonomous robots. In addition to Reynolds's flocking simulation, Heppner's roost idea has motivated Kennedy and Eberhart to draw the connection between flocking patterns (swarm-like) and roost. By understanding the connection, they synthesized the ideas into developing a model. Through many experiments and adjustment of their model, they proposed their final version of the model, particle swarm optimization [50], which contributes to optimization area and engineering applications. The next chapter will review the particle swarm optimization (PSO).

CHAPTER 4

PARTICLE SWARM OPTIMIZATION

When particle swarm optimization (PSO) was first introduced, the implementation focused on solving problems, mainly on single-objective optimization problems (SOPs). In recent years, many variations of PSO are designed and developed, aiming to improve its robustness and efficiency, especially on dealing with the issue of premature convergence, when solving the SOPs. The incentive of gaining attention in PSO field is also due to its simplicity and ability in solving problems closer to those in real world such as engineering applications, music composition, market modeling, and other applications. Until recently, PSO is applied to solve for multiobjective optimization, which is known as MOPSO. Before going into MOPSO discussions, it is necessary to learn the background of PSO first. This chapter will present the brief history of PSO, the PSO algorithm itself, and different variations of PSO.

4.1 Brief History of Particle Swan Optimization

The models of flocking simulation by Reynolds [54,55] and Heppner and Grenander [56] have something in common. Both models derived from the notion in which the reaction and response of each individual bird is based on the local interaction between its neighboring birds. Hence, as stated in [50], both models relied on continuing comparison of inter-individual distances and flight velocities for the birds to maintain

optimum distance between themselves and their neighbors.

In addition to the inspiration of these models, Kennedy and Eberhart further their study from Heppner's roost idea [56, 61]. In Heppner's bird simulation, the birds are attracted to the roost area until the whole flock lands on the roost. Knowing this is only in simulations where the roost is programmed and known by the birds. However, in nature, how do the birds know where to locate food ("roost") when they are hundred feet in the air? This is the question that sparked their interest to explore not only the animal behavior but also the area of social psychology, which related to social behavior of the human beings. From their study, they concluded that knowledge is shared within the flock. While a flock of birds is flying around to look for food, the fact is they are looking for signs of any food; sign of other birds eating or sign of other bird are approaching their target. Once any of the birds within the flock notice those signs, e.g., food, they pass their findings and knowledge to their peers. The flock of birds responds to the knowledge and circles around the food area, continue to collect and share new information, until it is sure that the food area is safe before they decides to go for it. The concept of knowledge sharing among the birds is intriguing to Kennedy and Eberhart. They started with the simple method called *cornfield vector* to model the behavior of how a flock of birds seeks for food, i.e., in this case food is the cornfield. Basically, velocity of the agent (bird) is adjusted based on two conditions, which are 1) the present position, XY coordinates of the each agent (bird), is compared with its best position found so far (*pbest*); and 2) the present position, XY coordinates of each agent (bird), is compared with its global best position (*gbest*) found by one member in the flock so far. A simple way of speaking: *pbest* represents the agent's desire to improve itself by remembering its best achievement

while *gbest* represents the agent's desire to learn from the best among its flock or community. Through experiments with the *cornfield vector* model, matching velocity and craziness, which are from Heppners' rules, are not necessary. The flock acts like swarm and is able to locate the cornfield. In addition, the *cornfield vector* model is expanded for multidimensional search. The authors apply the multidimensional *cornfield vector* model (algorithm) to train the weights of a three-layer feedforward perceptron neural network in solving the exclusive-or (XOR) problem. The algorithm produced good performance.

Kennedy and Eberhart continued to improve the *cornfield vector* model. In one version, they incorporated acceleration by distance instead of comparing the conditions. This way, the model is simplified into one velocity equation where either the distance between an agent's present position and its *pbest* or the distance between an agent's present position and the *gbest* is incorporated to adjust the velocity of the agent. In their final version [50], they combined the *pbest* and *gbest* into one velocity equation, added and changed some parameters in the equation. The final version is named particle swarm optimization (PSO) equation and the birds are represented by a general term, called *particles*, instead of agents.

After Kennedy and Eberhart introduced their original PSO, motivation to improve the original PSO has brought many researchers to design other variations of PSO [62-99]. One of the versions is considered the standard PSO due to its popularity in applying to many applications especially in optimization task. This standard PSO has a minor difference from the original PSO. Inertial weight is introduced to modify the velocity equation of the original PSO [62]. Including the inertial weight will enhance the

exploration in the search process, which produces better results if compare with the original PSO. The standard PSO equation is given in the following section.

4.2 Standard PSO Equations

While implementing PSO algorithm to optimize a problem, the collection of particles, acting like a swarm, will "fly" through the search space toward the regions where the optimum solutions may lie. The movement of each particle is adjusted via the velocity and position equations. The velocity equation updates the velocity of a particle, which in turn provides distance and direction of the particles. It is added to the particle's current position, gives the new particle's position. The velocity and position equations are given as

$$v_{i,j}(t+1) = w \times v_{i,j}(t) + c_1 \times r_1 \times \left(pbest_{i,j} - x_{i,j}(t)\right) + c_2 \times r_2 \times \left(gbest_j - x_{i,j}(t)\right) \tag{4.1}$$

$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1) \tag{4.2}$$

where $x_{i,j}(t)$ denotes the position (decision variable) for dimension j of particle i at iteration t; $v_{i,j}(t)$ is the velocity for dimension j of particle i at iteration t; w is the inertial weight to control the impact of the history of velocities on current velocity; c_1 and c_2 are the acceleration constants; r_1 and r_2 are random numbers within $[0,1]$ that are regenerated at every iteration; $pbest_{i,j}$ denotes the personal best position of dimension j attained by particle i thus far; and $gbest_j$ represents the global best position of dimension j discover by all particles.

There are three components at the right hand side of Equation (4.1). The first component is called the momentum component where the inertial weight controls the impact of the previous velocity. The second component, also known as the cognitive component, has the parameter $pbest_{i,j}$. This component is related to personal desire to exceed its current achievement. The third component involves $gbest_j$, is called social component since it represents social knowledge attained via the "collaborative" efforts of all the particles. In addition, it will guide particles to converge towards the attained optimal solution.

A problem faced by any optimization method is during the optimization process, the solution candidates may exceed outside of the decision variable bounds (i.e. Equation 2.4). The decision variable bounds are usually determined by users before applying any optimization method. The reason relies on the assumption that the method (or algorithm) should locate the global minimum within the user defined bounds. For PSO, despite the fact that user defined bounds are applied; additional condition is enforced to prevent the particles from exceeding outside of the bounds. Either one of the following conditions are acceptable:

- Position clipping criterion — This criterion is considered as hard boundary condition [63] since if the particles exceed the boundary, their positions are clipped. The position at dimension *j* for the particle is bounded by the minimum and maximum bounds in the decision space, i.e., $\lfloor x\min_j, x\max_j \rfloor$. $\lfloor \cdot \rfloor$ represents the floor function. The minimum and maximum bounds can be determined by user. On the other hand, the decision variables' lower and upper bounds (Equation

(2.4)) can be used as the position clipping criterion if they are given by the problem (refer to Figure 4.1).

Function positionclip (i, j, x_j^L, x_j^U, $x_{i,j}$)

```
/* i  = Index for particle i
/* j  = Dimension j of the decision variables
/* x_j^L = Lower bound of decision variable at dimension j
/* x_j^U = Upper bound of decision variable at dimension j
Begin
    xmin_j = x_j^L
    xmax_j = x_j^U
    /*Checking for lower bound,
    If  x_{i,j} ≤ xmin_j
          x_{i,j} = xmin_j
    ElseIf  x_{i,j} ≥ xmax_j
          x_{i,j} = xmax_j
    EndIf
End
```

Figure 4.1 Position clipping criterion.

- Velocity clipping criterion — Kennedy and Eberhart [50] have investigated the impact of applying velocity clipping criterion in PSO. Investigation revealed the compulsory of applying this criterion in order for the swarm to converge toward the global minimum. In [63], this criterion is considered as soft boundary condition since the particles can exist outside the boundary in the decision space even if their velocities are clipped. For this criterion, each particle's velocity at dimension j is not allowed to exceed the user defined maximum velocity threshold, i.e., $[-V_{\max}, V_{\max}]$. However, the user defined maximum velocity

threshold must not exceed the minimum and maximum bounds in the decision space. Figure 4.2 shows the psudocode of velocity clipping criterion.

```
Function velocityclip ( i , j , x_j^L , x_j^U , v_{i,j} , V_max )

   /* i  = Index for particle i
   /* j  = Dimension j of the decision variables
   /* x_j^L = Lower bound of decision variable at dimension j
   /* x_j^U = Upper bound of decision variable at dimension j
   /* V_max = Maximum velocity value
   Begin
      If  v_{i,j} ≤ -V_max
            v_{i,j} = -V_max
      ElseIf  v_{i,j} ≥ V_max
            v_{i,j} = V_max
      EndIf
   End
```

Figure 4.2 Velocity clipping criterion.

4.3 The Generic PSO Algorithm

As stated in previous chapter, PSO possess as the same characteristics as evolutionary algorithms, but the behavior of the individuals known as particles operates like a swarm that flies through the hyperdimensional search space to reach its destination. The behavior of the particles is influenced by their tendency to learn from their personal past experience and from the success of their peers to adjust the flying speed and direction.

The generic procedures are as follows: At iteration $t = 0$, the particles' positions and their velocities are randomly generated. Next, the particles' current best positions (*pbest*) are recorded. During the initialization process, parameters such as total number of particles in a swarm, max iteration counter, c_1 , c_2, and w, are set by user. Then, the

```
Begin
/*Initialization
Set t = 0
Set total number of particles in a swarm ( mswarm )
Initialize swarm, particles' positions are randomly generated ( x )
Initialize velocity rabdomly
Set c1 , c2 , and w
Set max iteration counter( t max ),
Store particles' current position ( pbest = x )
While t ≤ t max
    For i = 1 to mswarm
        Fitness Evaluation
        Update gbest by comparing with the current fitness values
      /* n = Dimension size of the decision variables
      For j = 1 to n
          Update pbest_{i,j} by comparing with the current fitness values
          Update velocity Equation (Equation 4.1)
          Update position Equation (Equation 4.2)
          Apply positionclip() (or velocityclip())
      EndFor
    EndFor
Endwhile
End
```

Figure 4.3 Pseudocode of the generic PSO algorithm.

following steps are repeated until the maximum iteration counter is reached: 1) the fitness of the objective problem is evaluated; 2) update *pbest* and *gbest* based on the comparison of the current and their (recorded) fitness values; 3) update particle velocity and position equations; and 4) apply either position or velocity clipping criterion to prevent particles from leaving the bounded search space. Figure 4.3 presents the pseudocode for the generic PSO algorithm.

4.4 Modifications in PSO

PSO has shown robust and efficient behavior in well known SOPs and some applications [188, 191-196]. However, one disadvantage posed by PSO is the lack of diversity and often trapped in the local optimum solution. Due to these reasons, it has

gained attention in researching ways to improve the performances of the original PSO. Modifications and improvements that are commonly found in publications involve several areas, i.e., parameters settings, modification of velocity and position equations, neighborhood topology, and multiple-swarm concept in PSO. The following sections provide a closer look in these modifications.

4.4.1 Parameter Settings

The three components each contribute to the velocity of the particle. The momentum component relates to how much impact should the previous velocity grant to the particle while the cognitive and social components contribute to the change in direction and velocity of the particles. The parameters attached to these three components result in their significant contributions to the particle's velocity, indirectly influence the efficiency of PSO. Knowing the importance of these parameters, various techniques are integrated to study the determination of these parameters.

4.4.1.1 Inertial Weight

In the standard PSO equation, the inertial weight is user defined. Larger inertial weight promotes exploration and conversely, smaller inertial weight support locals exploitation. Promoting too much exploration will render PSO failure to converge to optimum solution while excessive exploitation will result in premature convergence. So, to determine the "right" inertial weight is not an easy task since the optimization process for every problem is different. Several publications have proposed different methods of selecting inertial weights [64-66]. In [64], the inertial weight (w) in Equation (4.1) is

replaced by a random numbers that are uniformly distributed within $[0,1]$. Their reason is to facilitate both global exploration and local exploitation during the optimization process.

Another version applies linearly decreasing inertial weight with respect to the number of iteration count [65]. This method is known as linearly varying inertia weight:

$$w = \left[(w_1 - w_2) \times \frac{t\max - t}{t\max} \right] + w_2, \tag{4.3}$$

where w_1 is the initial inertial weight with larger value; w_2 is the final inertial weight that has smaller value; $t\max$ denotes the maximum iteration count; and t represents the current iteration count. Here is the idea: During early iteration counts, larger inertial weight will encourage global exploration to locate as many quality solutions as possible. As number of iteration counts increases, the inertial weight reduces until it reaches the final inertial weight. By reducing the inertial weight, local exploitation is slowly dominate the search process. The logic is, as iteration count is closer to maximum number of iterations it is assumed that the search region is close to the optimum solution. Hence, local exploitation will encourage the particles towards the optimum solution and not roam elsewhere.

Qin *et al.* [66] proposed an adaptive inertial weight particle swarm (AIW-PSO) optimization. In their approach, a measure called individual search ability (ISA) to indicate the current situation of each particle. In other words, the measure aimed to determine if the particle is lack of exploration or exploitation. Based on this measure, the inertial weight is dynamically adjusted via a transfer function and then assigned it to the particle. Experiment showed that AIW-PSO performs better than other selected PSO.

4.4.1.2 Acceleration Constants

The acceleration constants (c_1 and c_2) control the significance of the cognitive and social components to the particle's velocity. Both components play a critical role in guiding PSO to find the optimum solution. Hence, the balance of contribution is needed because a larger value of cognitive component will result in excessive exploration while a larger value of social component will lead to convergence toward the local solutions. Initially, Kennedy and Eberhart suggested that both acceleration constants should be set to 2 in order to bring the stochastic factor of the original PSO equation to 1 [50]. By doing this, the contribution of both components is balanced.

Ratnaweera *et al.* apply the similar technique as the linearly varying inertia weight (Equation 4.3) to adjust the acceleration constants [67]. They named their method as time varying acceleration coefficients (TVAC). The TVAC equations are given below [67].

$$c_1 = \left[\left(c_{1f} - c_{1i} \right) \times \frac{t\max - t}{t\max} \right] + c_{1i}, \tag{4.4}$$

$$c_2 = \left[\left(c_{2f} - c_{2i} \right) \times \frac{t\max - t}{t\max} \right] + c_{2i}, \tag{4.5}$$

where c_{1i} and c_{1f} are the initial and final acceleration constants for cognitive component; similarly, c_{2i} and c_{2f} are the initial and final acceleration constants for social component; $t\max$ denotes the maximum iteration count; and t represents the current iteration count. The authors suggested that c_1 should decrease from larger to lower values, e.g., the range from 2.5 to 0.5, and c_2 should increase from lower to higher values, e.g., the range from 0.5 to 2.5. The suggestion supported exploration in early

iteration counts with larger value of cognitive component and promoted quick convergence to the optimum solution in the later stage with larger value of social component.

4.4.1.3 Clipping Criterion

Generally, the maximum velocity threshold in velocity clipping criterion are pre-determined by user as discussed in Section 4.2. However, recent publication by Cui *et al.* proposed the adaptive velocity threshold idea for velocity clipping criterion [68]. Rather than using the same user defined maximum velocity threshold, their idea is to add a second maximum velocity threshold equation, where the new maximum velocity threshold is computed from multiplication of the current maximum velocity threshold with a probability density value for every iteration count. This multiplication is also applied to each dimension of the particle. Hence, the maximum velocity threshold is changed dynamically in every iteration count and in each dimension of the particle. In their discussion, by modifying the maximum velocity threshold dynamically, the particles will enhance their exploration capability with larger threshold as well as exploitation capability with smaller threshold. The frequencies of larger and smaller thresholds depend on the selected probability density functions such as Gaussian or Cauchy.

In another publication, the authors investigated the effect of different boundary conditions or clipping criterion on PSO performance [63]. Their study involved soft and hard boundary conditions (velocity and position clipping criteria), including their integration with reflection boundary condition (RBC), and absorbing boundary condition (ABC). These boundaries are incorporated in the standard PSO algorithm and the

algorithm is applied to a sphere test function and problem of synthesizing linear array antennas. Results show that combining hard boundary conditions with RBC or ABC produced an improved version of PSO. In addition, integration of RBC or ABC with soft boundary conditions gives flexibility in choosing the maximum velocity thresholds in order for PSO to obtain better convergence performance.

4.4.2 Modifications of PSO Equations

Since the introduction of the standard PSO, some researchers have attempted to improve the efficiency and performance of the standard PSO from another perspective—to design a new PSO model or modify the standard PSO.

In 2002, Maurice Clerc proposed a new PSO velocity equation [69]. Through studying the swarm behavior of the standard PSO, he realized under certain conditions the swarm will converge to the optimum solution. From there, he introduced a new parameter, the constriction factor (χ) to the standard PSO. Here is the authors's proposed PSO model or the Canonical PSO:

$$v_{i,j}(t+1) = \chi\left(v_{i,j}(t) + c_1 \times r_1 \times \left(pbest_{i,j} - x_{i,j}(t)\right) + c_2 \times r_2 \times \left(gbest_j - x_{i,j}(t)\right)\right), \tag{4.6}$$

$$\chi = \frac{2\kappa}{\left|2 - \phi - \sqrt{\phi(\phi-4)}\right|}, \tag{4.7}$$

$$\phi = c_1 + c_2, \tag{4.8}$$

where $\kappa \in [0,1]$ and $\phi > 4$. The parameter κ is a user-defined parameter and it controls the convergence speed to the optima. When κ is closer to 0, χ will be close to 0, the resulted velocity will be small. Smaller velocity facilitates a fine grain search, and

encourages convergence rate. If κ is closer to 1, particles show high exploration behavior, which results in slow convergence rate. The essence is to use constriction factor to restrain the velocity to guarantee convergence of the particles. Note that the position equation is the same as Equation 4.2. Experiments showed that with constriction factor, the particle's velocity is able to stay within the feasible search space and locate the optimum solution without even implementing the velocity clipping criterion.

Kennedy [70] proposed another PSO model, known as the Gaussian "bare bones" PSO. He studied the behavior of the velocity via simulations graphs for different versions of canonical PSO, and also the histogram of points tested with canonical PSO with *gbest* and *pbest* held constant. The study led to developing a simple PSO model that is based on Gaussian distribution model. This model has no velocity equation. The model is able to show similar behavior as canonical PSO. The model is as follow:

$$x_{i,j}(t+1) = Gaussian\left(\frac{pbest_{i,j} + gbest_j}{2}, \left|pbest_{i,j} - gbest_j\right|\right) \tag{4.9}$$

The author commented that this simplified model retain the fundamental characteristic of the PSO. The performance of Gaussian "bare bones" PSO is compared with the Canonical PSO in terms of progression characteristic of mean global best over 3000 iterations and ability to find the average optima global solution at 3000 iterations for the six standard test functions. In overall, the results show that Canonical PSO is able to progress quicker towards the mean global best and to obtain better quality of average optima global solution by 3000 iterations.

The above two PSO models (Equations 4.6-4.9) are derived from the study and analysis of the particles' behavior. Other variations in PSO model are designed by integrating with other existing techniques. [71] integrates the mutation operator to aid the

Gaussian PSO to escape the local minima. In the Gaussian PSO model, the velocity equation is

$$v_{i,j}(t+1) = r_1 \times \left(pbest_{i,j} - x_{i,j}(t)\right) + r_2 \times \left(gbest_j - x_{i,j}(t)\right) \tag{4.10}$$

In brief, during the optimization process, the particles' velocity is updated using Equation (4.10) and their position is updated using Equation (4.2). The particles' fitness are monitored to track the particles' movement progression. If there is no improvement in fitness value, a failure counter is evoked and updated in every iteration if the particles still show no sign of improvement. Once the failure counter hit the predefined number, the Gaussian PSO is replaced by Equation (4.11), in hope that particles will "jump" to a new region and escape possible local minima. Mutation operator can be either Gaussian or Cauchy probability function.

$$x_{i,j}(t+1) = x_{i,j}(t) + \eta Gaussian(0,1) \tag{4.11}$$

Note that η is a constant. The Gaussian PSO with jump strategy is tested in selected multimodel benchmark functions and is compared with other earlier PSO versions. Results showed competitive performance compared with canonical PSO [69] and self adaptive evolutionary programming. Another work also incorporated the Gaussian mutation in the position equation, which is somewhat similar to Equation (4.11) [72]. There are only two differences: 1) the η parameter is replaced by $x_{i,j}(t)$; and 2) the authors use the velocity equation from Canonical PSO.

The vector differential operator is incorporated in the PSO velocity equation to boost exploration capability [73]. The vector differential operator originates from differential evolution (DE). In addition to the new PSO velocity, three conditions are proposed in the new PSO algorithm (PSO-DV) to help swarm to stay out of local minima

and to encourage the swarm to continue searching for potential solutions. Details of PSO-DV algorithm can be found in [73].

4.4.3 Neighborhood Topology

Neighborhood topology is known as sociometry or topology of a swarm. The idea of employing neighborhood topology is related to how the particles in the swarm are connected with each other in terms of sharing their knowledge, e.g., best position. Usually, the convergence rate can be estimated by calculating the average distance between two particles in the neighborhood topology. Shorter average distance facilitates quick convergence speed, also resulted by lower degree of connectivity. The most basic topology is the *gbest* in the social component, which all the particles are connected in such that the knowledge is shared by all particles. If need to implement other neighborhood topologies, for example star topology, then the *gbest* is replaced by star topology. In Suganathan's modified PSO algorithm, the local best, *lbest* replaces *gbest* in the social component [74]. The local best is the best solution in the neighborhood. The neighbors of a particle are selected via Hamming distance in each iteration count. The neighborhood size corresponds to the increment of the iteration count. By doing so, the particles start with random search in early stage. Slowly, as iteration count increases, the particles will connect to a larger number of neighbors (i.e., larger neighborhood size), in order to concentrate more in localized search until they landed on the optimum solution.

The common neighborhood topologies are the global topology (*gbest*), ring topology (*lbest*), and star topology [75]. Figure 4.4 illustrates the common neighborhood topologies. In global topology (*gbest*), all particles in the swarm are connected to each

others. The movement of particles is influenced by the best particle in the swarm. Due to this characteristic, global topology tend to converge fast but increases potential for the swarm to be trapped in local optima. The particles in ring topology (*lbest*) are connected to their neighbors. If a particle locates a better solution, only its immediate neighbors are drawn towards that particle. Then, one of the immediate neighbors will pull its neighbors towards the direction where the better solution is located. The process repeats at every iteration until the optimum solution is located. Although the convergence process is much slower than global topology but ring topology favors exploration and has tendency of finding the optimum. Refer to Figure 4.4 (c), all particles are connected to a central particle for star topology. The central particle has the highest influence than the rest. When one of the particles (not central particle) finds an optimum solution, it only draws the central particle closer to its direction, which is the only attached particle. In the next iteration, the central particle will influence the rest of the particles. This topology converges slower than that of the global topology but faster than ring topology. Occasionally, it will converge towards the suboptimum solution (local optima).

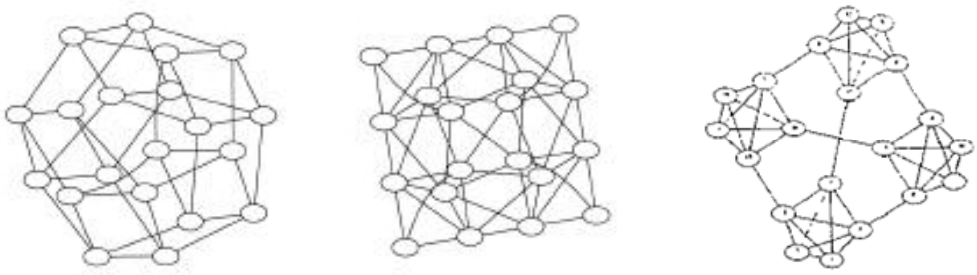
There are other topologies that are illustrated in Figure 4.5. Their movements associate with the connectivity, as described previously for ring and star topologies. The Von Neumann topology looks like square lattice connect to other square lattices around it. Pyramid topology has small three dimension wire frames connected together and four clusters topology consists of four small clusters that are interconnected with each other via simple connections. Unlike the common topologies listed in Figure 4.4, these topologies are harder to identify which ones give better performance, unless carefully crafted experiments on topologies influence on PSO performance are conducted. There

are studies that investigate the influence of topology to canonical PSO [76], both canonical and fully informed PSO [77], and Guaranteed Convergence PSO (GCPSO) [78].

An interesting way to implement the neighborhood topologies is suggested in [79] Instead of using the topologies in Figures 4.4 and 4.5, the authors proposed a randomly generated neighborhood topology and at a fixed amount of time, the neighborhood topology is re-structured into a new randomly chosen neighborhood topology. Simulation results showed probabilistic re-structuring neighborhood topology produces best results compare to the selected PSO algorithms on the chosen benchmark functions. Key reason of producing best results is because the diversity maintenance is enhanced by constantly applying different neighborhood topologies during the optimization process.

(a) **(b)** **(c)**

Figure 4.4 Graphical representation of the three common neighborhood topology [75,76]: (a) Global topology (*gbest*), (b) Ring topology (*lbest*), and (c) Star topology.



(a) **(b)** **(c)**

Figure 4.5 Graphical representation of the other neighborhood topology [75,76]: (a) Von Neumann, (b) Pyramid, and (c) Four Clusters.

4.4.4 Multiple-swarm Concept in PSO

PSO is designed in such the particles act a like swarm and quickly locate the optimum solution. The swarm-like characteristic renders PSO aptness to adopt the "subpopulation" framework. Research in fusing the multiple-swarm concept into PSO is well established in solving single objective and multimodal problems. Within this field, the motivation in adopting multiple swarms in the design of PSO is categorized into three main groups: solving multimodel problems, tracking all optima for multimodal problem in dynamic environment**,** and improve performance of SOPs by promoting exploration and diversity.

4.4.4.1 Solving Multimodal Problems

One motivation is utilizing multiple-swarm approach to solve multimodal problems. A multimodal problem consists of many local and global solutions. As mentioned earlier, the swarm-like characteristic in PSO has rendered its ability to rapidly solve optimization problems. Built on this, a new idea has emerged, which is utilizing this characteristic to locate multiple solutions in a given multimodal problem.

Several methods have been proposed. Brits *et al.* [80] adopted the niching (speciation) techniques into PSO to locate multiple solutions in a multimodal problem, and referred to this algorithm as niching particle swarm optimization (NichePSO). In NichePSO, the sequential niching algorithm and "partitioning criteria" are used as the indicator to form multiple subswarms from the main swarm. If any existing subswarms "belong" in the same niche they are merged together, and if any particles from the main swarm fall close to a subswarm, the subswarm can absorb these particles. Simulation

results show NichePSO can effectively handle high dimensional multimodal problem. Bird and Li [81] proposed an enhanced version of Speciation-based PSO (SPSO) [82], known as the ESPSO. In ESPSO, instead of using a predefined radius to form a species (particles that are within the species radius of the better particle), time-based measure and particles' personal best are used as an indicator to identify the species. Simulation results showed the performance of ESPSO has improved and species radius does not affect the performance. Another algorithm (multi-species particle swarm optimizer (MSPSO)) [83, 84] adopted a similar concept as [81,82]. Passaro and Starita proposed using a standard clustering algorithm to identify the niches in the swarm population and then restricting the neighborhood of each particle to the other particles in the same cluster [85]. By restricting the neighborhood, particles can perform local search within the cluster, which may discover any local minima located within the clusters. To save computational time, clustering procedure is only performed at a predefined interval. Seo *et al.* [86], in their multigrouped particle swarm optimization (MGPSO) suggested searching for N solutions of a multimodal function with a predefined parameter N, the number of groups. The repulsive velocity component is added to the particle updating equation, which will push the intruding particle out of the other group's global best territories (radius). In addition, the time-varying territory concept is proposed to allow the predefined radius (territory) to increase linearly during the search process to avoid several groups from settling on the same peak. Several benchmark functions are tested and MGPSO shows promise in solving multimodal problems. Zhang *et al.* [87] introduced a novel adaptive sequential niche particle swarm optimization (ASNPSO). Penalty function is adopted to modify the fitness values of the particles to control their influence within their subswarms in order to

prevent all subswarms from converging to optima. The uniqueness of this algorithm is that no niche radius is needed to define the "territory" of the subswarms. Experimental results show ASNPSO is efficient in finding all solutions for the selected test functions.

4.4.4.2 Tracking All Optima for Multimodal Problems in Dynamic Environment

Recently [88-89], multiple-swarm PSO was designed to locate and track the optima of a multimodal problem in a dynamic environment. Under the dynamic environment, the locations of all assumed optima of a multimodal problem change frequently. Hence, it is necessary to constantly track the optima.

Blackwell and Branke [88] proposed a multiswarm algorithm that comprises subroutines such as exclusion, anti-convergence, and PSO updating rules to balance the multiple swarm interaction. Extensive experimental studies have shown the multiswarm algorithm is robust and outperforms the selected approaches on the same benchmark functions. Parrott and Li [89] proposed an extended SPSO, named the dynamic SPSO (DSPSO), to locate multiple optima in the dynamic environment. Two modifications of SPSO are devised: 1) to compare the fitness of each particle's current local best with its previous record to continuously monitor the moving peaks; and 2) to use a predefined species population size to quantify the crowdedness of species and extra particles before they are reinitialized randomly in the solution space to search for new optima [89]. Simulation results showed that DSPSO is able to track the optima of a given test function at different levels of dynamism under a dynamic environment.

4.4.4.3 Promoting Exploration and Diversity

For some studies, multiple-swarm PSO has shown to improve the performance of PSO by promoting diversity. PSO move as a swarm while finding optima in the search space. This resulted in lack of diversity since a swarm tends to move in one group towards the same direction. If we break a swarm into multiple swarms, then the collaborated effort of multiple swarms exploring different regions in the search space to look for better solutions simultaneously. This will increase the chances of finding quality solutions efficiently.

Kennedy [90] adopted a clustering algorithm to cluster the swarm population into a certain number of clusters. Then, a particle's local best is replaced by their cluster center and the particles' global best replaced by the neighbors' best. By clustering the particle swarm population, the diversity and exploration of PSO has improved, effectively enhancing PSO performance. A cooperative particle swarm optimizer (CPSO) employs cooperative behavior among multiple swarms to improve the performance of the PSO [91]. The whole idea is to divide the decision variables into multiple parts and assign different parts to different multiple swarms. These swarms will optimize the different parts of the decision variable [91]. The authors stated that the reason for CPSO to show significant improvement compared to other PSOs is due to the increased diversity of the cooperative swarms. A new PSO, TPSO, was proposed by Chen and Yu [92]. In TPSO, the population is divided into two subswarms. One subswarm will optimize following the global best position, while the second subswarm will move in the opposite direction. The updating particles' positions are dependent on their local best, their corresponding subswarm's best, and the global best collected from two subswams.

If the global best has not improved for 15 successive iterations, the worst particles of a subswarm are replaced by the best ones of the other subswarm and the subswarms switch their flight directions. On the contrary, to improve the diversity of the particles, [93] developed a multi-population cooperative optimization (MCPSO). MCPSO is based on the concept of master-slave mode, where the swarm population will have a master swarm and multiple slave swarms. The slave swarms will explore the search space independently to maintain diversity of particles, while the master swarm will evolve via the best particles collected from the slave swarms [93]. Literatures [94-96] share the common spirit, where they emphasize in developing information exchange strategies within two or more swarms to enhance the diversity of the PSO. For example, in [94] two subswarms are updated independently for a certain intervals, and then the best particles (information) in each subswarm are exchanged. This procedure is repeated till the stopping criterion is met. In [95], four additional methods of information exchange are applied to investigate the improvement from the original design in [94]. Another method by [96] developed an algorithm to solve multimodal functions. The difference of this algorithm compared to [80-87] is locating the global optima only. The algorithm has two routines. Initially, swarm population is clustered into a predefined number of swarms. In the first routine, particles' positions are updated using their proposed PSO equation where three levels of communications are facilitated, i.e., personal level, global level, and neighborhood levels. At every given iteration, the second routine is activated where the particles in a swarm (subswarm) are divided into two sets; one set of particles (send list) is sent to another swarm, while the other set of particles (replacement list) will replace the individuals in other swarms [96]. This routine supports the diversity via exchanging the

particles between swarms, which prevents the particles from falling to the local optima. Despite some of the literature presented here, [90-96] indicates that multiple-swarm contribute to maintenance and enhancement in diversity.

4.5 Other PSO Variations

The above covers the various efforts to improve the performance of the standard PSO. Those that are discussed include tuning of the parameter setting in PSO equations, various designs of neighborhood topology, modification in PSO equations, and integration of "subpopulations" concept into PSO algorithm.

Other PSO variations emerge from incorporation of bio-inspired mechanism, and incorporation of concepts from other fields. For instance, Niu *et al.* proposed a novel PSO (PSOOFT) that integrate two mechanisms of optimal foraging theory (OFT) [97]. The two mechanisms are the reproduction strategy and path-choice based scheme. In the reproductive strategy, swarm is divided into two groups. The first group consists of "healthy" particles, i.e., lower fitness for minimization, and the second group is the "unhealthy" ones with higher fitness. Those that are unhealthy will die and replaced by the duplication of healthy particles. The underlying idea is to place more pressure on the particles in finding good solutions within a lesser time frame. Path-choice based scheme aims to provide balance between exploration and exploitation. All healthy particles will either conduct local search within their neighborhood of their *pbest* position or "migrated" to a new region. Their search behavior depends on a probability value, which is similar to mutation operator. In comparing their PSOOFT with the standard PSO, this novel PSO shows superior in both solution quality and convergence rate. A new PSO is

introduced based on the theory from the quantum delta potential well model [98]. Detail mathematical derivations and proofs are published in [98]. An interesting publication incorporated some psychological factor of emotion into the standard PSO [99]. The authors proposed the following concepts: Each particle has two emotions, which are joyful and sad. The emotional state of the particles is based on the emotional factor, which is compared with a randomly generated value. If certain condition is met, then the particle is updated using the "joyful" velocity equation or else "sad" velocity equation will be applied. Psychological model is incorporated in both "joyful" and "sad" velocity equations. Particle with "joyful" behavior tend to be more vibrant and will exploit both *pbest* and *gbest* experiences to determine its velocity. On the contrary, "sad" particle prefers local search, which represents its depressed behavior. Simulation results show their proposed PSO (EPSO) is better and more efficient than the standard PSO.

There are many other variations. This chapter only introduces those areas that are already established. In the following chapter, the PSO variation designed to deal with MOPs is discussed.

CHAPTER 5

MULTIOBJECTIVE PARTICLE SWARM OPTIMIZATION (MOPSO)

Success in solving various single objective and multi-model problems has shown the efficiency of PSO. Additional benefit is PSO's simplicity in implementation. In addition, the use of evolutionary algorithm in finding the Pareto front of MOPs has became very popular in recent years. Researchers have pushed the boundary of PSO by shifting the research direction towards designing new PSO algorithms in order to deal with MOPs. This chapter presents the introduction of multiobjective particle swarm optimization (MOPSO). General framework and literature reviews of recent works in developing MOPSOs are also included.

5.1 Particle Swarm Optimization Algorithm for MOPs

Previous chapters have reviewed some population based stochastic algorithms in the field of computational intelligences including evolutionary computation and swarm intelligence. Increment in number of publications in the areas of evolutionary algorithms (EAs) and PSOs show the gaining of research interest in the computational intelligences community. Successful implementations of EA and PSO in various types of problems have also contributed to the popularity [100-104].

There are appreciable differences in the design procedures of how PSO and a typical EA search for solutions in the decision space. For a typical EA, the individuals in the population searches for good solutions in the search space in every generation. Selection process is applied to favor individuals that represent the best solutions. These selected individuals undergo crossover operation to generate new individuals that inherit parts of the best solutions, while mutation operation is applied occasionally to introduce diversity to the population. On the other hand, as discussed in Chapter 4, PSO relies on the two equations to guide and advance the particles to the best solution.

Although the optimization process is different for PSO and EA, there are similarities between the techniques employed to support the procedure and concepts. Table 5.1 presents the comparison between the terminologies and techniques for EA and PSO. The terminologies of swarm and particle for PSO bear similar representation for EA's population and individual respectively. Fitness evaluation is the same for EA and PSO. Similar form of crossover operation can be found in the PSO velocity equation (Equation 4.1), in which mixing and exchanging information among particles occur in the cognitive and social components to adjust the equation (i.e., difference between *pbest* and current particle position, and difference between *gbest* and current particle position). The random numbers in the PSO velocity equation act like the mutation operator to introduce diversity to the swarm.

Table 5.1 Comparison between a typical EA and PSO.

Evolutionary Algorithms (EA)	Particle Swarm Optimization (PSO)
Population	Swarm/Swarm Population
Individual	Particle
Fitness	Fitness
Crossover operation	Cognitive and Social Components
Mutation operation	Random numbers
Selection of best individuals	Select the best knowledge (position)

Beside the common stochastic search mechanism similar to EAs, PSO has rapid convergence capability that falls short in many EAs [25-28]. Due to these reasons and its simplicity in implementation, PSO has recently been extended to deal with multiple objective optimization problems (MOPs). During the past few years, many publications are focused on how to modify PSO to handle multiple objective optimization problems [105-122]. PSO equipped to deal with MOPs is generally regarded as multiobjective particle swarm optimization (MOPSO).

5.2 General Framework of MOPSO

Within these few years, extensive researches in modifying and extending PSO to handle MOPs have shaped the conceptual framework for multiobjective particle swarm optimization algorithms (MOPSO) [105-122]. The MOPSO framework comprises features that are designed to retain the originality of PSO algorithm and to deal with dilemmas of typical Pareto optimizer, i.e., the two conflicting goals of achieving convergence and maintaining diversity to produce a well-distributed Pareto front. Figure 5.1 presents the generic framework of MOPSO. The framework is similar to the original PSO algorithm yet added a Pareto dominance scheme to assess the particles' dominance relationship and their current status in the objective space. Those features, represented by shaded boxes and bold face, are the modifications made in the original PSO. These features are now the established research areas mainly to improve MOPSO capability in dealing with the two conflicting goals. Research areas include global leader and personal best selection to improve convergence and to promote diversity [111-119], the introduction of external archive to record all nondominated candidates found during a

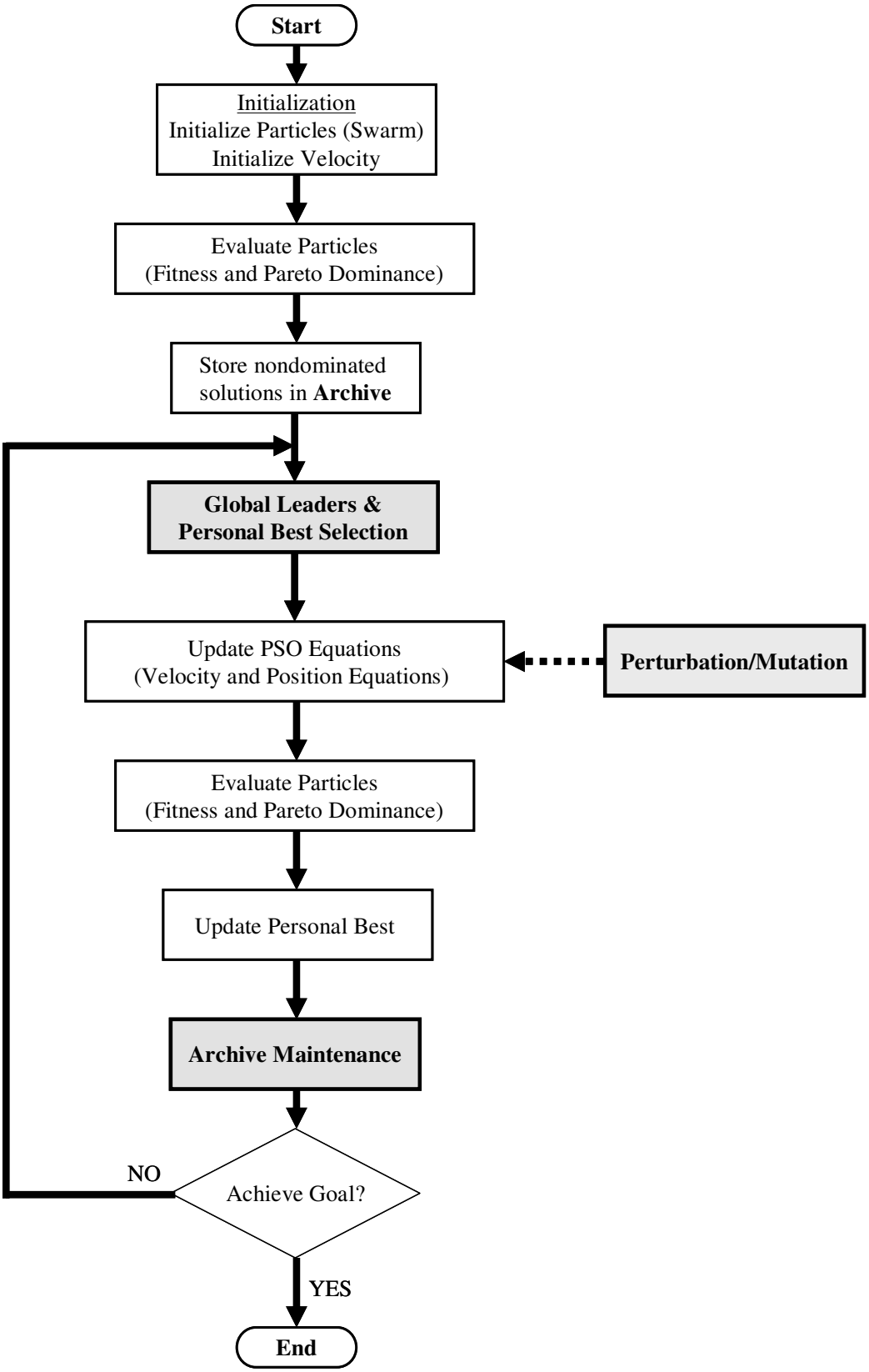


Figure 5.1 Generic framework of MOPSO algorithm.

search process and ways to maintain it [105-110], and incorporation of genetic operators such as mutation and perturbation to enhance the exploration capability [120-121]. Recently, successful implementation of multiple-swarm concept in PSO and subpopulation concept in MOEA encourage new area of integrating multiple-swarm concept in MOPSO formulation [122]. Relevant works of these areas are reviewed in the following sections.

5.2.1 External Archive

The introduction of the notion of elitism in the evolutionary algorithms is called the second generation evolutionary algorithms [2]. These EAs are briefly discussed in Chapter 2.2.3.2. Archiving is also known as elitism for other EAs. In PSO, each particle's flight in a search space is determined by its velocity computed from Equation (4.1). Especially for complicated MOPs, it is difficult to control the velocity of each particle to perform its optimal flight in a high-dimensional search space. Hence, for a typical MOPSO, external archive is often used to record any good particles found in each iteration.

Hu *et al.* extended their previous algorithm, called dynamic neighborhood PSO (DSPSO), by adding an extended memory or archive to record all nondominated solutions found in every iteration [105]. Fieldsend and Singh [106] proposed the use of unconstrained archives to overcome the inefficiency caused by truncation of constrained Pareto archives. They developed a data structure approach known as dominated tree to maintain the unconstraint archives. The dominated tree consists of a list of composite

points, and each composite point is a vector of archive members. These composite points are ordered by weak dominance relation [31].

Recently, Mostaghim and Teich adapted the ε–dominance method to control the archive size and help to reduce computational cost [107]. To update the archive, ε–dominance criterion is employed to evaluate the dominance relationship between the current particles in the swarms and the archive members. Dominated archive members or dominated particles in the current swarm are deleted, otherwise the archive members remain in the archive or the new particles are considered to join the archive. An upper bound equation is used to control the archive size. The equation indicates that the archive size depends on the upper bound of the objective values and the user-defined ε value. Coupled with other techniques such as sigma method for finding global leaders and turbulence factor to enhance exploration, experiment results shows applying ε–dominance reduced computational time and improved the quality when compared with MOPSO that used clustering techniques [128]. In [108], the archive maintenance is based on enhanced ε–dominance since the drawback of ε–dominance method is losing solutions near the boundary. The concept of deleting archive members and accepting new members is the same as what is reported in Mostaghim and Teich approach [107] except that enhanced ε–dominance is employed to evaluate the dominant relationship. If neither archive members nor current particles dominate each other, Euclidean distance between particle and the vertex of ε grid is used to decide if any current particles are accepted into the archive. The particle that has the smallest distance is accepted. The authors explained that this technique influences the production of well distributed solutions, which is validated by their simulation results.

Li proposed a different approach, known as the nondominated sorting particle swarm optimizer (NSPSO), in which the algorithm of NSGA-II is adopted in PSO design [109]. Rather than having a separate external archive, the offspring and previous population are joined to form an overall population (with twice the population size). The nondominated sorting is then applied to the overall population. Only the top portion of population is selected for the next iteration. For diversity preservation, niche count and crowding distance assignment are applied to guide the particles' selection of global leaders from the nondominated solution list.

Another prominent work was contributed by Coello Coello and Lechuga [110]. They proposed a multiple objective particle swarm optimization algorithm that incorporates the concept of Pareto dominance and adopts archive controller, which decides and stores the membership of new nondominated solutions found in each iteration. The deciding factor of accepting a new membership depends on the cases listed below:

Case 1: If the archive is empty (empty set), any new nondominated solutions are stored in the archive (Figure 5.2, case 1).

Case 2: If there are members in the archive, then the Pareto dominance relationship between a new solution and all archive members is evaluated. If new solution is dominated by all the members in the archive, it is rejected from membership (Figure 5.2, case 2).

Case 3: If a new solution and all archive members are not dominating each other, the new solution is accepted (Figure 5.2, case 3).

Case 4: If a new solution dominates some of the archive members, the new solution is accepted and those archive members that are dominated by the new solution are automatically discarded (Figure 5.2, case 4).

Case 5: If a new solution and all archive members are not dominating each other, the new solution is still accepted even if the archive size has reached its maximum limit. At this time, the adaptive grid procedure is evoked to readjust the grids and the hypercube size in order to fit in all the archive members and the newly accepted solution. Refer to Figure 5.2 (case 5), let's consider the maximum archive size is 6, a new solution is accepted. The figure shows that the grid is adjusted while the number of hypercubes is retained. After the adjustment is over, to maintain archive size, those archive members that located in the most crowded hypercubes in the objective space will be removed.

An adaptive grid feature based upon objective function values of archive members is applied to the archive with the goal of producing a well-distributed Pareto front. The key mechanism of the adaptive grid is to be able to adjust and recalculate the grids in the archive whenever the archive is updated. The adjustment of grid is essential to maintain uniform hypercubes formed in the archive. Archive coupled with grid feature allows global leaders to be selected from the archive via fitness sharing and roulette wheel selection.

5.2.2 Global Leaders Selection Mechanism

Global leader (*gbest*) selection mechanism is one of the key modifications to basic PSO to solve for MOPs. Particles in the population converge in a swarm-like

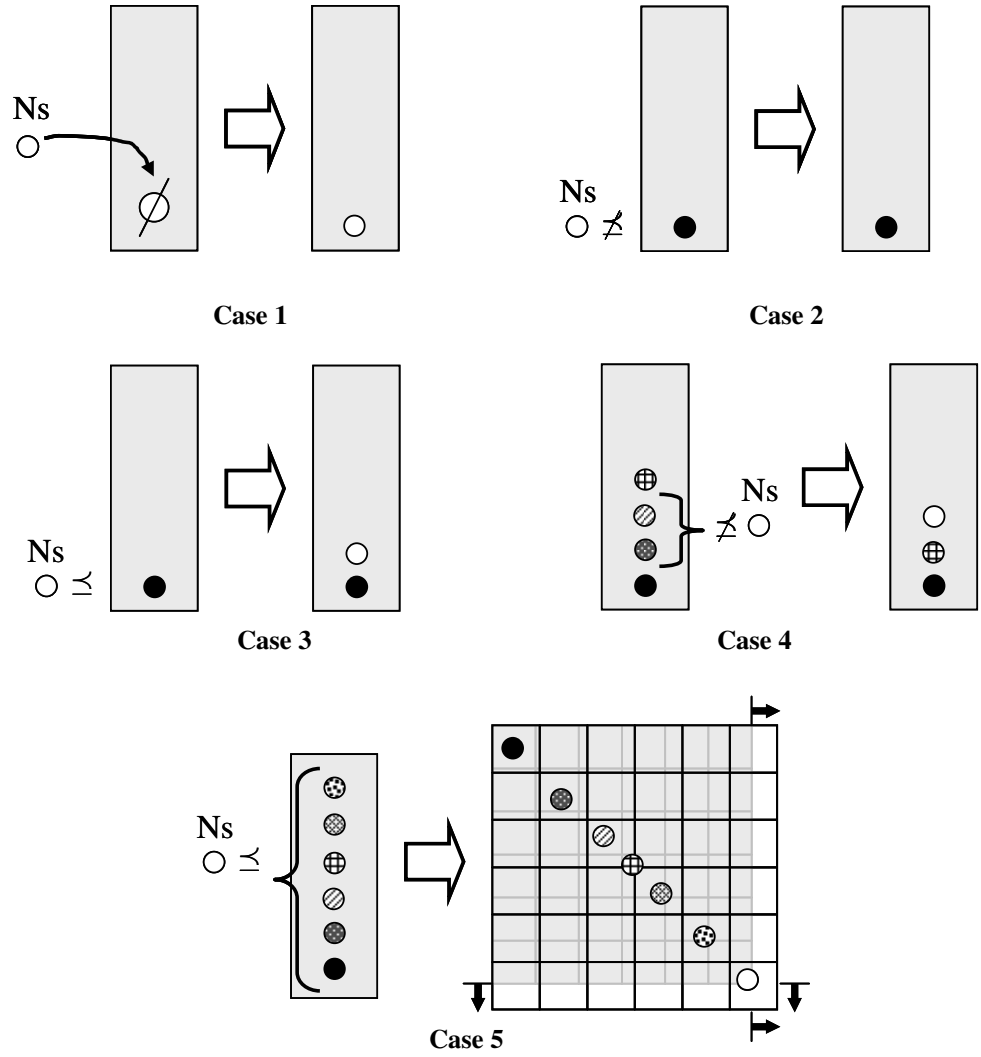


Figure 5.2 Possible cases presented in [110]. Note that Ns denotes as nondominated solution; a no filled circle represents a new nondominated solution; and a filled or patterned circle represents a archive member.

manner toward an optimal solution by following their best leader or the best particle within the population. However, for solving MOPs, the goal is to search for a set of Pareto optimal solutions. Hence, the particles in the population must follow a set of candidate best leaders that will lead them toward the optimal solutions. The selection design must balance between diversity preservation and faster convergence.

In recent works, Hu and Eberhart [111] proposed a dynamic neighborhood PSO (DSPSO) that includes three criteria: dynamic neighbors, new global best particle updating strategy, and one-dimension optimization. The scheme of selecting the global leaders involves the particles' neighborhood, which applies similar concept as the star topology (*lbest*) but the concept is modified to solve for MOPs. In every iteration, each particle finds its best new neighbors via calculating the distances from other particles of the first objective function (x-axis) in the objective space only. Then, the global leader to be assigned to each particle is the closest neighbor within its neighborhood. The number of neighbors is predetermined. Among the solutions in the neighborhood, each particle finds the local best particle, as the global best. Figure 5.3(a) illustrates an example of dynamic neighborhood procedure. The particles in the swarm are presented as circle (no filled) while the filled circles represent the nondominated solutions found in current iteration count. In this example, the number of nearest neighbors in the neighborhood is set to three. The neighborhood for particle A is presented in grey region. The arrows indicate the selected global leader from the particle's neighborhood. Notice in Figure 5.3(a), since the distance measure is based on the first objective, the selection of global leaders tends to point towards the downwards direction. Each particle's personal best (*pbest*) is updated only when these personal best history is dominated by a new solution. However, dynamic neighborhood PSO is limited in dealing with a small number of objective functions.

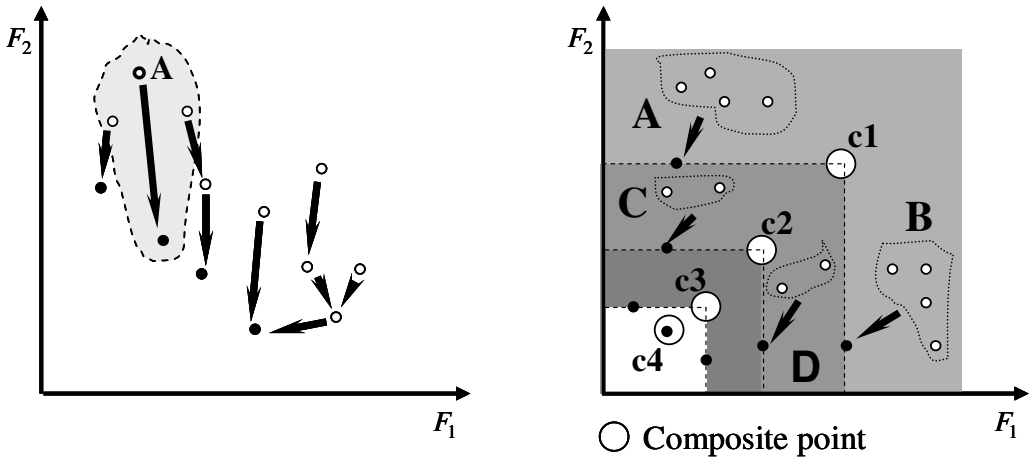
Another work by Zhang *et al.* [112] has suggested a selection scheme for global and local leaders to improve the MOPSO performance. The scheme involves computing the new leaders via the proposed equation that depends upon each objective function and

the current iteration, and deciding whether the particles should follow their leaders based upon the proposed criteria. Nonetheless, this approach did not provide a generalized equation to determine the new global leaders for problems with high-dimensional objective functions. Also, the new global and local leaders depend solely upon the global best and local best values that correspond to each objective function, which inadvertently may result in premature convergences if global best and local best corresponding to each objective function are very close.

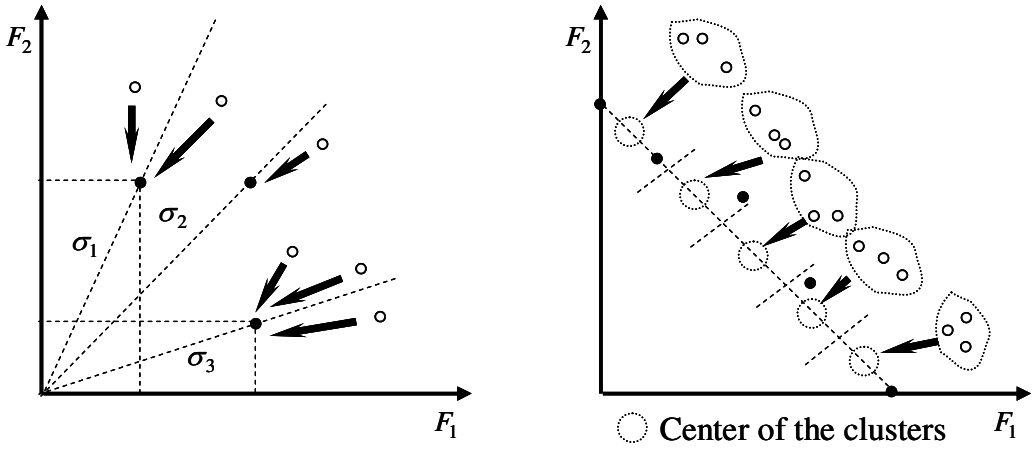
Fieldsend and Singh make use of their proposed archiving approach, i.e., dominated tree, for the global leader selection [106]. After the archive members (leaders) are ordered by weak dominance relation, with the coordinates of these composite points and the particles' locations, the particles can select their global leaders based upon their closeness to the archive members. Hence, the global leaders are the archive members whose fitness values of the composite points c_j contributing to the vertex are less than or equal to those of the particles. However, this approach restricts the particles' chances of selecting global leaders that belong to other composite points, even if some particles may provide better guides. Figure 5.3(b) presents Fieldsend and Singh's method of global leader selection for each particle in the swarm. In the figure, the composite points are label as $\{c_1, c_2, c_3, c_4\}$ and represented by a larger circle. The shaded regions show the areas that are separated or bounded by two composite points. Note that particles in the swarm are presented as circle (no filled) while the filled circles represent the archive members. The arrows show the particles select their closest archive member of composite point c_j as their global best leader.

Inspired by the Fieldsend and Singh's method in [106], Mostaghim and Teich [113] introduced the sigma method to search for the global best for each particle in the population. The particles select their global leaders based upon the minimum distance from the sigma values computed for all archive member and all particles in the swarm. Figure 5.3(c) shows the sigma method for two objective functions. The sigma values of the archive members are presented as $\{\sigma_1, \sigma_2, \sigma_3\}$, since only three archive members (filled circle) are shown. The arrows indicate the global leaders selected by the particles (no filled circle). In this report, this algorithm is referred to as sMOPSO. The sigma method can impose selection pressure on global leaders and since PSO has rapid convergence capability, this may lead to premature convergence for some MOPs. In a follow up work [114], combination of hybridizations of three global leader selection strategies, i.e., sigma, centroid, and random methods, for MOPSO are investigated. Four benchmark functions are selected and hypervolume indicator is used as performance metric. In general, simulation results indicate that hybrid selection strategies improved the diversity and convergence in MOPSO although no specific hybrid selection strategy dominated the performance since it is problem dependent. The authors suggested future research on hybrid selection strategies is needed.

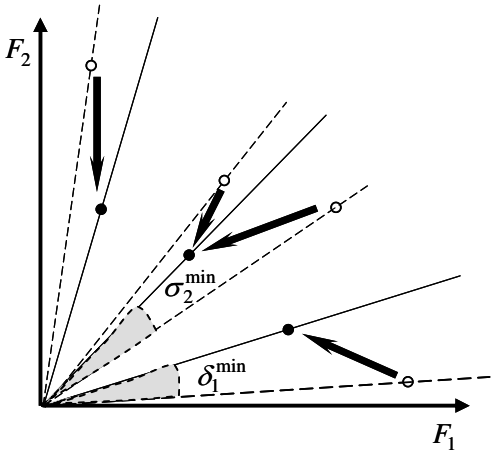
A new selection mechanism based on the idea of stripe is proposed to maintain diversity of MOPSO while solving for MOPs [115]. First, the maximum fitness value of an individual objective function in the archive is identified. Once the maximum fitness values of all objective functions are known, a "line" is estimated using these values. Figure 5.3 (d) illustrates the "line" drawn across the two objective functions. Then, the number of points, also known as the stripe centers, is uniformly distributed along the



(a) Dynamic neighborhood [111] **(b) Fieldsend and Singh's method [106]**



(c) Sigma method [113] **(d) Stripes method [115]**



(e) Minimum particle angle [116]

Figure 5.3 Figures depicting the different strategies of selecting the global leaders. The arrows indicate the global leaders (filled circles) selected by the particles in the swarm (circles (no filled)).

"line". By setting the maximum number of particles for each strip will provide distribution of particles in several stripes and to avoid excessive clustering in any particular stripes. The authors used the notion of clustering as stripe centers. Figure 5.1 (d) shows an example of stripes divided based on the predetermined number of stripes. Note that a maximum number of three particles are set for each center of cluster. This strategy showed impressive simulation results when stripe method is applied to MOPSO, ε-MOEA and NSGA-II. However, the authors applied their proposed strategy to test problems with only two objective functions. For higher number of objective functions, stripe method may be difficult to implement.

Gong *et al.* [116] introduced a strategy that is similar to the sigma method. In their proposed strategy, the selection of the global leader for a particle is based on the minimum particle angle ($\delta_i^{\min}$) of a particle in the swarm and an archive member. To locate the minimum particle angle, particle angles (δ_i) between a particle in the swarm and all the archive members are computed. The particle angle is computed by applying the inverse cosine function to the dot product of two fitness vectors of two particles. Figure 5.3 (e) presents the graphical representation of the proposed strategy. In the figure, the particles in the swarm are represented by circle (no filled) and archive members are represented by the filled circle. Arrows show which archive member is chosen by a particle based on the minimum particle angle. Note that the authors define the particle's density as the number of particles that choose the same archive members. For example in Figure 5.3 (e), the particle's density for the second archive member (middle one) is two.

Three methods of selecting global leaders based on Pareto dominance are proposed by Julio *et al.* [117]. The following is the brief introduction of the three methods:

1. **ROUNDS**: This method relies on the number of particles that are dominated by a member of the archive to be selected as global leader. The member that dominated the least particles is assigned as global leader of those particles which it dominates. The procedure is repeated for other members in the archive. The idea is to bring the particles to explore regions with sparse population. Thus, this method aims to promote diversity in the swarm.
2. **RANDOM**: Each particle finds a set of archive members that dominate it. Then, it randomly selects a global leader from the set with equal probability. If any particles are not dominated by any archive members, then their global leaders are randomly selected from the entire archive.
3. **PROB**: This method uses the same way as the RANDOM method except global leader selection is based on the probability that favor those archive members that dominate least particles.

Experiments are conducted to compare the efficiency of the three methods and sigma method [113]. The methods are applied to the two benchmark functions. Pareto fronts, archive growth and histogram of the metric are presented to evaluate these methods. The authors concluded that RANDOM method yields solutions closest to the true Pareto front but lack of diversity; while PROB method gives the balance of both solution quality and diversity. Experiments are repeated with rescaling the objective functions of the two test

functions. Only PROB and sigma methods are tested. Results show that the performance of PROB method is unaffected by the rescaled objective functions, which indicates this method does not relies on objective space distances for global best selection.

5.2.3 Personal Best Selection Mechanism

In a typical MOPSO, particle's personal best (*pbest*) updating mechanism is based on the Pareto dominance relationship between the current personal best and the new solution [110]. If the new solution dominates the current personal best, then only the personal best is replaced by the new solution. Not until recently, some research works introduce different strategies for updating the particle's personal best.

Recently, two cases of personal best updating strategies are proposed by Gong *et al.* [116]. One of the cases involves the strategy discussed early (Figure 5.1 (e)). For the first case, the personal best updating is strictly based on the Pareto dominance relationship if either the current personal best ($\boldsymbol{pbest}_i$) dominates the new solution ($\boldsymbol{x}_i(t+1)$) or the new solution dominates the current personal best. The second case is executed if both current personal best and the new solution do not dominates each other. The steps are briefly given here: First, both $\boldsymbol{pbest}_i$ and $\boldsymbol{x}_i(t+1)$ select their archive members via particle angle approach. Next, the particle density is updated for all archive members. To maintain the number of particles in the swarm, one of the archive members whose particle density has reached the maximum number (user defined) is selected. Then, a particle is randomly picked to check the Pareto dominance relationship with a $\boldsymbol{pbest}_i$. If this randomly selected particle dominates a $\boldsymbol{pbest}_i$, then delete the $\boldsymbol{pbest}_i$, otherwise delete the randomly selected particle. If both do not dominate each other, randomly delete

one of them. Based on the authors observation, the process of updating personal best via second case will promote good distribution of particles in the objective space.

Branke and Mostaghim [118] investigated the influence of the personal best particles in MOPSO. There are total nine personal best selection concepts mentioned and are briefly presented:

1. **Oldest**: Always keep the old position in the personal best memory unless new solution ($\boldsymbol{x}_i(t+1)$) dominates the current personal best ($\boldsymbol{pbest}_i$).
2. **Newest**: Always keep the latest position in the personal best memory except for the case where a $\boldsymbol{pbest}_i$ dominates $\boldsymbol{x}_i(t+1)$.
3. **Sum**: The deciding factor to update either the $\boldsymbol{pbest}_i$ or $\boldsymbol{x}_i(t+1)$ depends on the sum of objective values. Whichever contributes better sum will be updated.
4. **Random**: Randomly select a nondominated $\boldsymbol{pbest}_i$ from the personal best archive.
5. **WSum**: Higher weights are assigned to particles that hold good solutions. This way, the selection of $\boldsymbol{pbest}_i$ will also contribute in diversity maintenance.
6. **Global**: To increase the convergence rate, the selected personal best, $\boldsymbol{pbest}_i$, is the closest to the global best in the objective space.
7. **Diversity**: To improve diversity, the $\boldsymbol{pbest}_i$ is chosen from those personal best in archive that are isolated from other personal best in the archive.
8. **All**: Several personal bests from the archive are selected as the represented $\boldsymbol{pbest}_i$.
9. **None**: All personal bests are replaced by the global best.

Empirical results showed keeping personal best archive and their proposed strategies in selecting personal best from the archive produces significantly better results than traditional approaches.

Another selection technique is introduced by Ho *et al.* [119]. Their idea is to counter the difficulty of selecting personal best (***pbest***) when a particle finds more than one possible Pareto solutions. They proposed two repositories: one to store the latest ***pbest*** while another to store age variables of the ***pbest*** , which are assigned to each member of ***pbest*** in the first repository and are accumulated based on iteration counts (i.e., the age factor). Selection of a particle's ***pbest*** from the repository is done via weighted sum of its age variables, its fitness values and roulette wheel selection scheme. When a specific ***pbest*** is selected, its age variable is reassigned to the minimum age value. Main idea of employing the age factor (age variables) is to improve diversity and provide opportunity for "unpopular" ***pbest*** being chosen in the next iteration. The authors proposed the same technique to select the global best (*gbest*) from its archive.

5.2.4 Incorporation of Genetic Operators

In recent works, the incorporation of genetic operators such as mutation and perturbation operators has greatly enhanced the exploration capability of MOPSO. Although the terminologies such as mutation and perturbation operators are the same for many publications but the mechanism are tailored to suit the proposed MOPSOs. Inspired by the stochastic variable used in [50], Fieldsend and Singh integrated the turbulence (perturbation) to the velocity equation (Equation 4.1) to extend the exploration capability for MOPSO [106]. The turbulence is a random variable and also known as the craziness

parameter, which provides unpredictability to the particles' flight in searching for good solutions in the search space. Mostaghim and Teich [107, 113] also implemented a turbulence factor to the particle's updated position. Decision to apply the turbulence factor depends on a turbulence probability. The turbulence factor adds randomness to the particle's updated position and it is represented by a random value within $[0,1]$. Similar to [106] and [107,113], Ho *et al.* [119] incorporated both the craziness operation and craziness probability to the velocity equation to promote diversity of their MOPSO. The craziness operation is equivalent to the turbulence or perturbation operator.

In [108], mutation procedure is added in the proposed MOPSO to deal with the problem of premature convergence. The execution of mutation procedure is based on a mutation probability, which is dependent on the iteration counts. Mutation operator is applied to the updated velocity value in the position equation (Equation 4.2). Parameters such as direction of mutation and mutation distance are included in the mutation operator. Coello Coello *et al.* [120] proposed the use of mutation operator to improve the exploration capability on the MOPSO presented in [110]. The authors defined the behavior of mutation operator to determine the number of particles in the swarm that affected by the mutation operator. The function that describes the behavior of mutation operator is also used to determine the mutation range imposed to a particle. The concept of the behavior is to allow more particles in the swarm affected by the mutation operator in early search process. The number of particles that is affected will slowly reduce as the iteration count increases until the mutation operator halts.

In addition, Sierra and Coello Coello [121] suggested a new MOPSO, also known as OMOPSO, based upon Pareto dominance and incorporated 1) crowding factor to filter

out the list of available leaders, 2) mutation operators for different subdivisions of swarm, and 3) ε–dominance to control the archive size. Their approach is to divide the population into three subswarms of equal size. Each subswarm adapted to a different mutation operator. In doing so, the ability of exploration and exploitation was enhanced during the search process. The proposed idea showed good performance compared to the existing evolutionary algorithm. Although genetic operators are adopted by MOPSO, the selection of an appropriate initial population size plays an important role in homogenously exploring the high-dimensional search space.

5.2.5 Incorporation of Multiple Swarms

Over the years, numerous works related to subpopulation manipulation in multiobjective evolutionary algorithms (MOEAs) have been published [122-127]. Though the concept of subpopulation seems generic, various studies show convincing performance in adopting the subpopulation concept from different perspectives. In [122], the population is divided into subpopulations of equal size along one of the objective functions, and these subpopulations evolve separately in a parallel fashion. At some interval generations, the subpopulations are gathered and evolve as a whole population. Then the population is divided and redistributed again along a different objective function. In [123], an improved design known as Parallel Strength Pareto Evolutionary Algorithm consists of two models: in global parallelization model, each subpopulation performs evolutionary procedure, and in island model, the subpopulations exchange information using the migration concept. In another design, a subpopulation is used to optimize one decision variable [124]. All parameters from the first subpopulation and the

best individuals from the rest of the subpopulations are combined to form complete solutions, which will be evaluated and used to update the archive. Unlike [124], Vector Evaluated Differential Evolution [125] proposed that each subpopulation is assigned to an objective function and information is shared among the subpopulations via migration of best individuals. Some publications emphasize the implementation of subpopulation to solve specific problems. For example, Ando and Suzuki [126] proposed a Distributed Multi-Objective Genetic Algorithm, which employs a multiple subpopulation approach and replacement scheme based on the information theoretic entropy, to improve the performance in solving deceptive problems; while Izumi *et al.* [127] reported promising results with their proposed Evolution Strategy (ES) wherein the arithmetical crossover is modified by using the subpopulation's elite and the mean strength of that subpopulation. These studies [122-127] have consistently shown that their proposed MOEAs have either improved the performance or resulted in a highly competitive design validated through selected test functions. More importantly, subpopulation concept coupled with other ingredients often yields more efficient and effective designs, especially in enhancing the population diversity.

As elaborated in Chapter 4, many publications on multiple-swarm concept are mainly to solve single objective and multimodal problems. This concept is still a new research area in MOPSO. In general, mutation operators are typically incorporated to boost the exploration capability of the MOPSOs. For those MOPSOs that have no built-in mutation operators, incorporating multiple swarms into MOPSO can effectively enhance the exploration. In recent work, Toscano Pulido and Coello Coello proposed a multi-swarm MOPSO, or simply cMOPSO, that implements the subdivision of the

decision variable space in multiple swarms via clustering techniques [128]. Their goal was to improve the diversity of solutions on the Pareto front. At some point during the evolution process, different subswarms exchange information as each subswarm chooses a different leader to preserve diversity. cMOPSO has shown promising results when compared to NSGA-II [31], PAES [30], and coello coello's MOPSO [120]. Another proposed MOPSO design applies multi-swarm concept to cover the optimal Pareto front at the final stage of the search process in order to improve the quality of optimal Pareto front [130]. During the initial run, MOPSO employs a restricted archive size that is controlled using ε-dominance strategy presented in [107]. Once archive members are closerer to obtain the optimal Pareto front, sigma method [113] is applied to group the particles in the swarm according to their selected global leaders from the archive. The groups of particles are referred to as subswarms by the authors. These subswarms converge towards and cover the optimal Pareto front using their global leaders as their guide. The archive size in covering MOPSO is not restricted. This algorithm shows excellent results in producing uniform distributed and well extended Pareto fronts with reduced computational time for the four test functions.

5.2.6 Other MOPSO Designs

Researches in developing MOPSOs are not limited to the areas that reviewed above. In recent years, publications show various research areas including redesign of PSO equations, incorporation of techniques, concepts, theory or model from other fields. Several MOPSOs with various designs are briefly introduced in this section, providing some information of current research trends.

Ho *et al.* [119] designed an improved velocity equation using adjusted random parameters to control the balance of global and local searches in their proposed MOPSO; another approach applied local search and clustering technique on MOPSO to improve convergence and maintain diversity [129]; and a co-evolutionary PSO, i.e. CMOPSO, implemented co-evolutionary concept and designed with co-evolutionary operator, competition mutation operator, and selection mechanism, is proposed by Meng *et al.* [131]. Besides incorporating techniques to MOPSO, Santana-Quintero *et al.* introduced a new MOPSO that employs some concepts from rough set theory to design a local search approach [132]. Their main objective is to produce well-spread and well-distributed Pareto front for MOPs. Another MOPSO design is the hybrid design of PSO and agent-environment-rules (AER) model [133,134]. This new MOPSO is called intelligent PSO (IPSO). The proposed use of AER model aims to provide appropriate selection pressure to encourage convergence towards the optimal Pareto front. Key designs that insert into IPSO to support the goal are competition and clonal selection operators.

CHAPTER 6

PROPOSED ALGORITHM 1: DYNAMIC MULTIOBJECTIVE PARTICLE SWARM OPTIMIZATION (DMOPSO)

This chapter presents the first proposed MOPSO that incorporate the dynamic population concept to manage the swarm population for solving MOPs in order to improve the efficiency of the algorithm and to address the need to "estimate" a fixed swarm population size sufficiently to explore the search space without incurring excessive computational complexity. The multiple swarm concept is also applied to the swarm population where the number of swarms is predefined. Hence, the swarm size, i.e., number of particles in each swarm, is not fixed. In addition, clustering algorithm is used to group the global leaders in the archive based on the predetermined number of swarms to provide guide in global leaders selections for each swarm and to promote diversity within the swarm population. This proposed algorithm is named the dynamic multiobjective particle swarm optimization (DMOPSO). Elaboration of the proposed strategies employed in DMOPSO is provided. In the final section of this chapter, performance metrics and benchmark test functions are used to evaluate the performance and the computational cost of the proposed algorithm compared to five state-of-the-art MOPSO algorithms.

6.1 Introduction

In Chapter 5, existing MOPSO were reviewed. However, all these MOPSO designs adopt the notion of using a fixed population size throughout the process of searching for possible nondominated solutions until the Pareto optimal set is obtained. Although some may argue that a good algorithm design would assure a high probability of finding the Pareto optimal set, yet population size does indirectly contribute to the effectiveness and efficiency of the performance of an algorithm. One influence of population size on these population-based metaheuristic optimization algorithms is the computational cost. If an algorithm employs an overly large population size, it will benefit from a better chance of exploring the search space and discovering possible good solutions but inevitably suffer from an undesirable and high computational cost. On the other hand, an algorithm with an insufficient population size may result in premature convergence or may obtain only some sections of the Pareto front. Again, one may suggest that heuristically estimating an "appropriate" population size may be adequate since one need not know the exact fitness landscape to solve a MOP. It would be the case for these MOPs that possess lower numbers of objective functions or lower dimensions in decision space. Considering the MOPs that have large numbers of objective functions or large dimensions in decision space, and even of those MOPs qualified as the hard problems [152], this will pose a great challenge to "estimate" the population size to solve these MOPs without exerting a high computational cost. In addition, without a prior knowledge about the contour of the fitness landscape in a MOP, it might be unrealistic to estimate an "appropriate" population size to kickoff the search process. Hence, a compromised, yet effective, solution would be dynamically adjusting the population size

to explore the search space in balance between computational cost and the attained performance.

In fact, there are few publications that tackle the issue of population size. In earlier work, several methods of determining an optimal population size in Genetic Algorithm (GA) are proposed to solve SOPs [136-138]. Nonetheless, preservation of population diversity is not an issue in solving SOPs but not the case if solving MOPs [139]. Several published works have incorporated dynamic population strategy into EAs. Tan *et al.* [135] proposed an incrementing multiobjective evolutionary algorithm (IMOEA) that adaptively computes an appropriate, but conservative, population size according to the online evolved tradeoffs and its desired population distribution density. Although IMOEA demands a heuristic approach to estimate the desired population size for the next generation, simulation results show that IMOEA can perform better than several state-of-the-art MOEAs. Another algorithm, dynamic population-size multiobjective evolutionary algorithm (DMOEA), is proposed by Yen and Lu [139-140]. This algorithm includes a population growing strategy that is based upon the converted fitness and a population declining strategy that resorts to the three types of qualitative indicators: age, health, and crowdness. From the simulation results, the performance of DMOEA is competitive or even superior to the selected MOEAs. In addition, robustness study shown that the population size always converges to an optimal value independent of the tuning parameters chosen and the complexity of the Pareto front [140]. Eskandari *et al.* also proposed a GA-based stochastic multiobjective optimization technique to obtain the Pareto optimal set for simulation models in a computationally efficient manner [141]. They introduced a few features into their algorithm: a new ranking scheme that

bases on the stochastic dominant concept, a new genetic operator (blended crossover operator and non-uniform mutation operator), dynamic expansion operator to increase population size, and an importation operator to explore new regions of the search space.

In this study, the goal is to incorporate dynamic population size into a MOPSO since particle swarm optimization (PSO) has an advantage over evolutionary algorithm, in which, PSO has a rapid convergence capability. However, PSO often faces the problem of premature convergence. Hence, multiple-swarm MOPSO is employed to promote diversity within the swarm population to deal with the problem of premature convergence. In this chapter, the proposed MOPSO design involves two key concepts, which are dynamic swarm population size and multiple swarms. Design aspects that are incorporated in the proposed MOPSO include 1) strategy to facilitate access the status of the particles when the swarm population size varies ; 2) strategies to dynamically adjust the swarm population in order to provide the needs of computational resources at different stages and, at the same time, to promote the competition among the swarms so that convergence toward the optimal solutions and the diversity characteristics are preserved; and 3) adaptive local archive procedure to promote diversity within each swarm.

6.2 Proposed Algorithm Overview

Discussed in Chapters 4 and 5, PSO poses two unique characteristics that particles tend to move as a swarm and converge quickly toward the Pareto front. Both characteristics may present a problem when encountering complex MOPs in which the Pareto fronts may be composed of a set of solutions located at the disconnected segments

in the decision space. For such cases, the movement of particles as a single swarm and the fast convergence property may lead the swarm population toward only a segment of the Pareto front. To deal with this problem, Toscano Pulido and Coello Coello proposed the multiple swarms MOPSO (cMOPSO) [128]. Inspired by their work, the skeleton of DMOPSO is built on cMOPSO. In addition, DMOPSO incorporates four proposed strategies: 1) cell-based rank density estimation scheme to keep track of the rank and density values of the particles; 2) population growing strategy to increase the population size to promote exploration capability; 3) population declining strategy to prevent the population size from growing excessively; and 4) adaptive local archives design to improve the distributed solutions along the sections of the Pareto Front that associate with each subswarm. Figure 6.1 presents the pseudocode of DMOPSO involving four newly developed strategies highlighted in boldface.

The generic steps of DMOPSO are as follows. First, based upon a preset number of subswarms, every subswarm of particles is initialized and cell-based rank density estimation scheme is applied to initialize the rank and density values of the particles. Second, the group leaders of each subswarm are determined by the rank matrix. Third, all of the group leaders from subswarms are collected to form the set of global leaders ($Gleader$). A clustering algorithm is applied to $Gleader$ to group the leaders, where the number of groups is determined by the number of subswarms. Note that the clustering is done with respect to closeness in the decision space [128]. Next the resulting groups are assigned to their subswarm on condition that the number of internal iteration does not reach the user-defined maximum internal iteration value. Otherwise, the resulting groups are randomly assigned to any subswarm and the internal iteration is reset to zero. The

```
Begin
Parameters initialization for cell-based rank density estimation
scheme, population growing strategy, population declining strategy,
and adaptive local archives.

/*Swarm Population Initialization
Set no. of subswarms (n_swarm).
Initialize subswarms.
Set Maximum internal iterations (stmax).
Set Maximum iterations (tmax).
Set iteration t = 0. Set internal iteration st = 0.
Rank_and_density_estimation()

For each subswarm
    For each particle
        Fitness evaluation.
        Store local best (pbest_i).
    EndFor
    Store all found group leaders.
EndFor
Combine n_swarm of group leaders to Gleader.
t = 1
While t < tmax
     Apply clustering algorithm to Gleader.
     If st < stmax
        Assign group leaders to the subswarm.
        st = st + 1
     Else
        Randomly assign group leaders to a subswarm.
        st = 0
     EndIf
     Population_growing_strategy()
     For each subswarm
        Adaptive_local_archives()
        For each particle
            Select leader from group leaders.
            Flight.
            Fitness evaluation.
            Rank_and_density_estimation()
            Update local best (pbest_i).
         EndFor
         Store all group leaders.
     EndFor
     Combine n_swarm of group leaders to Gleader.
           Population_declining_strategy()
     t = t + 1
EndWhile
Report results in Gleader.
End
```

Figure 6.1 Pseudocode of DMOPSO.

internal iteration provides a chance for all of the subswarms to share their group leaders. Then, population growing strategy is applied to increase the number of particles while adaptive local archives scheme is applied to the group leaders of each subswarm to preserve the diversity. Next, the particles in each subswarm will select the leaders from their group leaders. As soon as the leader selection process is completed, the particles perform flight and update their local best ($\boldsymbol{pbest}_i$). Again, the following steps are repeated for all of the particles in their subswarm: 1) the particles' information is updated via the cell-based rank density estimation scheme; 2) the group leaders from subswarms are determined; and 3) all the group leaders are combined to *Gleader* . After these steps are finished, population declining strategy is performed to reduce the swarm population size, if justified. These steps are performed until they reach the maximum iteration.

6.3 Implementation Details

Unique designs of four strategies (highlighted in Figure 6.1) in supporting dynamic population are elaborated in this section.

6.3.1 Cell-based Rank Density Estimation Scheme

With a dynamic population size, adding or removing particles will affect Pareto rank of the existing particles and the population density of certain areas located on the objective space. This poses a problem of needing to recalculate the Pareto rank and density values of the particles to keep up the changes of the swarm population size. To counter the problem, we employed an existing scheme, cell-based rank and density estimation scheme, which has proved effective in DMOEA [139, 140].

Here, a briefly review of the cell-based rank and density estimation scheme is presented [139, 140]. The scheme consists of three main procedures: *Setting up the cells,* identify *particles' home addresses*, and *updating rank and density matrices*.

Setting Up the Cells: Divide the original *k*-dimensional objective space into $K_1 \times K_2 \times \ldots \times K_k$ cells (i.e., grids), thus the cell width in the *i*th objective dimension, d_i, can be calculated as

$$d_i = \frac{\max_{\mathbf{x} \in S} F_i(\mathbf{x}) - \min_{\mathbf{x} \in S} F_i(\mathbf{x})}{K_i}, i = 1,...,k, \tag{6.1}$$

where d_i is the width of the cell in the *i*th dimension, F_i refers to the *i*th objective, K_i denotes the number of cells designated for the *i*th dimension, $\mathbf{x}$ is taken from the whole decision space S, and we denote

$$F_i^{\min} = \min_{\mathbf{x} \in S} F_i(\mathbf{x}),\ F_i^{\max} = \max_{\mathbf{x} \in S} F_i(\mathbf{x}),\ i = 1,...,k, \tag{6.2}$$

The grid scales K_i, $i = 1,...,m$, are chosen heuristically and a prior knowledge would be desired in the choice of the grid scales. The F_i must be chosen large enough to accommodate the corresponding boundary range of the decision variables, $\mathbf{x}$, because if F_i is chosen too small, then $F_i^{\min}$ and $F_i^{\max}$ wouldn't be sufficiently small or large enough to include particles that are out of range in the objective space. Hence, the limitation of $F_i^{\min}$ and $F_i^{\max}$ must at least meet some minimum values that correspond to the boundary range of the decision variables, $\mathbf{x}$.

Identify Particles' Home Addresses: As shown in Figure 6.2, point **c** is denoted as the origin of the current objective space. In other words, **c** is the cross point of all the lower boundaries of an *k*-dimensional objective space. The position of **c** is denoted as

$\left[F_1^{\min}, F_2^{\min}, \ldots, F_k^{\min}\right]$. For a newly generated particle **Q**, whose position is $[q_1, q_2, \ldots, q_k]$ in the objective space, the distance between point **Q** and point **c** will be measured in each dimension in the objective space as $[t_1, t_2, \ldots, t_k]$, where

$$t_i = q_i - F_i^{\min},\ i = 1,...,k. \tag{6.3}$$

Therefore, the "home address" of particle **S** in the *i*th dimension is calculated as

$$h_i = \mathrm{mod}(t_i, d_i) + 1,\ i = 1,...,k, \tag{6.4}$$

where function $\mathrm{mod}(x, y)$ represents the modulus (integer part) after division x / y. Therefore, by this setting, finding the grid location (home address) of a single solution requires only k "division" operations. For example, in Figure 6.2, the "home address" for particle **Q** is **(4, 5)** and the other particles who share the same "home address" as **Q** are its "family members."

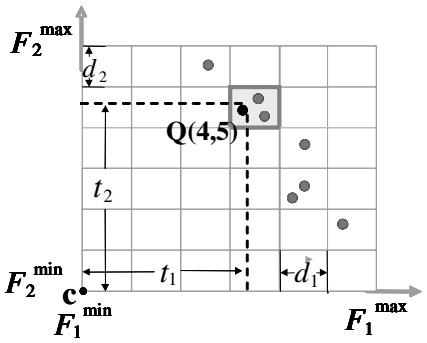


Figure 6.2 Illustration of cell-based rank and density estimation scheme.

Updating Rank and Density Matrices: The density value of a specific cell is referred to as the density value of the "home." The number of "family members" that share the same "home address" will be counted and saved as the density value of the "home." To update the rank values in the rank matrix, the ranking technique, known as Automatic Accumulated Ranking Strategy (AARS) is used [13].

Figures 6.3 to 6.5 present an example to demonstrate the cell-based rank and density scheme [13,139,140]. Assume two-dimensional objective space, $k = 2$, the objective space is determined via Equation (6.1) and divided into 6×6 cells using Equation (6.2). Figure 6.3(a) shows the initially setup of the objective space. Then, the center position of each cell is obtained, and two matrices are set up following the same cell configuration as Figure 6.3(a). These two matrices store the rank and density values of each cell, which initially has all 1's and 0's, respectively (shown in Figure 6.3(b)-(c)). Now, consider the initial population is generated and is mapped to the objective space (Figure 6.4(a)). The particles' home addresses are identified using Equations (6.3) and (6.4) and the rank and density matrices in Figure 6.4(b)-(c) show how the information of the particles are stored. When a new particle is generated and accepted, i.e., particle **C** in Figure 6.5(a), its "home address" can be located easily by following Equations (6.3) and (6.4). With its "home address", the rank values of the cells dominated by its "home" will be increased by one (Figure 6.5(b)), and the density value of its "home" will increase by one (Figure 6.5(c)). Meanwhile, if an existing individual is removed (example: particle **B** in Figure 6.4(a) is removed in Figure 6.5(a)), its "home" will be notified, and the rank values of the cells dominated by its "home" will be decreased by one, and the density value of its "home" will be decreased by one, correspondingly. Therefore, at each iteration, a particle can access its "home address" and then obtain the corresponding rank and density values. The "home address" is merely a "pointer" to locate a particle and to access its rank and density values. For instance, as shown in Figure 6.5, the "home address," rank and density values of particle **A** are (5,2), 2, and 1, respectively.

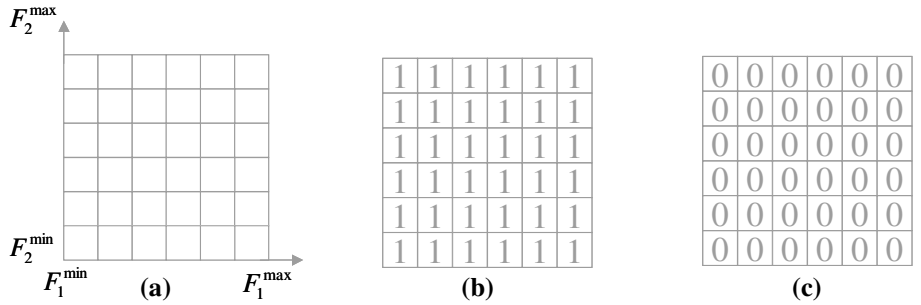


Figure 6.3 (a) Estimated objective space and divided cells, (b) initial rank value matrix of the given objective space, and (c) initial density value matrix of the given objective space [139,140].

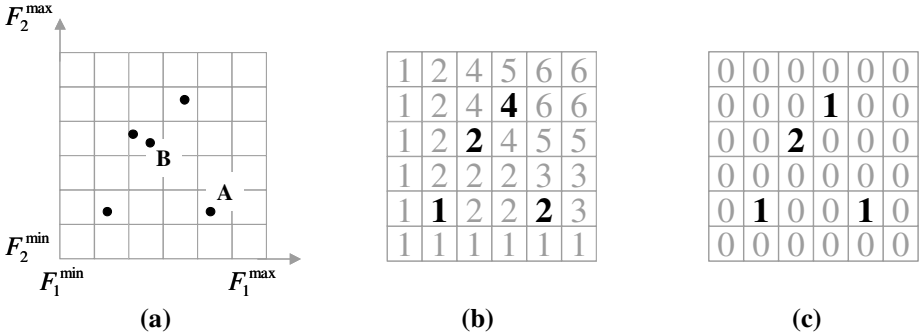


Figure 6.4 (a) Initial swarm population and the location of each particle, (b) rank value matrix of initial swarm population, and (c) density value matrix of initial swarm population [139,140].

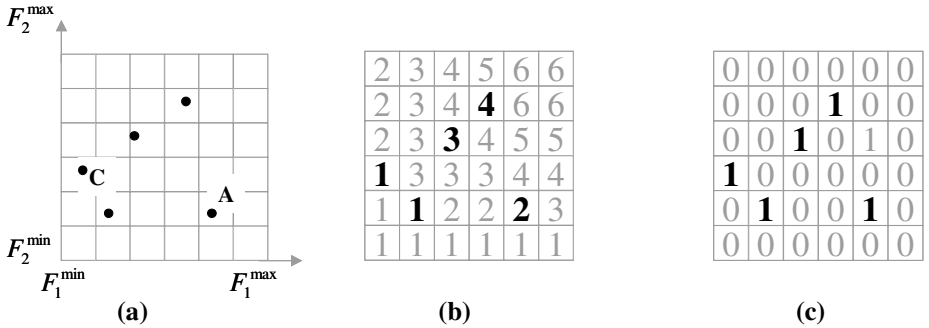


Figure 6.5 (a) New swarm population and the location of each particle, (b) rank value matrix of new swarm population, and (c) density value matrix of new swarm population [139,140].

With this scheme, fitter particles can be identified easily, since they just need to provide their "home addresses," and the current rank or density values of their home addresses. The cell-based rank and density estimation scheme is quite effective in managing a dynamic swarm population size in DMOPSO. The pseudocode of cell-based rank density estimation scheme is presented in Figure 6.6.

Function Rank_and_density_estimation (t , Pop , $(F_1^{\min},\ldots,F_k^{\min})$, $(F_1^{\max},\ldots,F_k^{\max})$, $(K_1,\ldots,K_k)$)

```
/* t = current iteration
/* Pop = current swarm population size
/* (F_1^min,...,F_k^min) = user-defined lower boundaries in m dimensional
                           objective space
/* (F_1^max,...,F_k^max) = user-defined upper boundaries in m dimensional
                           objective space
/* (K_1,...,K_k) = grid scales

Begin
   If t = 0,
      /*Calculate the cell rank and density values for initial
      swarm population.
      For i = 1 to k
         Calculate cell width from Equation (6.1)
      EndFor
      Rank_matrix = 1_{K_1×K_2×...×K_k}
      Density_matrix = 0_{K_1×K_2×...×K_k}
      /*Determine their home addresses.
      For each particle
         Compute Equation (6.3)
         Compute Equation (6.4)
      EndFor
      /*Update rank and density value.

   Rank_matrix([h_1 ... K_1],...,[h_k ... K_k]) = Rank_matrix([h_1 ... K_1],...,[h_k ... K_k]) + 1
   Density_matrix(h_1,...,h_k) = Density_matrix(h_1,...,h_k) + 1
End
```

Figure 6.6 Pseudocode of cell-based rank density estimation scheme [139, 140].

6.3.2 Perturbation Based Swarm Population Growing Strategy

Tan *et al.,* have proposed an incremental multiobjective evolutionary algorithm with dynamic population size that adaptively discovers the tradeoff surface and its desired population distribution density [135]. Among other proposed features, authors proposed a method of fuzzy boundary local perturbation to perturb the nondominated individuals to grow the population size. A similar concept is adopted in the proposed population growing strategy. A set of procedures is proposed to facilitate exploration and

exploitation capabilities for DMOPSO and, at the same time, to increase the swarm population size.

Procedure 1: The potential particles to be perturbed must have the highest possibility of producing new particles that will improve the convergence towards the Pareto front. In this case, the nondominated set is considered as candidate particles to produce new ones since they have a higher chance of producing fitter particles which will improve the convergence towards the Pareto front. However, selecting all particles in the nondominated set may result in excessive swarm population growth and, consequently, produce an uncontrollably large swarm population size. To solve this problem, the user can set a fixed parameter that will serve as the guideline for selecting the number of potential particles from the nondominated set. Instead of choosing a fixed parameter, a random number is used to stochastically determine the number of potential particles from the nondominated set to be chosen. The number of potential particles can be calculated using the following equation:

$$ns = \lfloor r_a \times (total\ no.\ of\ \ particles\ in\ nondominated\ set) \rfloor, \tag{6.5}$$

where r_a denotes a random number obtained from a uniform distribution within [0, 1], and *ns* denotes the number of particles to be selected to perturb. $\lfloor \cdot \rfloor$ represents the floor function. Once *ns* is determined, *ns* number of potential particles are randomly selected from the nondominated set. For example, refer to Figures 6.7(a)-(c). Based upon the information presented in the rank and density matrices, the total number of particles in the nondominated set of the current swarm population is equal to five. Assume *ns* is chosen to be 2, two particles (**D** and **E**) are randomly chosen as the candidate particles (Figure 6.7(d)). Note that these potential particles are referred to as selected particles.

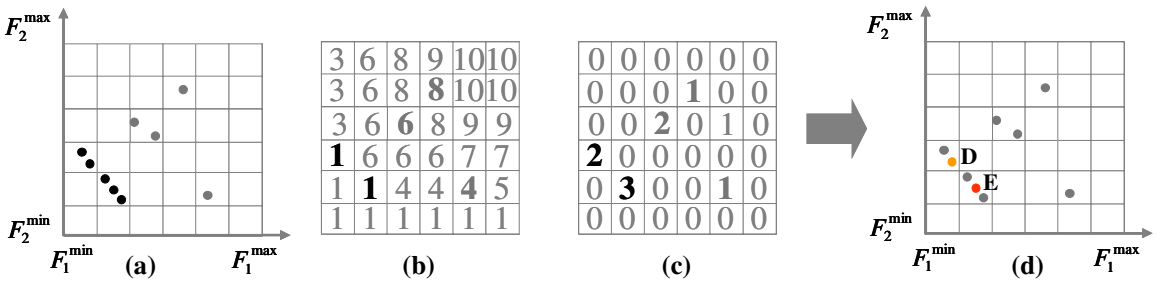


Figure 6.7 (a) Current swarm population and the location of each particle, (b) rank value matrix of current swarm population, (c) density value matrix of current swarm population, and (d) example of "potential" particles, particles D and E.

Procedure 2: The number of perturbations of the selected particle is adaptively determined in every iteration. Each selected particle's responsibility is to generate a certain number of new particles from the selected particle. A probability value is used to determine the number of perturbations adaptively in which the number of perturbations (number of new particles to be generated) is bound by the minimum and maximum number of perturbations, which is predefined by the user. Assuming at iteration t, the number of perturbations for each selected particle, $np(t)$, is determined by Equation (6.6):

$$np(t) = \begin{cases} (unp - lnp) \times \left(1 - 2 \times \left(\frac{t}{tmax}\right)^2 + \frac{lnp}{unp - lnp}\right), & 0 \le t \le (tmax/2) \\ (unp - lnp) \times \left(2 \times \left(\frac{t - tmax}{tmax}\right)^2 + \frac{lnp}{unp - lnp}\right), & (tmax/2) < t \le tmax \end{cases}, \tag{6.6}$$

where $tmax$ is denoted as the maximum iterations, lnp is the minimum number of perturbations, and unp is the maximum number of perturbations. unp is determined by maximum allowable perturbations for each particle, while lnp is determined based upon the minimum number of perturbations required for neighborhood exploration. Figure 6.8 is the illustration of Equation (6.6).

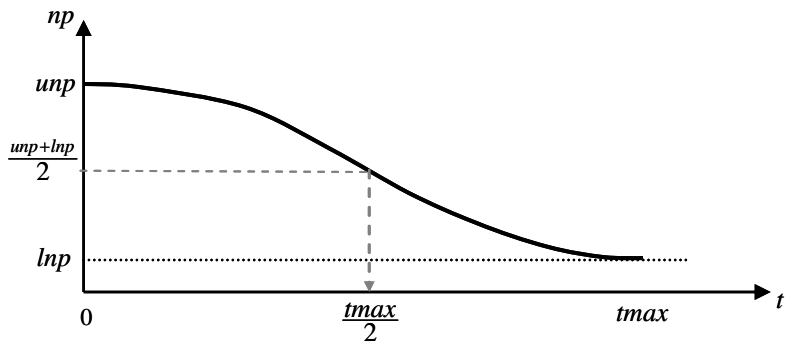


Figure 6.8 Number of perturbation per particle, *np* versus iteration, *t*.

Procedure 3: The concept of perturbations within and beyond the neighborhood to balance the exploitation and exploration capabilities of DMOPSO is adopted. To avoid generating too many new particles from being too far away from the selected particles, it is necessary to generate a larger number of new particles within the neighborhood than outside of the neighborhood. In order to achieve this goal, a set of equations is proposed as follows:

$$r_b = abs\left(Gaussian\ \left(0, 1/9\right)\right), \tag{6.7}$$

$$ld = rld \times x_j^L, \tag{6.8}$$

$$ud = rud \times x_j^U, \tag{6.9}$$

$$\Delta d\left(r_b\right) = \begin{cases} \left(ud - ld\right) \times \left(2 \times \left(r_b\right)^2 + \dfrac{ld}{ud - ld}\right), & 0 \le r_b \le 0.5 \\ \left(ud - ld\right) \times \left(1 - \left(2 \times \left(r_b - 1\right)^2\right) + \dfrac{ld}{ud - ld}\right), & 0.5 \le r_b \le 1 \end{cases}, \tag{6.10}$$

$$x_{i,j} = x_{i,j} + \Delta d\left(r_b\right). \tag{6.11}$$

Equation (6.7) is used to determine the additional distance from the selected particle corresponding to the decision space. The Δd is defined according to the function of r_b (see Figure 6.9). Several parameters are needed to compute Equation (6.10). These

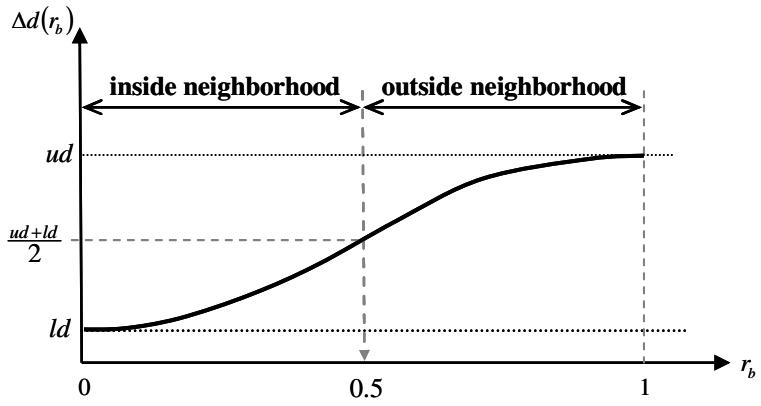


Figure 6.9 The additional distance $\Delta d(r_b)$ versus r_b.

parameters are *ld* , *ud* and r_b in which they can be computed via Equations (6.7) to (6.9), respectively. In Equation (6.8), *ld* is denoted as the minimum additional distance. It is computed by multiplying the parameters *rld* and x_j^L, where *rld* is the user-defined lower bound ratio and x_j^L is the lower bound of the decision variable x in dimension *j*. Parameter *ud* is denoted as the maximum additional distance. Equation (6.9) shows how the parameter *ud* is calculated where *rud* is the user-defined upper bound ratio and x_j^U is the upper bound of the decision variable x in dimension *j*. In this paper, the parameters *rld* and *rud* are selected within the range of $[0,0.02]$ and $(0.02,0.7]$, respectively.

Presented in Equation (6.7), the parameter r_b is the absolute value of a random number in which the random number is drawn from the Gaussian distribution with zero mean and a variance of $1/9$. With the mean 0 and variance (σ^2), $1/9$, more random numbers will be generated near the lower end of the range, i.e. $[0, 3\sigma/2]$, while less random numbers will be generated near the upper range, i.e., $(3\sigma/2, 3\sigma]$. Once the Δd is computed, it is added to the decision variable of the selected particle *i* at dimension *j*,

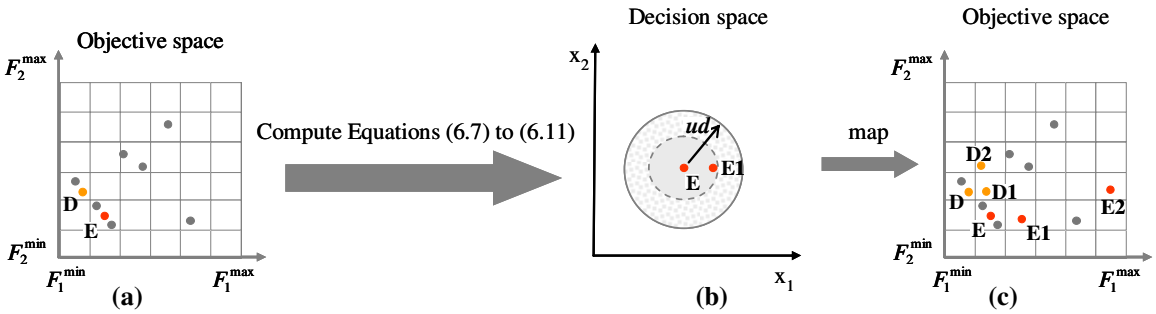


Figure 6.10 (a) Selected particles (D and E) from Figure 6(d), (b) representation of Equation (9) in decision space, and (c) current swarm population and new added ones in objective space.

i.e., $x_{i,j}$ (Equation (6.11)). Notice that since the resulted r_b value is more likely to be lower than or equal to 0.5 (according to Equation (6.10)), it is more likely that Δd will be small (Figure 9). Consequently, the new $x_{i,j}$ value will likely lie within the neighborhood rather than outside of the neighborhood. Figure 6.10(a)-(c) is the illustration of *Procedure 3*. In Figure 6.10(b), the inside neighborhood of particle **E** (from Figure 6.10(a)) in the decision space is bounded by the circumference of the inner circle. The outside neighborhood is the area between the inner circle and the outer circle, where the radius of the outer circle is *ud* (as presented in Figure 6.9). Figure 6.10(b) also shows a new particle is generated by computing Equations (6.7)-(6.11), and it is denoted as particle **E1**. Particle **E1** is mapped to the objective space illustrated in Figure 6.10(c). Observed in Figure 6.10(c), using *Procedure 3*, particles **E1** and **E2** are generated by particle **E** while particles **D1** and **D2** are generated by particle **D**. Figure 6.11 presents the pseudocode of population growing strategy.

6.3.3 Swarm Population Declining Strategy

To prevent the extensive growth in swarm population size, a population declining strategy is proposed to control the swarm population size. In DMOPSO, the necessary

```
Function Population_growing_strategy (rank_m,density_m, lnp , unp , tmax ,t,
rld , rud )

/*rank_m  = rank matrix; density_m = density matrix
/* lnp = the minimum number of perturbations;
/* unp = the maximum number of perturbations;
/* tmax = denoted as the maximum iteration; t = current iteration
/* rld = the user-defined lower bound ratio;
/* rud = the user-defined upper bound ratio;

Begin
    r_b = rand[0,1]
   Obtain nondominated set from rank_m and density_m.
   Calculate ns from Equation (6.5),
   Randomly select potential particles (or selected particles).
   Compute Equation (6.6), np(t)
   For 1 to ns ,
       For 1 to np(t),
         Generate r_b using Equation (6.7)
         Compute Equations (6.8)-(6.9) for ld and ud .
         Compute Δd using Equation (6.10).
         Add Δd to x_{i,j} (where i represents the selected particle i).
         Update rank_m and density_m.
       EndFor
   EndFor
End
```

Figure 6.11 Pseudocode of swarm population growing strategy.

condition to remove a particle depends upon the rank and crowdedness indicators. In this case, the values in the rank and density matrices are used to determine whether the particles in a cell are to be removed. In addition, a selection ratio is implemented to regulate the number of particles to be removed and to provide some sort of diversity preservation at the same time.

Rank Indicator: Imposed in this indicator is the idea that particles far from the nondominated front will have less of a chance to survive to the next iteration since they have a higher chance of "losing" their leaders. This means the particles far from the nondominated front are likely to be eliminated. The rank value of a cell obtained from the

rank matrix is converted into a rank indicator in order to measure the dominance status of a cell compared to the others. Figure 6.12(b) presents the rank matrix of the current swarm population depicted in Figure 6.12(a). Assume at iteration t, the cell c in which particle i is located has the rank value of $rank(c_i,t)$, the rank indicator of particle i located at cell c at iteration t, $R(i,t)$, is given as

$$R(i,t) = \frac{1}{rank(c_i,t)}. \tag{6.12}$$

Equation (6.12) indicates that a particle that resides in the cell with rank value "1" (e.g., particle **F** in Figure 6.12(a)) will have its R value equal to 1 as shown in Figure 6.12(c). The particle in a cell with a higher rank value will result in a lower R value. Refer to Figure 6.12(b); the rank value of particle **G** is higher than the rank value of particle **F** and Figure 6.12(c) shows that the resulted R value is much lower, which is 0.11. Hence, as the R value of a particle decreases, this implies that the particle has an increasing chance of being eliminated, since it is farther from the nondominated front.

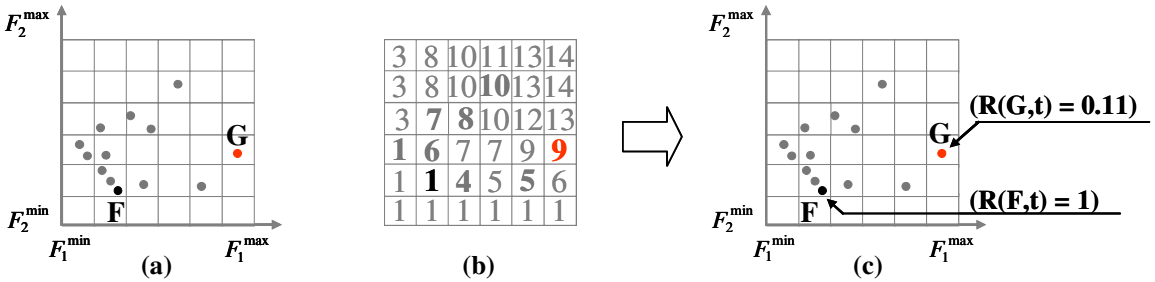


Figure 6.12 (a) Current swarm population and the location of each particle, (b) rank matrix of current swarm population, and (c) *R* values for particles F and G.

Crowdedness Indicator: This indicator involves the control of local population size, i.e., population size per cell. The population size per cell is regarded as the density value of a cell, which is defined as the number of particles located in a cell. Using the current density information of a concerned cell in which the information can be found in

the density matrix, the crowdedness indicator of a particle in a concern cell can be computed. Figure 6.13(b) shows the density matrix of the current population, which is illustrated in Figure 6.13(a). At iteration *t*, the cell *c* in which particle *i* is located has the density value $density(c_i, t)$. The crowdedness indicator of particle *i* located at cell *c* at iteration *t*, $D(i,t)$, is defined as

$$D(i,t) = \begin{cases} 1 - \dfrac{ppv}{density(c_i,t)}, & \text{if } density(c_i,t) > ppv \\ 0, & otherwise \end{cases}. \qquad (6.13)$$

Note that *ppv* value represents the desired particle size per cell and it is a user-defined parameter. Equation (6.13) shows that a cell with high density value will have a higher *D* value closer to 1. In Figure 6.13(b), the density value of particle **F** is equal to 3. With the *ppv* value set to 2, the *D* value of particle **F** is equal to 0.33 (refer to Figure 6.13(c)). On the other hand, if a cell has density value lower than or equal to *ppv*, then the *D* value is equal to 0 (particle **G** in Figure 6.13(c)). This indicates that if the particles reside in a concern cell that has more than the desired particle size, then these particles are likely to be eliminated to reduce the level of congestion in the concern cell. Note that $R(i,t)$ and $D(i,t)$ are between zero and one.

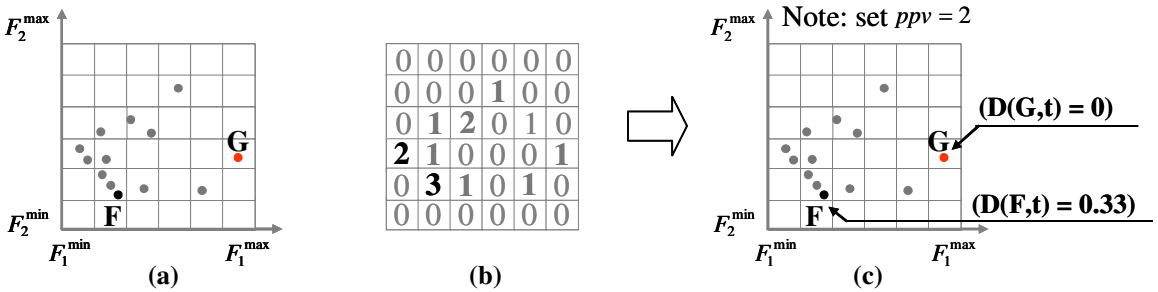


Figure 6.13 (a) Current swarm population and the location of each particle, (b) density matrix of current swarm population, and (c) *D* values for particles F and G.

Likelihood of Removing the Particle: At iteration *t*, the likelihood that the particle *i* with rank value $R(i,t)$ and density value $D(i,t)$ is to be eliminated is computed using the following equation:

$$L(i,t) = (1 - R(i,t)) \times D(i,t). \tag{6.14}$$

Equation (6.14) implies that for those particles that have low *R* values (i.e., away from the nondominated front) or have high *D* values (i.e., located in the crowded cells), these particles will have high likelihood value, *L*. Refer to Figures 6.12 and 6.13; the *L* values for particle **F** and **G** are both 0, implying they are either located in a nondominated front or a non-crowded cell. To determine whether a selected particle *i* will be removed, a random number with uniform distribution between $[0,1]$ is generated to compare with the likelihood $L(i,t)$. If the likelihood is larger than the random number, then particle *i* is selected as a potential candidate to be eliminated from the swarm population. Note that at iteration *t*, all selected particles to be eliminated are stored in a temporary memory, M_t. Then, the selection ratio is applied to determine the exact number of particles in M_t to be eliminated from the swarm population.

Selection Ratio: If the removal of particles is only based upon the *L* value, then there is a possibility of eliminating an excessively large quantity of particles in which some may carry unique schema to contribute in the search process. A selection ratio inspired by [142] is used to stochastically allocate a small percentage of particles in the swarm population for removal. Hence, given a selection ratio, $S_r \in [0,1]$, at iteration *t*, the equation to compute the number of particles with high likelihood *L* to be eliminated is given as

$$Pop_{remove}(t) = S_r \times |M_t|, \tag{6.15}$$

where $Pop_{remove}(t)$ is the allocated number of particles in the population for elimination and $|M_t|$ denotes the population size in M_t at iteration t. Note that the choice of the selection ratio is dependent upon the user's preference, where it can be a function of the swarm population size at each iteration or it can be a fixed ratio. For this study, the selection ratio is a fixed number, which is set to be a small number, i.e., $S_r \leq 0.2$. With a small selection ratio, there is a possibility that those selected particles in M_t may not be eliminated. In other words, some of the selected particles in M_t whose rank values are low or who are located in crowded cells may survive to the next iteration. In addition, a small selection ratio can prevent the removal of an uncontrollable, large number of particles while providing some degree of diversity preservation. Figure 6.14 presents the

```
Function Population_declining_strategy (rank_m, density_m, S , Pop , ppv )

   /* rank_m  = rank matrix; density_m = density matrix
   /* S  = selection ratio
   /* Pop = current swarm population size in M_t.
   /* ppv = desired population size per cell

   Begin
       r_c = rand[0,1]
       For each particle
           Compute Rank Indicator, R, using rank_m.
           Compute Crowdedness Indicator, D, using density_m and ppv .
           Compute Likelihood of removing the particle, L.
           If  L > r_5
             Store particle to M_t.
           EndIf
       EndFor
       Pop_remove(t) = S × |M_t|
       Randomly choose Pop_remove number of particles from M_t.
       Remove the chosen particles.
   End
```

Figure 6.14 Pseudocode of swarm population declining strategy.

pseudocode of population declining strategy. In the following, some observations are drawn.

1. It is obvious that the setting of *ppv* value depends on the grid scales, K_i, $i = 1,\ldots,k$. For instance, if the grid scale is very small (e.g., $K_i = 2$, $i = 1,\ldots,k$), then the *ppv* value should be large enough to balance the small grid scale. Otherwise the frequency of locating "crowded" cell (with high density value) will be high and may increase likelihood of removing those particles in the "crowded" cell. In fact, the minimum appropriate *ppv* value has an inverse relationship with the grid scale, considering that all K_i are set to the same number. In addition, if each K_i is set to a different value, then it unnecessarily complicates the setting of *ppv* value. To avoid such situation, the selection ratio, S_r, is implemented and the choice of *ppv* value will not solely affect the elimination of the particles.
2. At each iteration, the most undesirable particles will be chosen to be eliminated according to likelihood value, *L*, which is based on their rank value and density condition. These undesirable particles have either low *R* value or high *D* value. This implies that these particles are either not contributing to the search process or they are too many particles located in the confined area. By employing the likelihood of removing the particle scheme, these redundant particles will likely be eliminated. On the contrary, those particles with either high *R* value or low *D* value will have a chance to survive to the next iteration. In fact, particles with high *R* value are preserved because they will most likely contribute to the search process by bringing other particles to help in finding better solutions. Particles with low *D* value are also

preserved so that they are given a chance to explore the "isolated" area or may even discover the potential "undiscovered" area in the search space.

3. The selection ratio, S_r, has to be a small number because if the *S* is too big (e.g., 0.9), during the initial stage where the swarm population size is significantly small, high *S* will result in deleting most of the undesirable particles in M_t. Eliminating too many undesirable particles in M_t in the early stage of the evolutionary process may cause inefficiency in the optimization search because some of the undesirable particles may offer unique schema in the following iterations. In the initial stage, more particles imply a better chance in finding good solutions. As the swarm population size grows larger, high S_r will result in excessively deleting undesirable particles in M_t. As a result, the algorithm may incur more computational load to locate the optimal Pareto front since, at each iteration, the resulted swarm population size is considerably low. Hence, the selection ratio is suggested to be at most 0.2, which is less than 20 percent of the swarm population size in M_t. The criterion to choose a value for the selection ratio depends upon $(ud - ld)$ in Equation (6.10). If the gap of $(ud - ld)$ is large, then the population growing rate will increase in a fast pace and to control the fast rate of growth, the selection ratio, S_r, should be chosen slightly higher but no more than 0.2.

6.3.4 Adaptive Local Archive and Group Leader Selection Procedures

In cMOPSO [128], based upon a probability value, the particles in a subswarm randomly select the assigned group leaders since all resulting group leaders are grouped via a clustering algorithm. Random selection can provide equal probability of group leaders being chosen as the leader for a particle, and has a higher probability of achieving

tightly grouped solutions that are close to the true Pareto front [117]. Yet the resulting Pareto front may not well extend into the complete Pareto front. For this reason, the idea of local search procedure is adopted and the aim is similar to [143], which is used to improve the solutions in each swarm. Hence, the idea of local search procedure known as the adaptive local archive is proposed. Similar to the adaptive grid procedure proposed in [30] and [120], the aim of adaptive local archive is to improve the diversity in sections of Pareto front that associate with each subswarm. The following presents the adaptive local archive and group leader selection procedures.

Adaptive Local Archive Procedure: Once clustering algorithm is applied to the *Gleader* to group the leaders in the decision space, each group leader is now referred to as the local archive. Each local archive contains the group leaders that correspond to their subswarm (e.g., G1 and G2 shown in Figure 6.15(a)-(b)). For the purpose of visualization, m is chosen to be 2 in Figure 6.15. In each local archive, with the group leaders' objective values, the objective space is divided into a set of cells using the adaptive grid procedure. Then, each particle chooses its group leader by following the *Group Leader Selection Procedure*. In each local archive, the number of particles that the cell contains is recorded. At each iteration, if any new group leaders lie outside the current bound of the grid, then the objective space is re-divided based upon the new fitness values. Each particle is relocated to its nearest cell, and the number of particles that the cell contains is also updated. For simplicity, in this paper, the number of cells is predetermined from a user-defined number of grid subdivisions or K_a for all dimensions. This means that the m-dimensional objective space is divided into $K_1 \times K_2 \times \ldots \times K_m = K_a \times K_a \times \ldots \times K_a$ cells. Figure 6.15(c) shows the number of grid

subdivisions, K_a , is equal to 4. Figure 6.16 presents the pseudocode of adaptive local archive procedure.

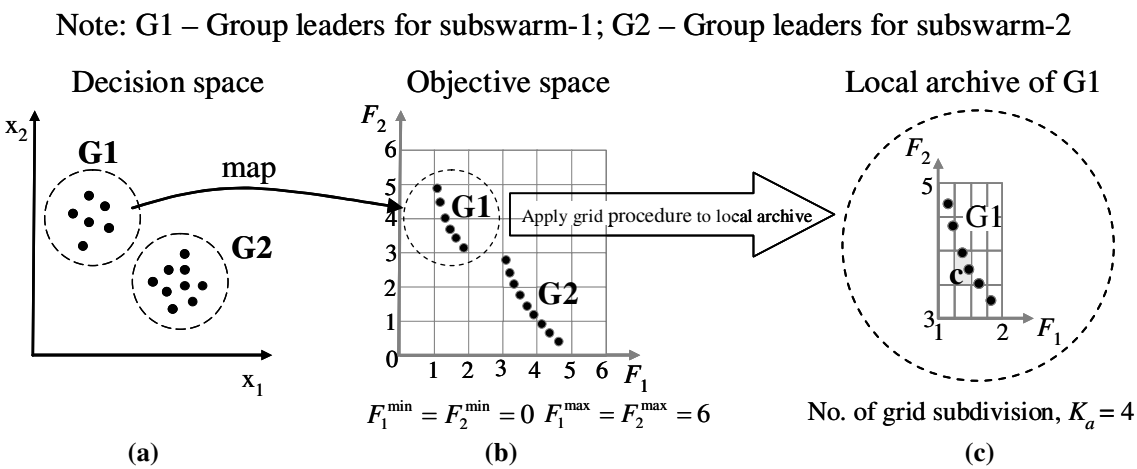


Figure 6.15 (a) Two group leaders are grouped via clustering algorithm, (b) two group leaders in decision space are mapped to objective space, and (c) adaptive grid procedure is applied to local archive of G1.

Function Adaptive_local_archives (K_a, n_{swarm}, g_leader)

```
/* Ka = Number of grid subdivisions;
/* nswarm = Number of subswarms;
/* g_leader = group leaders

Begin
   For  j = 1:nswarm
      Generate hypercubes based upon the Ka value and the fitness
     value of
      its group leaders.
      For each member in g_leader(j)
          Search for its nearest cell based upon the fitness value.
          Update number of particles (leaders) in the cell.
      EndFor
   EndFor
End
```

Figure 6.16 Pseudocode of adaptive local archives algorithm.

Group Leader Selection Procedure: After the adaptive local archive procedure is completed, the information on the number of "occupants" in the cells of the local archive is utilized. These cells that contain more than one particle are first assigned a fixed value. With the idea of fitness sharing, the fixed values of the cells are divided by the number of

particles they contain. For simplicity, each resulted value of a cell will be defined as $F_A(a)$, where a represents a cell in the local archive. Next, by using all available F_A values, roulette wheel selection is applied to select the cell. In the selected cell, particle i will randomly select one of the "occupants" within the cell. The idea of applying the fitness sharing is to measure the level of congestion in each cell. Those cells that are highly congested will have low F_A values or vice versa. With roulette wheel selection, this selection scheme favors the least congested cell. As a result, the particle will choose one of the group leaders that reside in the least congested cell. Therefore, the leaders are selected in such a way that diversity is preserved.

6.4. Comparative Study

In this section, two studies are conducted. In the first study, presented in Subsection 6.4.4, the performance of the DMOPSO is compared with five state-of-the-art MOPSO algorithms: OMOPSO [121], MOPSO [120], Cluster-MOPSO (cMOPSO) [128], Sigma-MOPSO (sMOPSO) [113], and NSPSO [109]. The comparison is done on the standard test suit. The second study, presented in Subsection 6.4.5, investigates the computational cost of the proposed algorithm and the selected MOPSOs.

6.4.1 Test Function Suite

To compare the performance of DMOPSO with the five selected MOPSOs, the standard ZDT test suite and an additional test function selected from the DTLZ test suite are used [39,70]. The test functions are presented in Table 6.1. As noted in the comments column, the test functions possess different characteristics to test the performance of the

algorithms. The first five test functions are two-objective minimization problems and the number of decision variables used here is 100, i.e., $n = 100$. The sixth test function or DTLZ2 is three objective functions with 12 decision variables.

Table 6.1 The six test problems used in this study. All objective functions are to be minimized.

Problems	Objective Functions	Variable Bounds	Comments
ZDT1	$F_1(\mathbf{x}) = x_1$ $F_2(\mathbf{x}) = c(\mathbf{x})\left[1 - \sqrt{F_1(\mathbf{x})/c(\mathbf{x})}\right]$ $c(\mathbf{x}) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$	$x_i \in [0,1]$ $i = 1,\ldots,n$	It has the convex Pareto fronts. It challenges the algorithm's ability to find and produce a quality spread of Pareto front.
ZDT2	$F_1(\mathbf{x}) = x_1$ $F_2(\mathbf{x}) = c(\mathbf{x})\left[1 - (F_1(\mathbf{x})/c(\mathbf{x}))^2\right]$ $c(\mathbf{x}) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$	$x_i \in [0,1]$ $i = 1,\ldots,n$	It has the non-convex Pareto fronts. It challenges the algorithm's ability to find and produce a quality spread of Pareto front.
ZDT3	$F_1(\mathbf{x}) = x_1$ $F_2(\mathbf{x}) = c(\mathbf{x})\left[1 - \sqrt{F_1(\mathbf{x})/c(\mathbf{x})} - \frac{F_1(\mathbf{x})}{c(\mathbf{x})}\sin(10\pi F_1(\mathbf{x}))\right]$ $c(\mathbf{x}) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$	$x_i \in [0,1]$ $i = 1,\ldots,n$	It possesses a convex and disconnected Pareto front. It exploits the algorithms' ability to search for all of the disconnected regions and to maintain a uniform spread on those regions.
ZDT4	$F_1(\mathbf{x}) = x_1$ $F_2(\mathbf{x}) = c(\mathbf{x})\left[1 - \sqrt{F_1(\mathbf{x})/c(\mathbf{x})}\right]$ $c(\mathbf{x}) = 1 + 10(n-1) + \sum_{i=2}^{n}\left(x_i^2 - 10\cos(4\pi x_i)\right)$	$x_1 \in [0,1]$ $x_i \in [-5,5]$ $i = 2,\ldots,n$	The difficulty is finding the global Pareto front in all of the 21^9 local segments. It is also a multifrontal problem where it presents a complexity with multi-modality characteristic.
ZDT6	$F_1(\mathbf{x}) = 1 - \exp(4x_1)\sin^6(6\pi x_1)$ $F_2(\mathbf{x}) = c(\mathbf{x})\left[1 - (F_1(\mathbf{x})/c(\mathbf{x}))^2\right]$ $c(\mathbf{x}) = 1 + 9\left[\left(\sum_{i=2}^{n} x_i\right)\Big/(n-1)\right]^{0.25}$	$x_i \in [0,1]$ $i = 1,\ldots,n$	It has a nonconvex Pareto front. Its difficulties rest on the low density of solutions across the non-convex Pareto front and the non-uniform spread of solutions along the Pareto front.
DTLZ2	$F_1(\mathbf{x}) = (1 + c(\mathbf{x}))\cos\left(\frac{\pi x_1}{2}\right)\cos\left(\frac{\pi x_2}{2}\right)$ $F_2(\mathbf{x}) = (1 + c(\mathbf{x}))\cos\left(\frac{\pi x_1}{2}\right)\sin\left(\frac{\pi x_2}{2}\right)$ $F_3(\mathbf{x}) = (1 + c(\mathbf{x}))\sin\left(\frac{\pi x_1}{2}\right)$ $c(\mathbf{x}) = \sum_{i=1}^{n}(x_i - 0.5)^2$	$x_i \in [0,1]$ $i = 1,\ldots,n$	Its true Pareto front is on the first quadrant of a unit sphere. Since the true Pareto front is a surface, this test function poses a challenge for MOPSOs to search for the true Pareto front.

6.4.2 Parameter Settings

Each algorithm is set to perform 20,000 fitness function evaluations. The parameter configurations for all selected MOPSO algorithms are summarized in Table 6.2, while Table 6.3 presents the DMOPSO's parameter configurations for each test function. Note that all of the algorithms produced final Pareto fronts of fixed size swarm population except for cMOPSO and DMOPSO, which do not have fixed archive sizes. All of the algorithms are implemented in Matlab. In this study, all of the algorithms use a real-number representation for decision variables. However, binary representation of decision variables can be easily adopted, if necessary. For each experiment, 50 independent runs were conducted to collect the statistical results.

Table 6.2 Parameter configurations for five selected MOPSOs.

	Internal population size	Archive size	No. of iterations	Other parameters or remarks
OMOPSO	100	100	200	Mutation probability = $1/codesize$ and the values of w, c_1 and c_2 are random values (as proposed in [121]) $\varepsilon = 0.0075$ (Note: For ZDT6, $\varepsilon = 0.001$)
MOPSO	100	100	200	50 divisions adaptive grid; mutation probability = 0.5, (as proposed in [120])
cMOPSO	40	Not fixed	100	No. subswarms, $n_{swarm} = 4$; internal iterations, $stmax = 5$ (as proposed in [128])
sMOPSO	200	200	100	Fixed inertial weight value, w = 0.4; Turbulence Factor, R is $[-1,1]$
NSPSO	200	-	100	Fixed inertial weight value, w = 0.4

6.4.3 Selected Performance Metrics

Both quantitative and qualitative comparisons are made to validate the proposed DMOPSO against the five selected MOPSOs. For qualitative comparison, the plots of final Pareto fronts are presented for visualization. As for quantitative comparison, two performance metrics are taken into consideration to measure the performance of

algorithms with respect to dominance relations. The results are illustrated by statistical box plots.

Table 6.3 Parameter configurations for DMOPSO with number of iterations is based upon 20,000 evaluations.

Test Suites	Internal population size	Archive size	Parameter settings Special remarks
ZDT1	5 per swarm	Not fixed	No. subswarms, $n_{swarm} = 4$; grid scale, $K_i = 100, i = 1,2$; $stmax = 5$; $unp = 3$, $lnp = 1$; $rud = 0.7$; $rld = 0.02$; $ppv = 10$; $S = 0.02$; and $K_a = 10$.
ZDT2	20 per swarm	Not fixed	No. subswarms, $n_{swarm} = 2$; grid scale, $K_i = 100, i = 1,2$; $stmax = 5$; $unp = 3$, $lnp = 1$; $rud = 0.7$; $rld = 0.02$; $ppv = 10$; $S = 0.02$; and $K_a = 40$.
ZDT3	6 per swarm	Not fixed	No. subswarms, $n_{swarm} = 3$; grid scale, $K_i = 100, i = 1,2$; $stmax = 5$; $unp = 3$, $lnp = 1$; $rud = 0.7$; $rld = 0.02$; $ppv = 10$; $S = 0.02$; and $K_a = 15$.
ZDT4	20 per swarm	Not fixed	No. subswarms, $n_{swarm} = 2$; grid scale, $K_i = 100, i = 1,2$; $stmax = 5$; $unp = 5$, $lnp = 1$; $rud = 0.9$; $rld = 0.02$; $ppv = 10$; $S = 0.02$; and $K_a = 40$.
ZDT6	5 per swarm	Not fixed	No. subswarms, $n_{swarm} = 4$; grid scale, $K_i = 100, i = 1,2$; $stmax = 5$; $unp = 3$, $lnp = 1$;; $rud = 0.9$; $rld = 0.02$; $ppv = 10$; $S = 0.02$; and $K_a = 40$.
DTLZ2	5 per swarm	Not fixed	No. subswarms, $n_{swarm} = 4$; grid scale, $K_i = 80, i = 1,2,3$; $stmax = 5$; $unp = 3$, $lnp = 1$; $rud = 0.7$; $rld = 0.02$; $ppv = 10$; $S = 0.02$; and $K_a = 10$.

Hypervolume Indicator (S Metric) [47]: Assuming a minimization problem, this unary indicator calculates the size of the region covered by a reference point. Larger value indicates that the nondominated set produced is better. The advantage of this indicator is able to measure both diversity and how well the algorithm converges to the true Pareto front. Given two nondominated sets, A and B, with the same reference point, then the hypervolume indicator of A is denoted as $I_H(A)$and the hypervolume indicator of B is denoted as $I_H(B)$. If $I_H(A) > I_H(B)$, then B is not better than A for all pairs. This means a certain portion of objective space is dominated by A and not by B. However, a reference point is required to compute this indicator. In this chapter, the method to

determine the reference point is as follow: First, the collection of both nondominated sets is combined into a single set. Second, from this set, the worst objective function values of *m*-dimension are found, and they are shifted by a fixed parameter. Then, the shifted version of these worst values is used as the reference point. Mann-Whitney rank-sum test is implemented to test for significant difference between two independent samples [144].

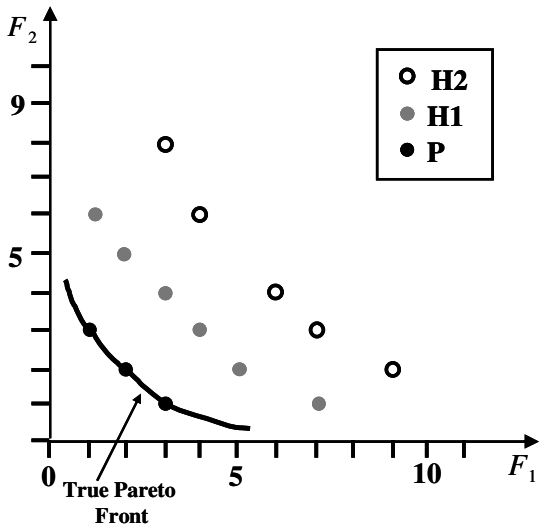


Figure 6.17 Sets H1, H2, and P are shown. By using the additive binary epsilon indicator, H1 strictly dominates H2 and H1 is strictly dominated by the true Pareto front.

Table 6.4 The computed additive binary epsilon indicator, $I_{\varepsilon+}(A,B)$, for all combination of H1, H2, and P as shown in Figure 6.17.

B	A		
	H1	H2	P
H1	1	2	0
H2	-1	1	-1
P	2	4	1

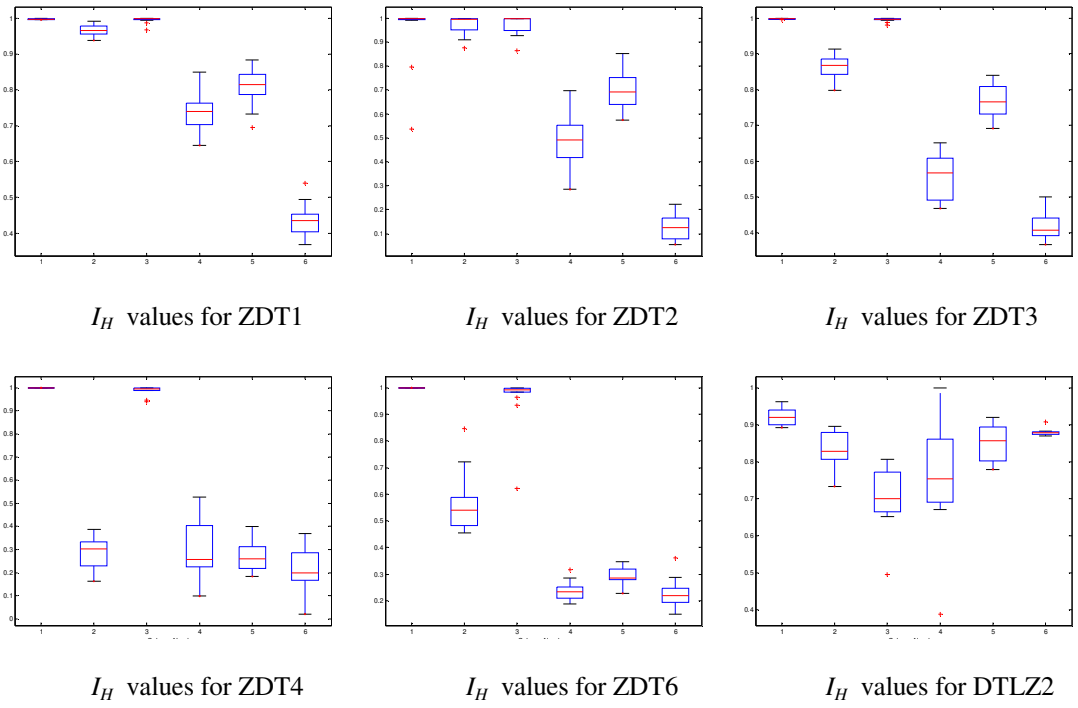
Additive Binary Epsilon Indicator [46]: This binary indicator aims to detect whether a nondominated set is better than another. Given two nondominated sets, *A* and *B*, the additive binary epsilon indicator for the pair are denoted as $I_{\varepsilon+}(A,B)$ and $I_{\varepsilon+}(B,A)$. If $I_{\varepsilon+}(A,B)<0$ and $I_{\varepsilon+}(B,A)>0$, then *A* is strictly better than *B*. If $I_{\varepsilon+}(B,A)\le 0$ and $I_{\varepsilon+}(A,B)<I_{\varepsilon+}(B,A)$, then this implies that *A* weakly dominates *B*. Lastly, if

$I_{\varepsilon+}(A,B) > 0$ and $I_{\varepsilon+}(B,A) > 0$, then $A \parallel B$, which indicates that A and B are incomparable. Again, Mann-Whitney rank-sum test is implemented to check if there is significant difference between the two distributions for $I_{\varepsilon+}(A,B)$ and $I_{\varepsilon+}(B,A)$. For example, Table 6.4 shows the computed indicator, $I_{\varepsilon+}$, for the sets H1, H2, and P, which are showed in Figure 6.17. From Table 6.4, it can shows that H1 strictly dominates H2 since $I_{\varepsilon+}(H1,H2) = -1$ and $I_{\varepsilon+}(H2,H1) = 2$, and P strictly dominates H1 since $I_{\varepsilon+}(P,H1) = 0$ and $I_{\varepsilon+}(H1,P) = 2$. Similar conclusion, P strictly dominates H2. Again, Mann-Whitney rank-sum test is implemented to check if there is significant difference between the two distributions for $I_{\varepsilon+}(A,B)$ and $I_{\varepsilon+}(B,A)$ [144].

6.4.4 Performance Evaluation of DMOPSO against the selected MOPSOs

The performance metric for hypervolume indicator (I_H value) is computed for each MOPSO over 50 independent runs. Figure 6.18 presents the box plots of I_H values found in all MOPSOs considered. The figure shows that DMOPSO and MOPSO share the highest I_H values for most test functions except for test function DTLZ2. DMOPSO achieves the highest I_H value for DTLZ2. Higher I_H value indicates the ability of the algorithm to dominate a larger region in the objective space. It is hard to determine whether DMOPSO is significantly better than MOPSO for test functions ZDT1, ZDT2, ZDT3, ZDT4, and ZDT6 since they attain the relative close I_H values from Figure 6.18. Hence, the Mann-Whitney rank-sum test is used to examine the distribution of the I_H values. The tested results are presented in Table 6.5. Observe the results in Table 6.5: DMOPSO and MOPSO share the same victory for test functions ZDT1, ZDT2, ZDT3, and ZDT4. Only for function ZDT2 does OMOPSO share the winner's slot with

DMOPSO and MOPSO. For the rest of the MOPSOs (i.e., cMOPSO, sMOPSO, and NSPSO), DMOPSO clearly performed better. In addition, Figure 6.18 shows that the standard deviations for DMOPSO are consistently lower, which indicates DMOPSO is more reliable in producing better solutions than those selected MOPSOs.



I_H values for ZDT1 I_H values for ZDT2 I_H values for ZDT3

I_H values for ZDT4 I_H values for ZDT6 I_H values for DTLZ2

Figure 6.18 Box plot of hypervolume indicator (I_H values) for all test functions (Start from top left) by algorithms 1-6 represented (in order): DMOPSO, OMOPSO, MOPSO, cMOPSO, sMOPSO, and NSPSO.

Figures 6.19-6.24 illustrate the results (in box plots) for additive binary ε-indicator where each figure gives the results for a test function. Each figure presents two box plots of $I_{\varepsilon+}(\text{DMOPSO}, X_{1-5})$ and $I_{\varepsilon+}(X_{1-5}, \text{DMOPSO})$, in which algorithms 1-5 represent OMOPSO, MOPSO, cMOPSO, sMOPSO, and NSPSO, respectively. It seems that DMOPSO performs relatively better with respect to dominance relation than most of the MOPSOs (i.e., OMOPSO, cMOPSO, sMOPSO, and NSPSO) for functions ZDT1 to ZDT6. For example, Figure 6.19 shows that DMOPSO strictly dominates NSPSO on

Table 6.5 The distribution of I_H values tested using Mann-Whitney rank-sum Test [144]. The table presents the z values and p-values with respect to the alternative hypothesis (i.e., p-value < α=0.05) for each pair of DMOPSO and a selected MOPSO. In each cell, both values are presented in a bracket: (z value, p-value). The distribution of DMOPSO is significantly difference or better than those selected MOPSO unless stated.

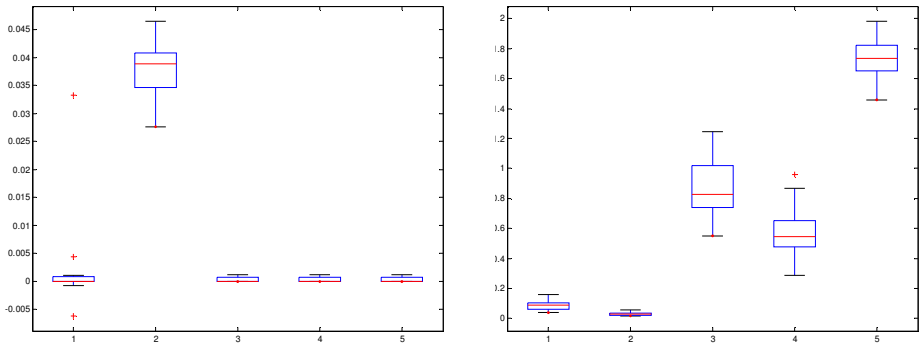
Test Functions	I_H (DMOPSO) AND				
	I_H (OMOPSO)	I_H (MOPSO)	I_H (cMOPSO)	I_H (sMOPSO)	I_H (NSPSO)
ZDT1	(-4.6455, 3.4E-06)	(-0.4148,>0.05) *no difference*	(-4.6455, 3.4E-06)	(-4.6455, 3.4E-06)	(-4.6455, 3.4E-06)
ZDT2	(-0.9125,>0.05) *no difference*	(-0.4977,>0.05) *no difference*	(-4.4796, 7.5E-06)	(-3.9404, 8.1E-06)	(-4.6455, 3.4E-06)
ZDT3	(-4.6455, 3.4E-06)	(-0.1585,>0.05) *no difference*	(-4.6455, 3.4E-06)	(-4.6455, 3.4E-06)	(-4.6455, 3.4E-06)
ZDT4	(-4.6455, 3.4E-06)	(-0.2903,>0.05) *no difference*	(-4.6455, 3.4E-06)	(-4.6455, 3.4E-06)	(-4.6455, 3.4E-06)
ZDT6	(-4.6455, 3.4E-06)	(-4.6041, 4.1E-06)	(-4.6455, 3.4E-06)	(-4.6455, 3.4E-06)	(-4.6455, 3.4E-06)
DTLZ2	(-4.6614, 3.1E-06)	(-4.8124, 1.5E-06)	(-3.6046, 3.1E-04)	(-3.6046, 3.1E-04)	(-4.3595, 1.3E-05)

function ZDT1 since the $I_{\varepsilon+}(\text{DMOPSO}, X_5) \approx 0$ and $I_{\varepsilon+}(X_5, \text{DMOPSO}) >> 0$, similarly, this applies to all algorithms. Figures 6.19-6.24 show that algorithm MOPSO seems to perform as well as DMOPSO for functions ZDT1 to ZDT6. Moreover, we can observe that DMOPSO has lower standard deviations, which are consistent with those shown in Figure 6.17. The box plot on DTLZ2 in Figure 6.24 may show that DMOPSO does not strictly dominate the rest of the MOPSOs since $I_{\varepsilon+}(\text{DMOPSO}, X_{1-5}) > 0$ and $I_{\varepsilon+}(X_{1-5}, \text{DMOPSO}) > 0$. For further analysis, the distributions of $I_{\varepsilon+}$ values are tested using the Mann-Whitney rank-sum test, which are presented in Table 6.6. Table 6.6 also confirms that MOPSO performs equally well as DMOPSO on function ZDT2. Hence, when we combine results given in Figure 6.24 and Table 6.6 for function DTLZ2, we can conclude that DMOPSO weakly dominates algorithms OMOPSO, MOPSO, cMOPSO,

and sMOPSO. In general, results in Table 6.6 and Figures 6.19-6.24 confirm that DMOPSO is significantly better than most or even all of the MOPSOs in terms of performance on all test functions.

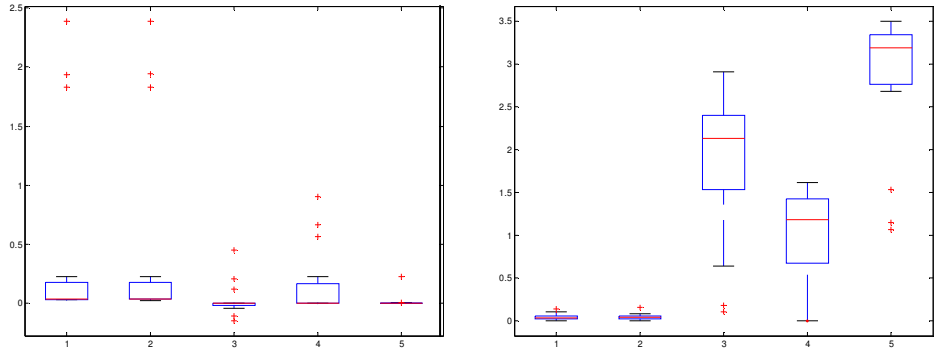
Table 6.6 The distribution of $I_{\varepsilon+}$ values tested using Mann-Whitney rank-sum Test [144]. The table presents the z values and p-values with respect to the alternative hypothesis (i.e., p-value < α=0.05) for each pair of DMOPSO and a selected MOPSO. In each cell, both values are presented in a bracket like this: (z value, p-value). For simplicity, DMOPSO is represented by *A*, and algorithms *B1* to *B5* are referred to as OMOPSO, MOPSO, cMOPSO, sMOPSO, and NSPSO, respectively. The distribution of DMOPSO is significantly difference or better than those selected MOPSO unless stated.

Test Functions	$I_{\varepsilon+}$ *(A,B1)* and $I_{\varepsilon+}$ *(B1,A)*	$I_{\varepsilon+}$ *(A,B2)* and $I_{\varepsilon+}$ *(B2,A)*	$I_{\varepsilon+}$ *(A,B3)* and $I_{\varepsilon+}$ *(B3,A)*	$I_{\varepsilon+}$ *(A,B4)* and $I_{\varepsilon+}$ *(B4,A)*	$I_{\varepsilon+}$ *(A,B5)* and $I_{\varepsilon+}$ *(B5,A)*
ZDT1	(-4.6637, 3.1E-06)	(-2.6546, 8.0E-03)	(-4.6896, 2.7E-06)	(--4.6896, 2.7E-06)	(-4.6896, 2.7E-06)
ZDT2	(-2.1983, 2.8E-02)	(-1.2029,>0.05) *no difference*	(-4.4401, 9.0E-06)	(-2.8169, 4.8E-03)	(-4.7088, 2.5E-06)
ZDT3	(-4.6637, 3.1E-06)	(-3.2353, 1.0E-03)	(-4.6637, 3.1E-06)	(-4.6748, 2.9E-06)	(-4.6748, 2.9E-06)
ZDT4	(-4.7088, 2.5E-06)	(-4.1063, 4.0E-05)	(-4.6896, 2.7E-06)	(-4.7332, 2.2E-06)	(-4.7636, 1.9E-06)
ZDT6	(-4.6455, 3.4E-06)	(-2.8205, 4.8E-03)	(-4.6455, 3.4E-06)	(-4.6455, 3.4E-06)	(-4.6455, 3.4E-06)
DTLZ2	(-4.6614, 3.1E-06)	(-4.8124, 1.5E-06)	(-4.8124, 1.5E-06)	(-2.3968, 1.6E-02)	(-1.6419,>0.05) *no difference*



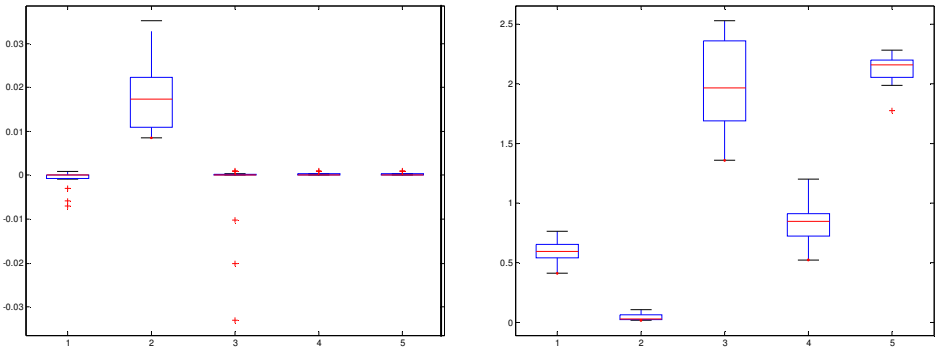
$I_{\varepsilon+}(DMOPSO, X_{1-5})$ $I_{\varepsilon+}(X_{1-5}, DMOPSO)$

Figure 6.19 Box plot based upon additive binary epsilon indicator ($I_{\varepsilon+}$ values) on test function ZDT1 (algorithms 1-5 are referred to as OMOPSO, MOPSO, cMOPSO, sMOPSO, and NSPSO, respectively).



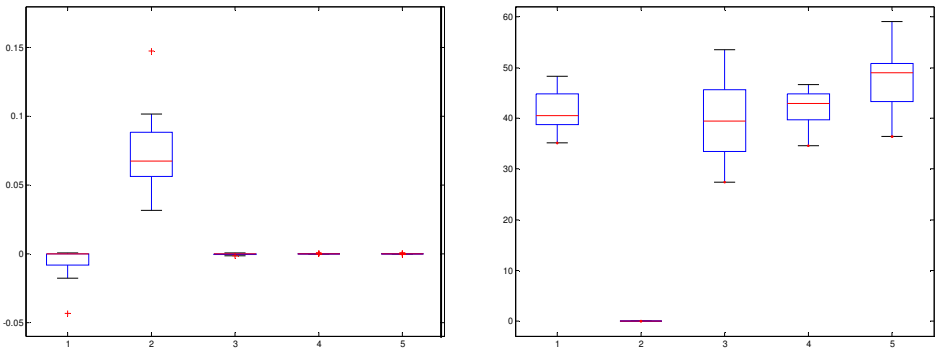
$I_{\varepsilon+}(DMOPSO, X_{1-5})$ $I_{\varepsilon+}(X_{1-5}, DMOPSO)$

Figure 6.20 Box plot based upon additive binary epsilon indicator ($I_{\varepsilon+}$ values) on test function ZDT2 (algorithms 1-5 are referred to as OMOPSO, MOPSO, cMOPSO, sMOPSO, and NSPSO, respectively).



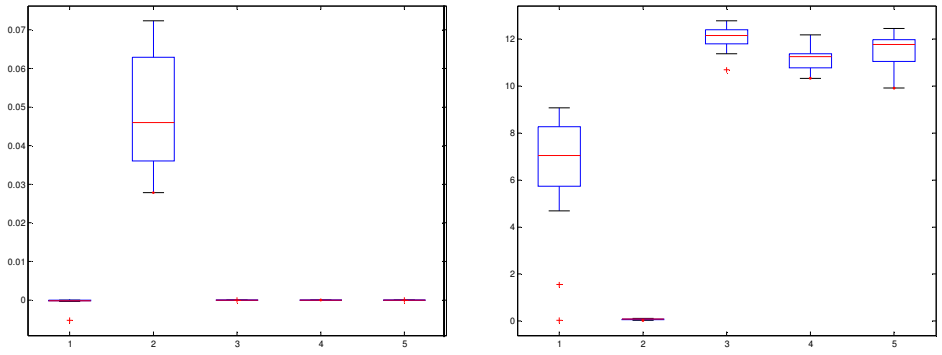
$I_{\varepsilon+}(DMOPSO, X_{1-5})$ $I_{\varepsilon+}(X_{1-5}, DMOPSO)$

Figure 6.21 Box plot based upon additive binary epsilon indicator ($I_{\varepsilon+}$ values) on test function ZDT3 (algorithms 1-5 are referred to as OMOPSO, MOPSO, cMOPSO, sMOPSO, and NSPSO, respectively).



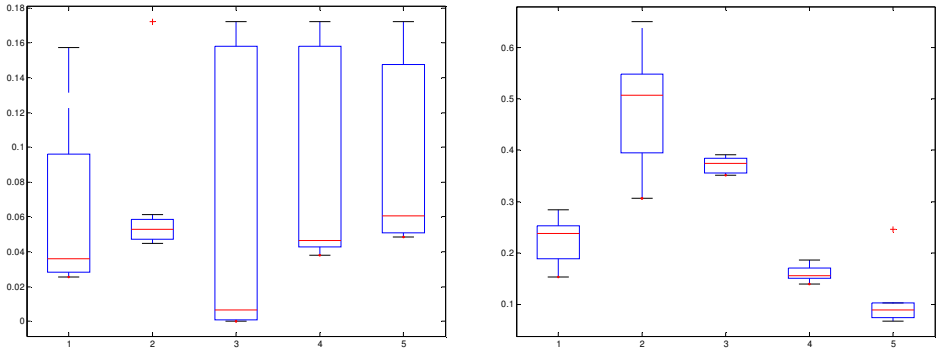
$I_{\varepsilon+}(DMOPSO, X_{1-5})$ $I_{\varepsilon+}(X_{1-5}, DMOPSO)$

Figure 6.22 Box plot based upon additive binary epsilon indicator ($I_{\varepsilon+}$ values) on test function ZDT4 (algorithms 1-5 are referred to as OMOPSO, MOPSO, cMOPSO, sMOPSO, and NSPSO, respectively).



$I_{\varepsilon+}(DMOPSO, X_{1-5})$ $I_{\varepsilon+}(X_{1-5}, DMOPSO)$

Figure 6.23 Box plot based upon additive binary epsilon indicator ($I_{\varepsilon+}$ values) on test function ZDT6 (algorithms 1-5 are referred to as OMOPSO, MOPSO, cMOPSO, sMOPSO, and NSPSO, respectively).

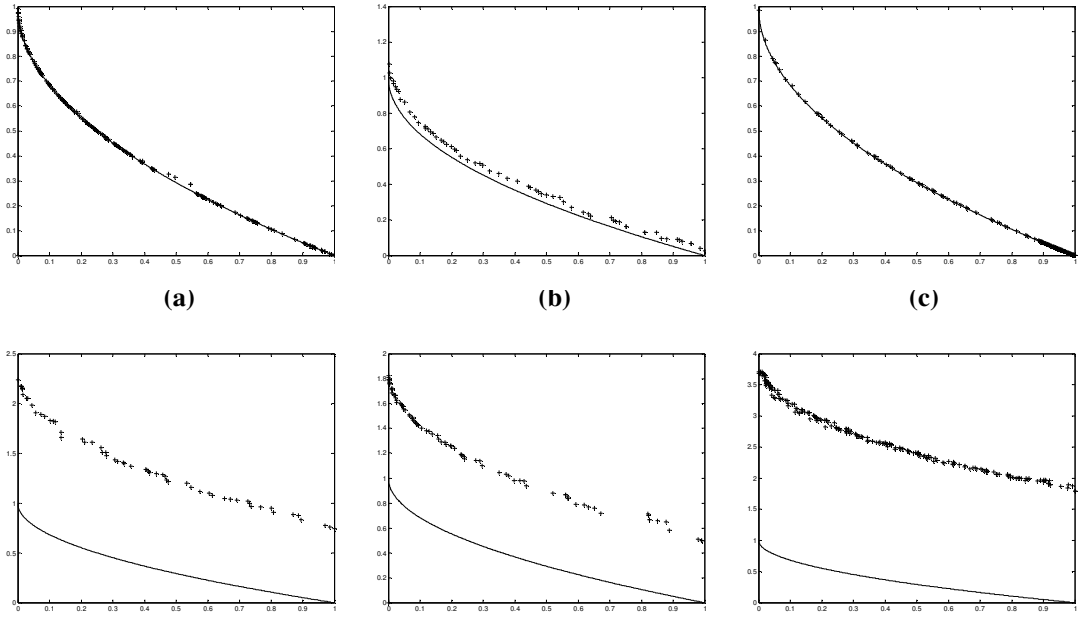


$I_{\varepsilon+}(DMOPSO, X_{1-5})$ $I_{\varepsilon+}(X_{1-5}, DMOPSO)$

Figure 6.24 Box plot based upon additive binary epsilon indicator ($I_{\varepsilon+}$ values) on test function DTLZ2 (algorithms 1-5 are referred to as OMOPSO, MOPSO, cMOPSO, sMOPSO, and NSPSO, respectively).

For qualitative comparison, the resulting Pareto fronts (from a single run) of the six MOPSOs on all test functions from the same initial population are illustrated in Figures 6.25-6.30. The figures show DMOPSO is able to find the well-extended, near-optimal Pareto fronts despite a large number of decision variables for test functions ZDT1 to ZDT6. MOPSO comes up second, where it can produce quality Pareto fronts similar to those produced by DMOPSO except for function DTLZ2. cMOPSO, sMOPSO, and NSPSO produce the worst Pareto fronts since they have difficulty in converging

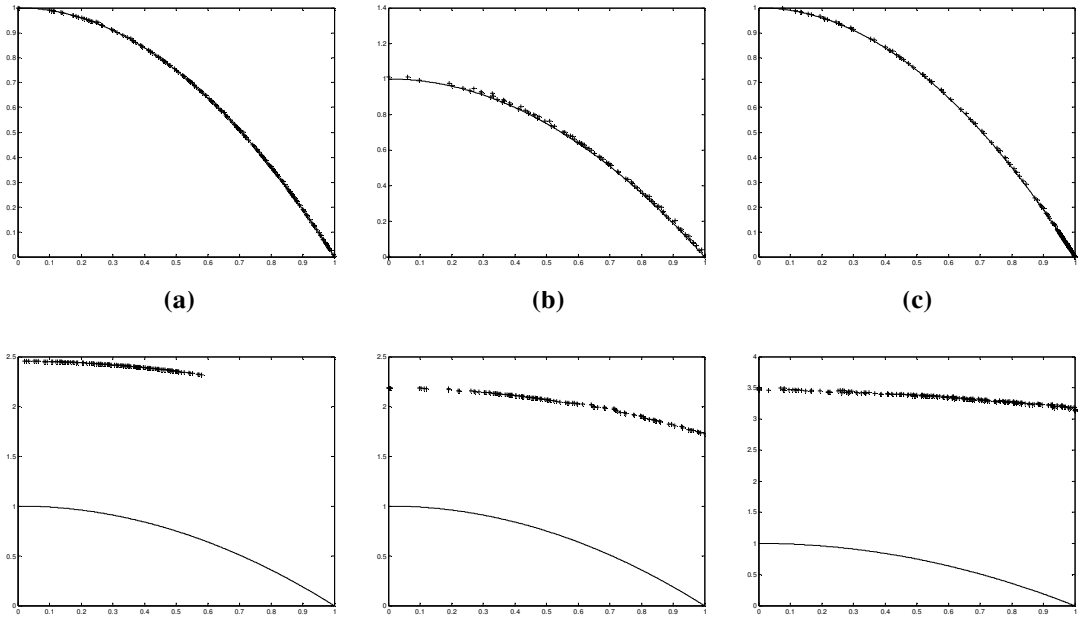
towards the true Pareto front, especially for functions ZDT1 to ZDT6 with high-dimensional decision spaces.



(a) **(b)** **(c)**

(d) **(e)** **(f)**

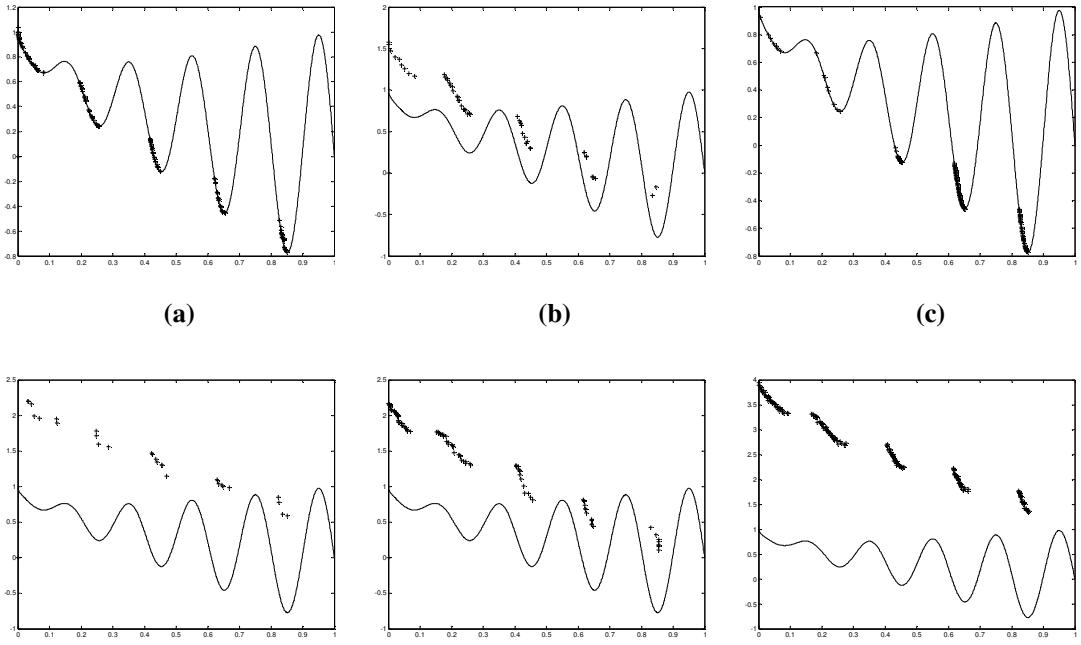
Figure 6.25 Pareto fronts produced by (a) DMOPSO, (b) OMOPSO, (c) MOPSO, (d) cMOPSO, (e) sMOPSO, and (f) NSPSO on test function ZDT1.



(a) **(b)** **(c)**

(d) **(e)** **(f)**

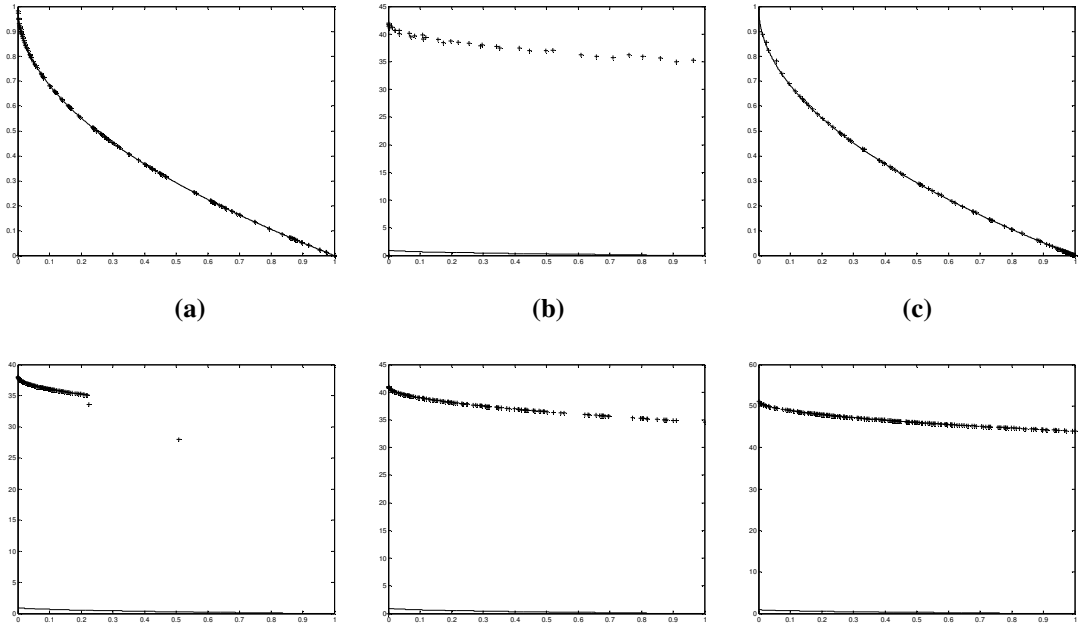
Figure 6.26 Pareto fronts produced by (a) DMOPSO, (b) OMOPSO, (c) MOPSO, (d) cMOPSO, (e) sMOPSO, and (f) NSPSO on test function ZDT2.



(a) **(b)** **(c)**

(d) **(e)** **(f)**

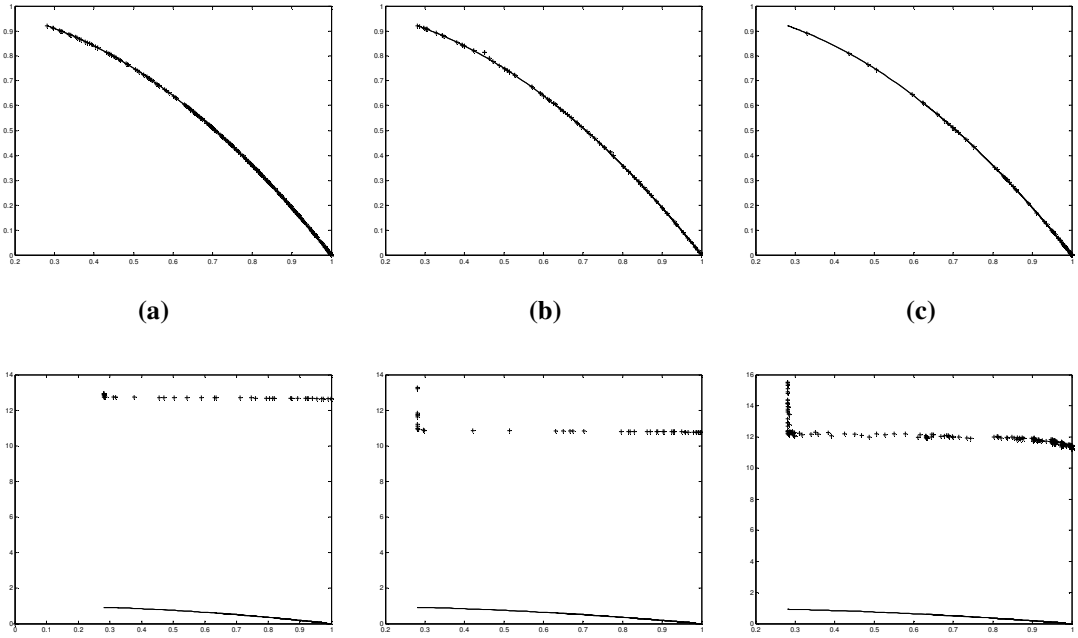
Figure 6.27 Pareto fronts produced by (a) DMOPSO, (b) OMOPSO, (c) MOPSO, (d) cMOPSO, (e) sMOPSO, and (f) NSPSO on test function ZDT3.



(a) **(b)** **(c)**

(d) **(e)** **(f)**

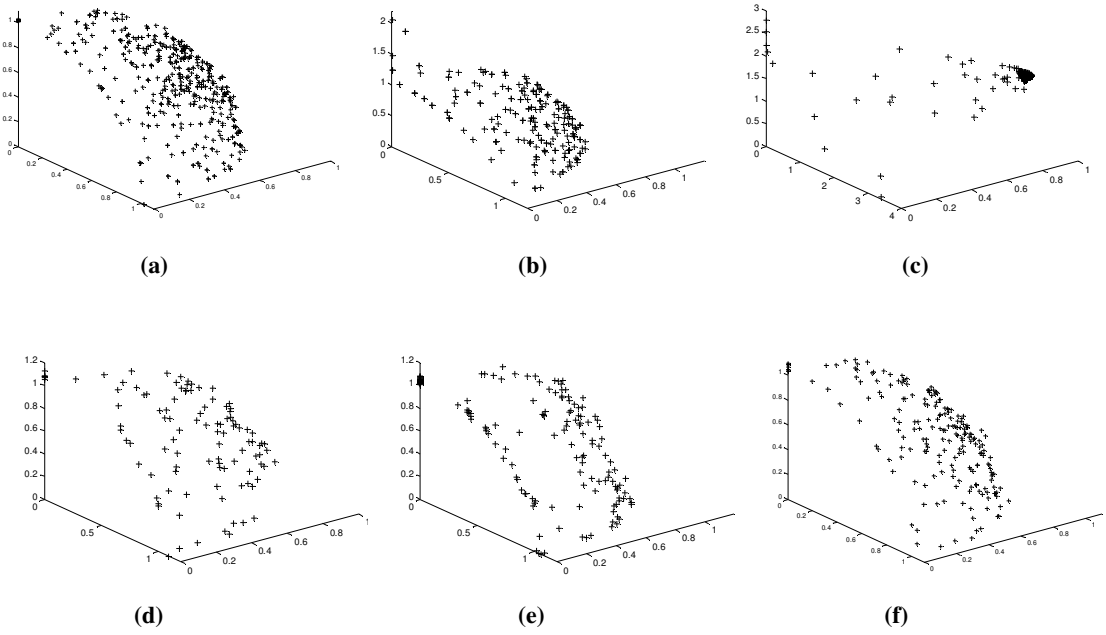
Figure 6.28 Pareto fronts produced by (a) DMOPSO, (b) OMOPSO, (c) MOPSO, (d) cMOPSO, (e) sMOPSO, and (f) NSPSO on test function ZDT4.



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure 6.29 Pareto fronts produced by (a) DMOPSO, (b) OMOPSO, (c) MOPSO, (d) cMOPSO, (e) sMOPSO, and (f) NSPSO on test function ZDT6.



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure 6.30 Pareto fronts produced by (a) DMOPSO, (b) OMOPSO, (c) MOPSO, (d) cMOPSO, (e) sMOPSO, and (f) NSPSO on test function DTLZ2.

6.4.5 Investigation of Computational Cost of DMOPSO with Selected MOPSOs

By introducing the dynamic population approach, DMOPSO produces better performances overall as compared to the selected MOPSOs. However, it is essential to investigate whether the dynamic population approach will increase the computational complexity.

Table 6.7 Average number of evaluations required per run for all test problems from all selected algorithms and DMOPSO to achieve GD =0.001.

	GD =0.001	DMOPSO	OMOPSO	MOPSO	cMOPSO	sMOPSO	NSPSO
ZDT1	Average No. Evaluations	**7270.6**	16140	7510	500000	500000	500000
ZDT2	Average No. Evaluations	**2983.3**	9060	4572	500000	500000	500000
ZDT3	Average No. Evaluations	**500000**	500000	500000	500000	500000	500000
ZDT4	Average No. Evaluations	**8856.2**	500000	9580	500000	500000	500000
ZDT6	Average No. Evaluations	**3190.8**	23880	5340	500000	500000	500000
DTLZ2	Average No. Evaluations	**18234**	23200	500000	59840	500000	268480

The investigation simply compares the required number of fitness evaluations needed by DMOPSO and the selected MOPSOs to achieve the targeted generational distance [45], *GD*, of 0.001 for the selected test problems. To avoid any MOPSO that could consume excessive computations to reach the goal set in *GD*, a limit of 500,000 fitness evaluations is imposed as the stopping criterion. To obtain the running time, a Matlab function is used to measure the time elapsed for each MOPSO. Each MOPSO performs 50 independent runs to collect the statistical results. All parameter settings for the chosen MOPSOs are the same as those shown in Table 6.2 and Table 6.3. Table 6.7 presents the average number of fitness evaluations and time needed per run for all of the selected MOPSOs and DMOPSO. Table 6.7 shows that DMOPSO demands the least average number of fitness evaluations as opposed to other selected MOPSOs to reach the

desired *GD* values for ZDT1, ZDT2, ZDT4, ZDT6, and DTLZ2. Except for ZDT3, all of the MOPSOs are unable to find the Pareto front with the targeted *GD* value of 0.001 within 500,000 fitness evaluations. Overall, it is observed that DMOPSO can save at least 8 percents of the required computational complexity in terms of number of fitness evaluation. In other words, DMOPSO delivers better performance with less computational complexity.

CHAPTER 7

PROPOSED ALGORITHM 2: DYNAMIC MULTIPLE SWARMS IN MULTIOBJECTIVE PARTICLE SWARM OPTIMIZATION (DSMOPSO)

. The second proposed MOPSO, called DSMOPSO, is described in this chapter. In this proposed algorithm, dynamic population concept is applied but in a different perspective. Instead of changing the population size as discussed in Chapter 6, number of swarms is adapted dynamically throughout the search process and the swarm size, i.e. number of particles in a swarm, is fixed and predefined by user. The objective is to promote diversity and local search capability to enhance the solution quality on the optimal Pareto front, and to eliminate the need to estimate an initial number of swarms to improve the computational cost without compromising the performance of the algorithm. In DSMOPSO, three novel strategies are incorporated: the dynamic swarm strategy to allocate an appropriate number of swarms as needed and justified, the modified PSO update mechanism to better manage the convergence and communication among and within swarms, and objective space compression and expansion strategy to progressively exploit the objective space during different stages of the search process. Experiments are conducted to evaluate the performance, as well as the required computational cost, of DSMOPSO against the selected MOPSOs. Sensitivity of the algorithm towards the setting of the involved parameters is also investigated.

7.1 Introduction

The multiple-swarm PSO bears a remarkable resemblance with the mixed-species flocking. In nature, there are certain bird species joined together in a flock to travel, to feed, and to collectively defend against any predators. Evidence indicates that increase in feeding efficiency may be the key motivation of rendering the formation of mixed-species flocks [153]. The birds in different species collaborate and share information among each other if any food sources are located. Different bird species may prefer different foods and acquire different foraging techniques. In addition, different species act as flock leaders under various environments to lead and influence the flocking behavior of a variety of bird species [154]. The number of species in a flock may vary depending upon the types of food sources available and the degree of competition among them. By analogy, the bird species join together in a flock to achieve certain foraging behaviors that will benefit each other, which is similar to the notion that multiple swarms in PSO explore the search space together to attain the objective of finding the optimal solutions, while different food preference in mixed species flocking corresponds to the tendency of multiple-swarm PSO in locating possible solutions in different regions in a fitness landscape. In addition, different species that assume the leadership under various environments is analogous to the notion that multiple swarms in PSO select their global leaders that would lead and influence their movement toward the best solution found so far. The information shared within a species and among species is also closely portrayed in multi-swarm PSO movement.

These evidences of analogy are found in publications, as discussed in Chapters 4 and 5, wherein multiple-swarm PSO is used to solve different optimization problems, particularly in multimodal function optimization [80-87], multimodal function

optimization in dynamic environments [88,89], single objective optimization problems (SOPs) [90-96], and more recently, multiobjective optimization problems (MOPs) [128]. Unlike what biology indicates in mixed-species flocking that the number of species involved varies dynamically, all of these multiple-swarm PSOs adopt the notion of using a heuristically chosen number of swarms with a fixed swarm size throughout the search process. Although a good algorithm design would guarantee a high probability of finding the Pareto optimal set, the number of swarms with a fixed swarm size indirectly contributes to the effectiveness and efficiency of the performance of an algorithm, particularly on the computational cost. If a multiple-swarm PSO employs an overly large number of swarms with a fixed swarm size, it will enjoy a better chance of discovering possible good solutions that lead to the optimal Pareto set, but inevitably suffer from an undesirable, high computational cost. This implies a limited food source that might induce excessive competition among a large number of bird species. On the other hand, an insufficient number of swarms will undermine chances of exploring the search space to discover potential good solutions, and coupled with PSO's high speed in convergence; this may lead to undesirable premature convergence or result in degraded quality of the optimal Pareto set. Again, one may suggest that a rough estimate of an "appropriate" population size may be adequate for a good design since one need not know the exact "optimal" number of swarms to solve an optimization problem. It would be the case for many single objective or multimodal problems, and for some MOPs that have lower numbers of objective functions or lower dimension of decision variables. Considering the cases where the MOPs have a large number of objective functions or a large dimension in decision variables, and even those MOPs qualified as the hard problems [152], this will

pose a great challenge to "estimate" an appropriate number of swarms to solve these MOPs without exerting excessive computational cost. In addition, without prior knowledge about the topology of the fitness landscape for an MOP, it might be unrealistic to expect an "appropriate" number of swarms can be determined to kickoff the search process. Hence, a compromised, yet effective, solution would be to dynamically adjust the number of swarms (with a fixed swarm size) to explore the search space in balance between computational cost and the attained performance throughout the search process. Hence, this motivated us to propose a multiobjective particle swarm optimization (MOPSO) that adaptively adjusts the number of swarms needed throughout the search process. This proposed algorithm is named dynamic multiple swarms in multiobjective particle swarm optimization (DSMOPSO).

7.2 Proposed Algorithm Overview

The proposed algorithm, dynamic multiple swarms in multiobjective particle swarm optimization (DSMOPSO), involves two key strategies: swarm growing strategy to allocate more swarms if necessary and justified, and swarm declining strategy to eliminate swarms that wouldn't contribute in search for Pareto front. Additional designs are included to support the above two strategies. These designs include 1) cell-based rank density estimation scheme to effectively keep track of the rank and density values of the particles (swarm members); 2) objective space compression and expansion strategy to adjust the size of the objective space whenever needed to progressively search for high precision true Pareto front; 3) PSO updating equation is modified to exploit its usefulness

and to accommodate the multiple-swarm concept; and 4) swarm local best archive is updated based on the progression of their swarm representative, the swarm leaders.

```
Begin
Parameters initialization for cell-based rank density estimation
scheme, crowdedness indicator, age indicator, and objective space
compression and expansion strategy.

/*Initialization
Set swarm size
Randomly generate one swarm
Set Maximum iterations (tmax)
Set iteration t = 0

For each particle
    Fitness evaluation
    Rank_and_density_estimation()
EndFor
Identify swarm leaders
Update_swarms_localbest()
Archive_maintenance

t = 1
While t < tmax
    Objective_space_compression_expansion_strategy()
    Swarm_growing_strategy()
    Swarm_declining_strategy()

    For each swarm
        For each particle
            Flight()
            Fitness evaluation
            Rank_and_density_estimation()
        EndFor
        Identify swarm leaders
        Update_swarms_localbest()
        Archive_maintenance
    EndFor
    t = t + 1
EndWhile
Report results in archive
End
```

Figure 7.1 Pseudocode of DSMOPSO.

The generic steps of DSMOPSO are as follows: At iteration $t = 0$, with predefined swarm size, a single swarm is generated, and cell-based rank density estimation scheme is calculated to setup the rank and density values of the swarm

members. Next, the swarm leader is identified. Third, every swarm local best is recorded and the fittest swarm members are stored in the archive. When iteration step is increased, the condition to evoke the objective space compression and expansion strategy is checked. If the condition is satisfied, objective space compression expansion strategy is performed. Otherwise, we jump to the next step. Swarm growing strategy is applied to increase the number of swarms while swarm declining strategy is employed to control the number of swarms in the swarm population. Then, the swarms perform flight. Again, the following steps are repeated for all swarms: 1) update the swarms' information via the cell-based rank density estimation scheme; 2) identify swarm leaders; 3) update their local best; and 4) perform archive maintenance. Then, the loop goes back to objective space compression expansion strategy subroutine. Once maximum iteration is achieved, the solutions in the archive are the best Pareto front found.

7.3 Implementation Details

The detail of all the key steps in Figure 7.1 is elaborated in the following subsections.

7.3.1 Cell-based Rank Density Estimation Scheme

As the number of swarms varies every iteration, the swarm population size will modify as well. This modification, i.e., adding or removing particles, will affect Pareto rank of the existing particles and the population density of certain areas located in the objective space. This poses a problem of needing to recalculate the Pareto rank and density values of the particles to keep up the changes of the swarm population size. To

counter the problem, we employed an existing scheme, cell-based rank and density estimation scheme, which has been discussed in Subsection 6.3.1.

7.3.2 Identify Swarm Leaders

Every swarm has its own set of "swarm members." The number of "swarm members," also called swarm size, is determined by a user-specified parameter, *ssize*. Each swarm has its own representative; the representative is named "swarm leader." The swarm leaders are decided based on the idea of best "candidate" among its "swarm members." Hence, to choose a swarm leader, it has to have the best rank value (i.e., least rank value for minimization problems). If more than one swarm member shares the same best rank value, a swarm leader is randomly chosen among them. Selection of swarm leaders is done at every iteration. Choosing swarm leaders based on their rank values indicates how much the swarms have progressed in finding the optimal Pareto front. In addition, these swarm leaders will be the deciding factor for some of the procedures such as updating swarms' local bests and dynamic swarm number strategy. In this chapter, the notation for the swarm leader of swarm n is represented by $sLeader^n$

7.3.3 Update Local Best of Swarms

As mentioned in Subsection 7.2, PSO equation is modeled in such that the particles learn from their own experiences and from the success of their peers. To achieve the former objective, the particles' own personal best positions attained so far are updated at every iteration step, wherein this information is later used to update particle velocities and the particle positions in the search space. For multiple swarms, a similar procedure is

applied here and the term 'swarm local best archive' is used to indicate the swarms' own personal best positions. Figure 7.2 summarizes the steps involved for updating the swarm local best archive. The procedure to update the swarms' local best position by comparing the rank values of swarm leaders in the current iteration with those recorded in the swarm local best archive. Consider minimization problems, the procedures are summarized as below:

- If the swarm local best archive is empty or the reinitialized parameter (St) is triggered, record rank values of all swarm leaders, their corresponding positions, and the positions of their respective swarm members.
- If the swarm local best archive is nonempty, the rank values of the swarm leaders ($Lprank$) in the current iteration are compared with those recorded in the swarm local best archive. Any of the current swarm leaders that have lesser rank values are identified; their rank values, their positions, and the positions of their corresponding swarm members ($Lpbest$) will replace the recorded ones. If the rank values of a current swarm and its recorded swarm leader have the same rank value, then pure Pareto ranking method [38] is applied to both of the swarm leaders. If the current swarm dominates the recorded swarm leader, then the current one will replace the recorded one. If both do not dominate each other, one of them is randomly chosen to update the swarm local best archive.

7.3.4 Archive Maintenance

A fixed size archive is implemented in DSMOPSO to record any good particles (nondominated solutions) found during the search process and these solutions in the

```
Function Update_swarms_localbest(Lrank, LPrank, LPbest, t, St, n_swarm, swarms)

   /*Lrank = current rank value of swarm leaders
   /*LPrank = rank of local best for swarm leaders
   /*LPbest = local best of the swarm leaders' groups
   /*t = current iteration;
   /*St = parameter indicate a need to reinitialize Lprank and Lpbest
   /*n_swarm = number of swarms
   /*swarms = current swarms
   Begin
      If (t = 0) ∪ (St = 0)
         LPbest = swarms
         LPrank = Lrank
      Else
         For i=1 to n_swarm
            If Lrank(i) < LPrank(i)
               Update LPrank(i) and LPbest(i)
            ElseIf Lrank(i) = LPrank(i)
               Find swarm leader(i) current position, Lswarm
               Find swarm leader(i) local best position, LPbest(i)
               Compute the fitness values for Lswarm and LPbest(i)
               Check Pareto dominance of the fitness values
               Update LPbest(i) if fitness of Lswarm is better
            EndIf
         EndFor
      EndIf
   End
```

Figure 7.2 Pseudocode of update local best for the swarm leaders.

archive serve as potential global best candidates (*gbest*) for the particles. At each iteration count, new solutions are compared with respect to any members in the archive. If new solutions are not dominated by any archive members, they are accepted into the archive. Similarly, any archive members dominated by any new solutions are removed from the archive. If the archive population size exceeds the allocated archive size, then crowding distance [31] is applied to remove the crowded members and to maintain uniform distribution among the archive members. There are existing methods for the particles to select their global leaders (*gbest*) [106,112,113,115-117]. In this paper, the crowding distance values of the archive members are used to guide the particles to select

their *gbest* . Larger crowding distance values imply archive members are less crowded, and are likely to be selected as particles' *gbest* . Once the search process is terminated, the solutions in archive become the final Pareto front.

7.3.5 Particle Update Mechanism (Flight)

A major problem in employing multiple-swarm concept is the need to exchange information among swarms, especially if no mutation operator is incorporated. Without information exchange, the particles may find several disconnected segments of a Pareto front due to lack of diversity among swarms. Hence, information exchange among swarms is vital in promoting diversity among swarms. In recent work, Yen and Daneshyari [96] adopted a three-level PSO updating rule wherein the particles learn their experiences based on personal, neighborhood, and global levels to adjust their flying speed and direction in the search space. The idea is to further enhance the information sharing among particles by incorporating the concept of neighborhood in the updating PSO equation.

Though diversity among swarms is essential, diversity within a swarm is equally important. In [88], the swarm members of each swarm are splitted according to the user defined configuration, i.e., the swarm members composed of either neutral and charged particles or neutral and quantum particles. For example, for the formal configuration, for a given swarm size, part of the swarm members are neutral particles and the remaining are charged particles. Based on the configuration, the particle updating rule is dependent on the particle's types, which are neutral, charged, or quantum particles. By employing this strategy, the diversity within a swarm is encouraged.

Inspired by both forms of swarms interaction discussed above, we propose revised PSO update rules. The rules involve three forms of communication as follows:

PSO update rule 1: Allow particles in a swarm to update using three-level PSO updating rule. This will allow swarms to share information from the global leaders (archive), their swarm leaders, and their personal best achievement. The new velocity and position equations are given.

$$v_{i,j}^{n}(t+1) = w \times v_{i,j}^{n}(t) + c_1 \times r_1 \times \left(pbest_{i,j}^{n} - x_{i,j}^{n}(t)\right) + c_2 \times r_2 \times \left(gbest_{j} - x_{i,j}^{n}(t)\right) + c_3 \times r_3 \times \left(Lpbest_{j}^{n} - x_{i,j}^{n}(t)\right) \quad (7.1)$$

$$x_{i,j}^{n}(t+1) = x_{i,j}^{n}(t) + v_{i,j}^{n}(t+1) \quad (7.2)$$

where $v_{i,j}^{n}(t)$ is the jth dimensional velocity of swarm member *i* of swarm *n* in iteration *t*; $x_{i,j}^{n}(t)$ is the jth dimensional position of swarm member *i* of swarm *n* in iteration *t*; $pbest_{i,j}^{n}$ denotes the jth dimensional local best position of the swarm members *i* of swarm *n* in iteration *t*; $gbest_{j}$ is the jth dimensional global best selected from archive in iteration *t*; $Lpbest_{j}^{n}$ is the jth dimensional local best position of swarm leader of swarm *n* in iteration *t*; r_1, r_2, and r_3 are random numbers within $[0,1]$ that are regenerated every time they occur; w is the inertial weight; and c_1, c_2 and c_3 are the acceleration constants. Note that the each of the acceleration constants is randomly varied between 1.5 and 2 at every iteration to provide different emphasis on the components in Equation (7.1) and to deal with the difficulty in choosing the "optimal" settings for these constants to prevent the particles' velocities from exploding. The inertial weight is randomly varied between 0.1 and 0.5 to encourage exploration and local search in different iteration counts.

PSO update rule 2: Allow particles in a swarm to update via perturbation around their corresponding swarm leader. This will facilitate local search and promote diversity within a swarm. Basically, this rule can be achieved via perturbation concept. The area of perturbation is determined by parameter r_d, a random number generated from a Gaussian distribution with zero mean $(\mu = 0)$ and a variance (σ^2). For simplicity, parameter σ^2 is set to 0.1 to limit the perturbation region around the swarm leader and to prevent swarm members from moving too far from each other. Note that the setting for parameter σ^2 depends on the user's preference and the size of the decision space. The center point of this area is the swarm leader as shown in Equation (7.4). Both Equations (7.3) and (7.4) are similar to the quantum particle updating rule in [145],

$$r_d = Gaussian\ (0, 0.1) \tag{7.3}$$

$$x_{i,j}^n(t+1) = Lpbest_j^n + r_b \tag{7.4}$$

PSO update rule 3: Under some conditions, the particles should exchange information with a leader other than their own. Equations (7.1) and (7.2) are implemented to update the particles in each swarm. The only modification is the $Lpbest_j^n$ term in Equation (7.1) is replaced by $Lpbest_j^{\tilde{n}}$, in which the superscript $\tilde{n}$ is to indicate that the swarm members can choose any swarm leaders other than their own swarm leader.

Randomly splitting the swarm members and delegating them to both PSO updating rules 1 and 2 encourage the swarm members to contribute two separate goals since PSO updating rule 1 promotes convergence, discovery, and improves good solutions; while PSO updating rule 2 encourages local search and diversity within a swarm. On the other hand, PSO updating rule 3 promotes convergence and diversity among swarms. For

simplicity, a random number r_e with uniform distribution between $[0,1]$is generated to decide which PSO update rule(s) to carry out. If $r_e > 0.5$, the swarms are updated by PSO update rules 1 and 2. The first half of the swarm members in a swarm is updated via rule 1, while the rest are updated using rule 2. On the other hand, if $r_e \leq 0.5$, then the swarm members are updated using PSO rule 3. Figure 7.3 presents the pseudocode of updating the particles (flight). Note that 0.5 is chosen without any prior knowledge as the deciding factor to provide equal probability for information exchange within a swarm and among swarms.

```
Function Flight(swarms, n_swarm , ssize)

   /*swarms = current swarms
   /* n_swarm = number of swarms
   /*ssize = swarm size
   Begin
        If ssize is even
           N = ssize/2
         Else
           N = (ssize/2) + 0.5
        EndIf
        For each swarm
           r_e = rand[0,1]
           If (r_e > 0.5) ∩ (n_swarm > 2)
               1:N particles are updated via Equations (7.1) and (7.2)
                  Generate r_d using Equation (7.3)
               (N+1): ssize particles are updated via Equation (7.4)
            Else
               Particles are updated via Equations (7.1) and (7.2) with a
randomly assigned swarm leader
          EndIf
        EndFor
   End
```

Figure 7.3 Pseudocode of updating the particles.

7.3.6 Swarm Growing Strategy

DSMOPSO employs two independent strategies—swarm growing strategy and swarm declining strategy to manage the number of swarms needed during the different stages of search process. Similar to the motivation given in [140], the first strategy aims to increase the number of swarms and to ensure every swarm to survive a sufficient number of iterations so that it can contribute to the search process in finding better solutions, while swarm declining strategy is applied to control the number of swarm from growing excessively. In this subsection, swarm growing strategy is discussed and swarms declining strategy will be discussed next.

In previous chapter, DMOPSO proposed a population growing strategy based on the concept proposed by Tan *et al*. [135]. The design involves procedures such as selecting potential particles to be perturbed, determining number of perturbations, and deciding where to perform perturbation. The improved design in DSMOPSO carefully addresses this deficiency and includes utilizing rank and crowdedness indicators to select potential candidates for swarm leaders and applying Voronoi diagram to generate a swarm of particles from a swarm template. First, rank and crowdedness indicators are introduced, and then the procedures for swarm growing strategy are elaborated.

The rank and crowdedness indicators are the same as elaborated in Subsection 6.3.4. The only difference is these indicators are applied to the swarm leaders instated of the particles in the swarm population presented in Chapter 6. Figure 7.4 shows how the R and D values of the swarm leaders **E** and **F** are determined. Refer to Figure 7.4, the swarm leader **E** (in Figure 7.4(a)) resides in the cell with rank value "1" (rank matrix in Figure 7.4(b)). Hence, apply Equation (6.12), its R value equal to 1 as shown in Figure

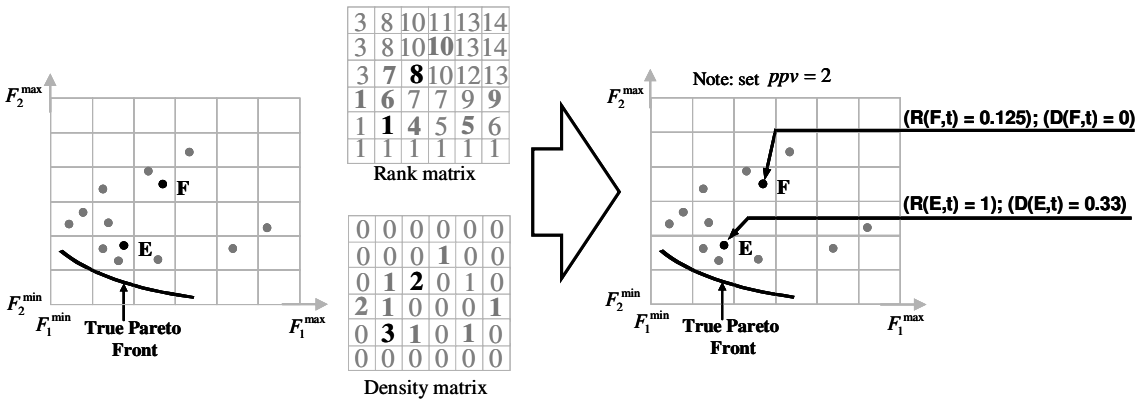


Figure 7.4 (a) Swarm leaders and their locations on the objective space, (b) rank matrix (Top) and density matrix (Bottom) of the swarm leaders, and (c) *R* and *D* values for swarm leaders E and F.

7.4(c). Meanwhile, those swarm leaders located in cell with higher rank values will result in a lower *R* value. For example, Figure 7.4(a) shows swarm leader **F** resides in cell with rank value "8", which is higher rank value than swarm leader **E**. So, its *R* value is much lower, which is 0.125. Hence, the *R* value can be used to quantify the chances for a swarm of being eliminated or to decide if a swarm leader is chosen as potential candidate for generating new swarms. Now, refer to the density matrix in Figure 7.4(b), the density value for swarm leaders **E** and **F** are 3 and 2 respectively. With the *ppv* value set to 2, Equation (6.13) indicates that the *D* value of swarm leader **E** is 0.33 while since the density value is lower or equal to *ppv*, the *D* value of swarm leader **F** is 0 (refer to Figure 7.4(c)). Hence, crowdedness indicator can be use to measure if the swarm leaders reside in a congested cells by comparing their density values with the *ppv* value.

The following three procedures outline the swarm growing strategy proposed.

Procedure 1: Identify potential swarm leaders from the swarm local best archive to generate "swarm templates" that will be used to create new swarms. The chosen swarm leaders should have the highest probability of producing new swarms that will

improve convergence toward the Pareto front and land on the unexplored areas in the objective space. In this paper, rank and crowdedness indicators are used to quantify the potential of the swarm leaders. First, gather the swarm leaders from the swarm local best archive. Second, the *R* and *D* values of the swarm leaders are computed using Equations (6.12) and (6.13). Third, for those cells that have more than one swarm leader, their local pure Pareto rank values are calculated using the Pareto ranking scheme proposed by Goldberg [146]. The local pure Pareto rank values are denoted as r_L. If only one swarm leader is in the cell, then r_L is equal to one by default. Finally, at iteration *t*, the likelihood that the swarm leader *i* with $R(i,t)$, $D(i,t)$ and $r_L(i,t)$, to be chosen is computed using the following equation:

$$l_g(i,t) = R(i,t) \times \left(\frac{1}{r_L(i,t)} \right) \times (1 - D(i,t)). \tag{7.5}$$

Equation (7.5) implies that those swarm leaders with higher *R*, lower *D*, or lower r_L values will have a high likelihood value, l_g. Those swarm leaders with higher likelihood value, l_g have a higher chance of being selected as potential candidates to generate the "swarm templates." Refer to Figure 7.5(c), the l_g values for swarm leaders **E** and **F** are 0.335 and 0.125, respectively. Based on the l_g values, swarm leader **E** has a higher chance of being chosen as the potential swarm leader compared to swarm leader **F**. To determine whether a swarm leader *i* will be chosen, a random number with uniform distribution between $[0,1]$ is generated to compare with the likelihood $l_g(i,t)$. If the likelihood is larger than the random number, then swarm leader *i* is selected as a potential candidate to generate a "swarm template," $\mathbf{x}_{new}$. The swarm template ($\mathbf{x}_{new}$) is generated

via a uniform mutation operator with the mutation rate equal to $1/number\ of\ dimensions\ in\ decision\ space$ [49].

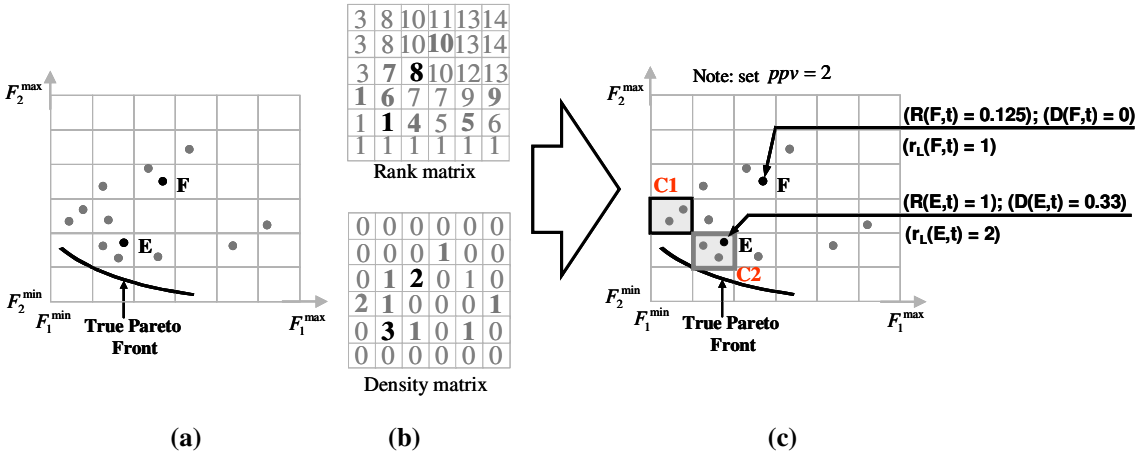


Figure 7.5 (a) Swarm leaders and their locations on the objective space, (b) rank matrix (Top) and density matrix (Bottom) of the swarm leaders, and (c) *R*, *D*, and r_L values for swarm leaders E and F.

Procedure 2: Once the swarm template ($\mathbf{x}_{new}$) is generated via Procedure 1, the template is perturbed to generate a swarm of particles. In order for perturbation to happen, the perturbation region centered around $\mathbf{x}_{new}$ needs to be defined. In this design, the motivation of employing Voronoi diagram [147] to determine the perturbation region is as follow: 1) the shape of the perturbation region is self-adapted and depends on the distribution of the solutions in the chosen dimensions; and 2) no user-defined parameter or a fixed model to define the perturbation region as proposed in DMOPSO is required. There are three key steps in this procedure to generate a member in the new swarm.

1) *Generate a Voronoi Diagram*: Firstly, in addition to $\mathbf{x}_{new}$, a particle from every swarm is randomly selected. For example, Figure 7.6 shows eight randomly selected particles from eight different swarms in addition to $\mathbf{x}_{new}$. Second, two dimensions

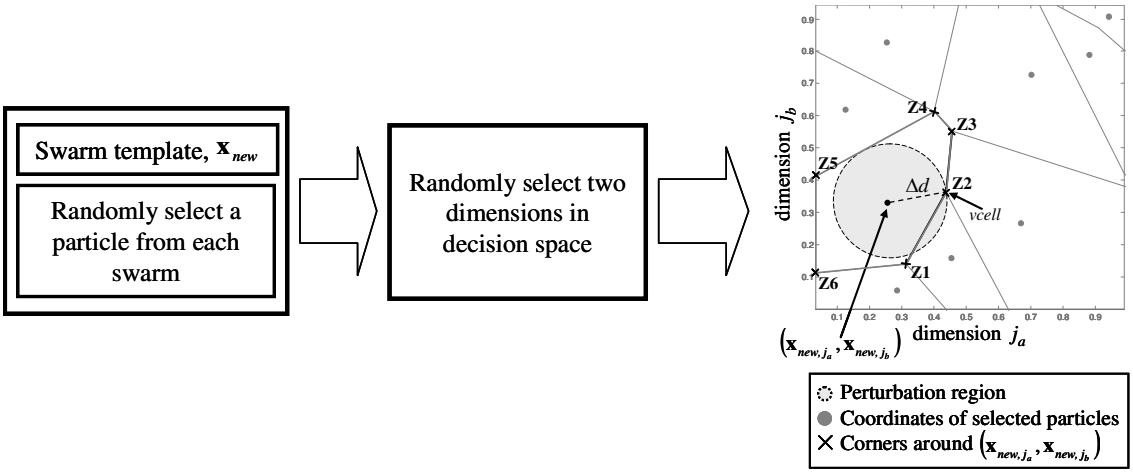


Figure 7.6 Block diagram depicts how an example Voronoi diagram of eight randomly selected particles and $\mathbf{x}_{new}$ is generated.

within the total number of dimensions in the decision space are randomly selected. These selected dimensions are denoted as j_a and j_b. Third, the dimension j_a and j_b of the selected particles and $\mathbf{x}_{new}$ represent the data points and are used to generate the Voronoi mesh, as presented in the reduced two dimensional space in Figure 7.6. Refer to Figure 7.6, the selected particles and $\mathbf{x}_{new}$ are represented as "circle." Also, the x-axis corresponds to dimension j_a and the y-axis corresponds to dimension j_b. In this study, only two dimensions are selected to build the Voronoi diagram for computational simplicity.

2) *Determine the Perturbation Region*: The black circle in Figure 7.6 represents the coordinate of $\mathbf{x}_{new}$, i.e., $(\mathbf{x}_{new,j_a}, \mathbf{x}_{new,j_b})$. Since there are more than two corners around the coordinate of $\mathbf{x}_{new}$ (i.e., represented with 'x' symbol and labeled as Z1 to Z6 in Figure 7.6), a corner is randomly selected. In Figure 6, the selected corner is Z2 and denoted as *vcell*. The distance between the center and Z2, i.e., Δd, is computed to form the perturbation region of $\mathbf{x}_{new}$.

Function newswarm = Generate_Swarm($\mathbf{x}_{new}$, *ssize*, *swarms*)

```
/* x_new  = a new swarm template
/* ssize = swarm size
/* swarms = current swarms

Begin
   For 1 to ssize
      Randomly choose a particle from every swarm
      Store chosen particles to T
      Randomly choose two dimensions, [j_a, j_b], from [1 max(dimension of
      x_new )]
      Draw out the two dimensions from every particle in T
      Draw out the two dimensions from x_new , i.e. (x_{new,j_a}, x_{new,j_b})
      Use all drawn values to calculate the Voronoi diagram
       Randomly choose a coordinate ( vcell ) around the Voronoi cell,
        where (x_{new,j_a}, x_{new,j_b}) coordinate is its center
      Find distance ( Δd ) between vcell and (x_{new,j_a}, x_{new,j_b}) coordinate
      Compute Equations (7.6), r_g
      r_g = rand[0,1]
      If  r_g > 0.5
          Add (Δd × r_g) to  x_{new,j_a}
      Else
          Add (Δd × r_g) to  x_{new,j_b}
      EndIf
   EndFor
End
```

Figure 7.7 Pseudocode of generating a new swarm via Voronoi procedure.

3) *Generate a Swarm Member*: Once a corner is selected, a swarm member is generated by applying the following equations:

$$r_g = Gaussian\ (0,0.1), \tag{7.6}$$

$$\mathbf{x}_{new,j_a}(t) = \mathbf{x}_{new,j_a}(t) + (\Delta d \times r_g) \tag{7.7a}$$

$$\mathbf{x}_{new,j_b}(t) = \mathbf{x}_{new,j_b}(t) + (\Delta d \times r_g), \tag{7.7b}$$

$$\mathbf{x}_{new,j}(t) = \mathbf{x}_{new,j}(t), \quad (j = 1,\ldots,N,\ j \neq j_a,\ j \neq j_b), \tag{7.7c}$$

where r_g is a random number generated from a Gaussian distribution with zero mean $(\mu = 0)$ and a variance (σ^2) of 0.1 (Equation (7.6)); and Δd is the distance between the center and the selected corner. Equations (7.7a) - (7.7c) are applied to generate a new swarm member. Please note only two chosen dimensions will be perturbed, while other dimensions remain the same as those in $\mathbf{x}_{new}$. This procedure is repeated until all the swarm members in a newly created swarm are generated. Figure 7.7 presents the pseudocode of generating a new swarm via Voronoi procedure.

```
Function Swarm_growing_strategy(LPrank, LPden, ppv , h_L , ssize, swarms)

   /*LPrank = rank of local best for swarm leaders
   /*LPden = density of local best for swarm leaders
   /* ppv = desired population size per cell
   /* h_L = home address of swarm leaders
   /*ssize = swarm size
   /*swarms = current swarms

   Begin
      Compute Rank Indicator, R, using LPrank
      Compute Crowdedness Indicator, D, using LPden and ppv
      Compute local rank value, r_L using h_L and Pareto ranking scheme
      Compute Rank Indicator for local rank value, R_L , using r_L
      Compute l_g using Equation (7.5)
      r_h = rand[0,1]
      If l_g > r_h
          Find and store 'chosen' swarm leaders to P
          Number of 'chosen' swarm leader = ns
      EndIf
      While count ≤ ns
          Select a swarm leader, x from P
          Generate a new particle (seed), x_new , by applying perturbation
          newswarm = Generate_Swarm( x_new , ssize, swarms)
          count = count + 1
      EndWhile
   Collect all the new swarms
   End
```

Figure 7.8 Pseudocode of swarm growing strategy

Applying Voronoi concept to determine the perturbation region presents a unique advantage as we do not need to define the perturbation region using a user-specified parameter or a fixed model. When the number of swarms is small in the early stage, the resulted Voronoi diagram will have large Voronoi mesh, which leads to a large perturbation region. Hence, support in generating swarms with swarm members farther away from each other promotes diversity within a swarm and allows the swarm members to explore larger unvisited regions in the objective space. On the other hand, when the number of swarms is large, which is often the case as the swarms are approaching the Pareto front, the area of perturbation region will be small according to the above design. A smaller perturbation region will generate swarms with swarm members closer to each other. This will encourage a local search within the swarm members towards the later stage of the search. Figure 7.8 presents the pseudocode of swarm growing strategy.

7.3.7 Swarm Declining Strategy

As mentioned earlier, the swarm declining strategy is proposed to control the number of swarms from growing excessively. The condition to remove a swarm is based on three qualitative indicators. Two of the three indicators are introduced in subsection 6.3.3, i.e., Equations (6.12) and (6.13). An additional indicator is known as the age indicator, which is used to measure the "lifespan" of the swarms. The age indicator ensures that those swarms generated recently, especially those newly generated swarms, will have enough lifespan to contribute to the search process. Assume at iteration *t*, the age of swarm leader *i* is denoted as $age(i,t)$ and its age indicator at iteration *t*, $A(i,t)$, is given by

$$A(i,t)=\begin{cases}1-\dfrac{A_{th}}{age(i,t)}, & if\ age(i,t)>A_{th}\\ 0, & otherwise\end{cases}, \tag{7.8}$$

where A_{th} is the predetermined age threshold. Equation (7.8) implies that any swarms' ages smaller than A_{th} will not be removed. When a particle is created, its age will be set at 0 and its age will be increased by one if it survives another iteration. Simulation study also indicates the performance of the DSMOPSO is not sensitive to the choice of the age threshold.

Two different likelihoods of removing swarms, utilizing three indicators mentioned above, are applied here. These indicators for the swarm leaders in swarm local best archive are computed. Two different likelihoods are as follows:

1) Likelihood of Removing the Swarms with Higher Rank Values: At iteration *t*, the likelihood that the swarm leader *i* with rank indicator $R(i,t)$ and age indicator $A(i,t)$, to be eliminated is computed using the following equation:

$$l_1(i,t)=(1-R(i,t))\times A(i,t). \tag{7.9}$$

Equation (7.9) implies that for those swarm leaders located farther away from the non-dominated solution and have exceeded the age threshold, A_{th}, should have a higher likelihood of being eliminated. This implies any bird species that fail to catch up with the mixed-species flock after traveling together for some time will likely be lost from the flock.

2) Likelihood of Removing the Swarms in the Same Cell with Rank Values Having Reached One: At iteration *t*, the likelihood that the swarm leader *i* with local rank

```
Function Swarm_declining_strategy(age, LPrank, LPden, ppv , t , A_th , h_L )

   /*age = age value of swarms
   /*LPrank = rank of local best for swarm leaders
   /*LPden = density of local best for swarm leaders
   /* ppv = desired population size per cell
   /* t = current iteration
   /* A_th = age threshold
   /* h_L = home address of swarm leaders
   Begin
      For each swarm
           r_k = rand[0,1]
           r_l = rand[0,1]
           Compute Rank Indicator, R, using Lprank
           Compute Crowdedness Indicator, D, using LPden and ppv
           Compute Age Indicator, A, using age and A_th
           Compute local rank value, r_L , using h_L and Pareto ranking
scheme
           Compute Rank Indicator for local rank value, R_L , using r_L
           Compute l_1 and l_2 using Equations (7.9) and (7.10)
respectively
           If l_1 > r_k ∪ l_2 > r_l
             Record swarms index to M
           EndIf
      EndFor
      Remove swarms recorded in M
   End
```

Figure 7.9 Pseudocode of swarm declining strategy.

value $r_L(i,t)$, density indicator $D(i,t)$, and age indicator $A(i,t)$, will be eliminated is given by

$$l_2(i,t) = \left(1 - \frac{1}{r_L(i,t)}\right) \times \left(D(i,t) - 1\right) \times A(i,t), \tag{7.10}$$

where $r_L(i,t)$ is the local Pareto rank values computed using the Pareto ranking scheme [146]. For those swarm leaders that are residing in cells with rank indicator $R(i,t)$ equal to one, the logical choice is to delete those swarms with higher local Pareto rank values, $r_L(i,t)$ that reside in the crowded cell and exceed the age

threshold, A_{th}. This also implies that some bird species in the mixed-species flock that have traveled together for some time may compete for the limited food source in an enclosed area

The first likelihood removes swarms that are not likely to contribute the progression towards the Pareto front, while the second likelihood acts like a diversity mechanism to encourage diversity among swarms. Two random numbers with uniform distribution between $[0,1]$ are generated to compare with l_1 and l_2. All the swarms that have either l_1 or l_2 greater than the two random numbers are removed from the swarm population. Figure 7.9 presents the pseudocode of swarm declining strategy.

7.3.8 Objective Space Compression and Expansion Strategy

The disadvantage of implementing the cell-based rank density scheme is the inability to assure the needed resolution of the resulting Pareto front, because an individual's rank value is represented by the rank value of its "home address," not by its own dominance status [140], even this is a very effective design in determining the ranking relationship during the evolutionary process. The reason is the boundaries of the objective, i.e., $F_i^{\min}$ and $F_i^{\max}$, are usually selected large enough, sometimes too large, to ensure that the entire true Pareto front is included within these boundaries. In addition to this, if the predetermined cell scales, $K_1 \times K_2 \times \ldots \times K_m$, are not chosen to be correspondingly large enough, then the cell width is too spacious compared to the true Pareto front, which may result in an inaccurate Pareto optimal set [140]. One logical choice to counter this problem is to increase the cell scales to a very large number. However, this approach will increase the computational complexity unnecessarily.

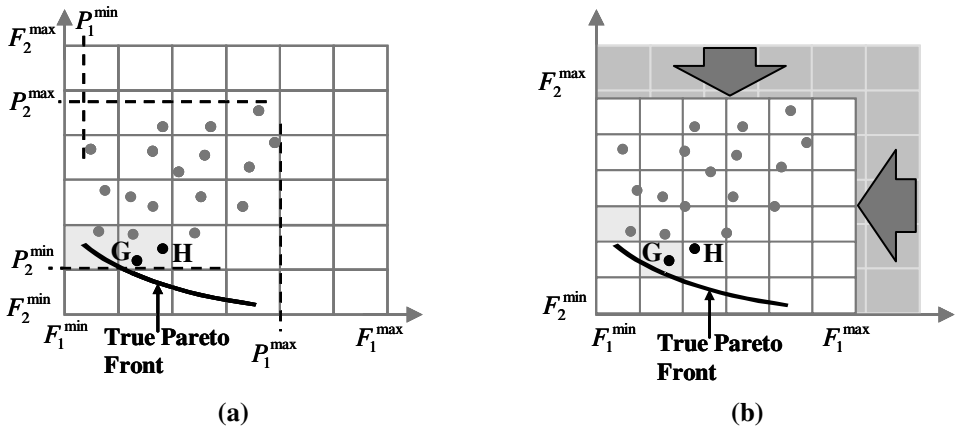


Figure 7.10 Illustration of objective space compression strategy (arrows in (b) signify the objective space is compressed).

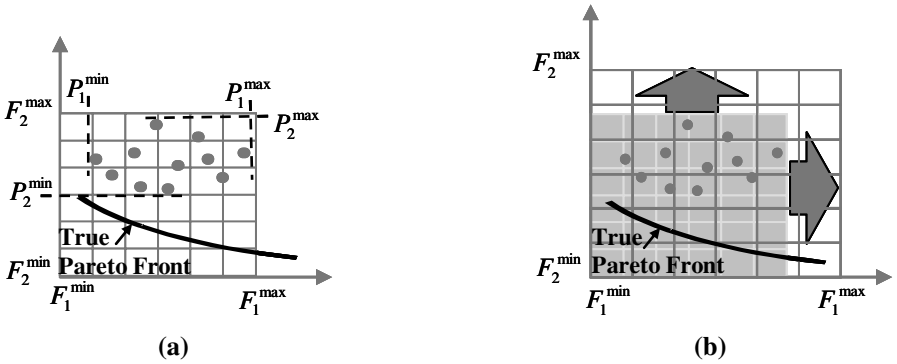


Figure 7.11 Illustration of objective space expansion strategy (arrows in (b) signify the objective space is compressed).

Inspired by DMOEA, DSMOPSO proposes the objective space compression and expansion strategy. This strategy is designed to adjust the size of the objective space based on some criteria and to ensure that swarms progressively find the true Pareto front. This design is to counter two problems: 1) at early iterations, the swarm leaders' local bests tend to quickly converge prematurely to the cells with rank value equal to 1. If this happens, the swarm population will not progress since they are stuck in those cells under the assumption that they had "found" the true Pareto front. Hence, objective space compression strategy is applied to reinforce the swarms' progression. Illustrated in Figure 7.10(a), the grey cells have rank value of 1. The swarm leaders **G** and **H** are located in a

grey cell. After the objective space is compressed, the location of grey cells is updated and only swarm leader **G** now resides in a grey cell while the rank value of swarm leader **H** is 2 (refer to Figure 7.10(b)). This will encourage swarm leader **H** to move toward the true Pareto front; 2) when the objective space compression strategy is applied several times at early iterations, there is a possibility that the objective space is overly compressed and can cause the boundaries of the objective to not cover the true Pareto front (refer to Figure 7.11(a)). For this case, the objective space expansion strategy is applied to enlarge the boundaries of the objective until the true Pareto front is approached.

The implementation of the objective space compression and expansion strategy is given below:

Compression Strategy: At iteration *t*, the lower and upper boundaries of the *i*th dimension of the objective space and current population are denoted as $F_i^{\min}$, $F_i^{\max}$, $P_i^{\min}$ and $P_i^{\max}$ [140]. The criteria to implement objective space compression strategy are:

C1-All those swarms with minimum age value greater than the age threshold, A_{th},

C2-Their maximum cell rank of the swarm leaders in swarm local best archive is equal to one, and

C3a- For upper boundary: $\left|F_i^{\max} - P_i^{\max}\right| > \delta\left(F_i^{\max} - F_i^{\min}\right)$ and/or

C3b- For lower boundary: $\left|F_i^{\min} - P_i^{\min}\right| > \delta\left(F_i^{\max} - F_i^{\min}\right)$.

The third criterion controls the sensitivity of triggering the objective space compression strategy. The ratio δ is within [0,1] and it implies that the objective space will compress

if there is at least a $(\delta \times 100)\%$ space in any dimension. An example is shown in Figure 7.10(a), i.e., distance between $F_1^{\max}$ and $P_1^{\max}$, and distance between $F_2^{\max}$ and $P_2^{\max}$. If the criteria are satisfied, then the new boundaries of the *i*th dimension of the objective space are adjusted as follow:

For upper boundary: $F_i^{\max} = \dfrac{F_i^{\max} + P_i^{\max}}{2}$ (given criteria C1, C2 and C3a are satisfied) or

(7.11a)

For lower boundary: $F_i^{\min} = \dfrac{F_i^{\min} + P_i^{\min}}{2}$ (given criteria C1, C2 and C3b are satisfied).

(7.11b)

Equations (7.11a) and (7.11b) indicate that the distance of the upper and lower boundaries are reduced by half of its original value.

Expansion Strategy: The criteria to implement objective space compression strategy are given:

E1-All those swarms with minimum age value greater than the age threshold, A_{th},

E2-Their maximum cell rank of the swarm leaders in swarm local best archive is equal to one, and

E3a- For upper boundary: $\left|F_i^{\max} - P_i^{\max}\right| \le 0.005\left(F_i^{\max} - F_i^{\min}\right)$ and/or

E3b- For lower boundary: $\left|F_i^{\min} - P_i^{\min}\right| \le 0.005\left(F_i^{\max} - F_i^{\min}\right)$.

The third criterion implies that the objective will be expanded if there is at most 0.5 percent space in any dimension, which also means that the distance between the upper boundary of objective space and the upper boundary of current population is too small as depicted in Figure 7.11(a). Note that 0.5 percent is chosen for practical purposes. If this

criterion is satisfied, then the new boundaries of the *i*th dimension of the objective space are adjusted as follows:

For upper boundary: $F_i^{\max} = F_i^{\max} + \frac{\left(F_i^{\max} + P_i^{\max}\right)}{2}$ (given criteria E1, E2 and E3a are satisfied) or (7.12a)

For lower boundary: $F_i^{\min} = F_i^{\min} - \frac{\left(F_i^{\min} + P_i^{\min}\right)}{2}$ (given criteria E1, E2 and E3b are satisfied). (7.12b)

Equations (7.12a) and (7.12b) indicate that the distance of the upper and lower boundaries are expanded by half of its original value.

After the compression or expansion strategy is performed, the "home address" of each swarm, the rank and density matrices are recalculated because they may not be the same as before. In addition, the swarm local best archive is reinitialized again. In the pseudocode presented in Figure 7.12, a parameter St is set to 0, indicating that the objective space compression and expansion strategy has been performed.

7.4 Comparative Study

Three studies are conducted. The first study aims to evaluate the performance of the DSMOPSO against the selected algorithms. Performance evaluation is determined using the standard test suits and both qualitative and quantitative metrics. In the second study, the comparison on the computational cost of the proposed algorithm and selected MOPSOs is presented in Subsection 7.4.4. Lastly, in Subsection 7.4.5, a series of experiments are performed to investigate the effect of the parameter settings on the proposed algorithm.

```
Function Objective_space_compression_expansion_strategy(δ, F^A, age, A_th,
F^pop, (F_1^min,...,F_m^min), (F_1^max,...,F_m^max), LPrank)

   /* δ = user defined parameters
   /* F^A = fitness values of particles in Archive
   /* age = age value of swarms
   /* A_th = age threshold
   /* F^pop = fitness values of swarm population (from all swarms)
   /* (F_1^min,...,F_m^min) = all lower boundaries in m dimensional objective space
   /* (F_1^max,...,F_m^max) = all upper boundaries in m dimensional objective
      space
   /* LPrank = rank of local best for swarm leaders

   Begin
      Find all swarm leaders that have age > A_th, L_age
      If LPrank of L_age are equal to 1,
         Find (P_1^min,...,P_m^min) = min[F^A, F^pop]
         Find (P_1^max,...,P_m^max) = max[F^A, F^pop]
         For i = 1 to m
               /*For upper boundary
               If abs(F_i^max − P_i^max) > 0.005,     /* compression-max
                  If abs(F_i^max − P_i^max) > δ × (F_i^max − F_i^min)
                     Compute Equation (7.11a)
                  EndIf
               Else                                  /* expansion-max
                  Compute Equations (7.12a)
               EndIf

               /*For lower boundary
               If abs(F_i^min − P_i^min) > 0.005,     /* compression-min
                  If abs(F_i^min − P_i^min) > δ × (F_i^max − F_i^min)
                     Compute Equation (7.11b)
                  EndIf
               Else                                  /* expansion-min
                     Compute Equations (7.12b)
               EndIf
         Endfor
      EndIf
      Set St = 0
      Rank_and_density_estimation()
   End
```

Figure 7.12 Pseudocode of objective space compression and expansion strategy.

7.4.1 Experimental Framework

Three MOPSOs are selected for performance and computational cost comparison. Among those three MOPSOs, cMOPSO [128] and DMOPSO are state-of-the-art multiple-swarm MOPSOs; while MOPSO [120] is selected since it has produced good performance. Each algorithm is set to perform only 30,000 fitness evaluations as suggested in [49]. The parameter configurations for all MOPSOs are summarized in Table 7.1. All of the algorithms are implemented in Matlab. All of the algorithms use a real number representation for decision variables. However, binary representation of decision variables can be easily adopted, if necessary. For each experiment, 50 independent runs were conducted to collect the statistical results. The algorithms are tested on the ZDT test suite, which is listed in Table 6.1 and we set the number of variables equal to 30, i.e., $n = 30$.

7.4.2 Selected Performance Metrics

Similar to Chapter 6, all comparisons are based on both quantitative and qualitative measures. Quantitative comparison is based on the plots of the final Pareto fronts in a given run. For quantitative comparison, two performance metrics are taken into consideration to measure the quality of algorithms with respect to dominance relations. The results are illustrated by statistical box plots. The performance metrics used here are same as given in Subsection 6.4.3: hypervolume indicator (S Metric) and additive binary epsilon indicator.

Table 7.1 Parameter configurations for existing MOPSOs and DSMOPSO.

	Parameters Settings for MOPSO
cMOPSO [128]	No. swarms = 4; Internal iterations, $stmax = 5$; Population size = 40; Archive size = Not fixed; No. of iterations = 150
MOPSO [120]	50 divisions adaptive grid; Mutation probability = 0.5; Population size = 100; Archive size = 100; No. of iterations = 300
DMOPSO	Test Function ZDT1 No. swarms = 4; Swarm size = 5; $K_i = 50, i = 1,2$; Archive size = Not fixed; $stmax = 5$; unp = 3, lnp = 1; $\alpha = \beta = 0.5$; rud = 0.7; rld = 0.02 ; $ppv = 10$; S = 0.02; and K_a = 10. Test Function ZDT2 No. swarms = 2; Swarm size = 20; $K_i = 50, i = 1,2$; Archive size = Not fixed; $stmax = 5$; unp = 3, lnp = 1; $\alpha = \beta = 0.5$; rud = 0.7; rld = 0.02 ; $ppv = 10$; S = 0.02; and K_a = 40. Test Function ZDT3 No. swarms = 3; Swarm size = 6; $K_i = 50, i = 1,2$; Archive size = Not fixed; $stmax = 5$; unp = 3, lnp = 1; $\alpha = \beta = 0.5$; rud = 0.7; rld = 0.02 ; $ppv = 10$; S = 0.02; and K_a = 15. Test Function ZDT4 No. swarms = 2; Swarm size = 20; $K_i = 50, i = 1,2$; Archive size = Not fixed; $stmax = 5$; unp = 5, lnp = 1; $\alpha = \beta = 0.5$; rud = 0.9; rld = 0.02 ; $ppv = 10$; S = 0.02; and K_a = 40. Test Function ZDT6 No. swarms = 4; Swarm size = 5; $K_i = 50, i = 1,2$; Archive size = Not fixed; $stmax = 5$; unp = 3, lnp = 1; $\alpha = \beta = 0.5$; rud = 0.9; rld = 0.02 ; $ppv = 10$; S = 0.02; and K_a = 40. Test Function DTLZ2 No. swarms = 4; Swarm size = 5; $K_i = 50, i = 1,2$; Archive size = Not fixed; $stmax = 5$; unp = 3, lnp = 1; $\alpha = \beta = 0.5$; rud = 0.7; rld = 0.02 ; $ppv = 10$; S = 0.02; and K_a = 10.
***DSMOPSO**	Swarm size = 6; $K_i = 6, i = 1,2$; $A_{th} = 3$; $ppv = 3$; $\delta = 0.1$; Archive size = 100

7.4.3 Performance Evaluation

Figure 7.13 presents the box plots of hypervolume indicator, i.e., the I_H values, found by all chosen MOPSOs for all test problems. The figure shows DSMOPSO, DMOPSO, and MOPSO achieve high I_H values for all test problems. Higher I_H values indicate the solutions found by an algorithm are able to dominate a larger region in the objective space. In Figure 7.13, the I_H values of MOPSOs are normalized for each test problem. So, the highest I_H value will equal one. Comparing the I_H values of DSMOPSO with DMOPSO and MOPSO for ZDT1, ZDT3, and DTLZ2, DSMOPSO is slightly

lower, which is also confirmed by the tested result computed using the Wilcoxon rank-sum test in Table 7.2. However, the results in Table 7.2 show DSMOPSO is better than DMOPSO for ZDT2 and ZDT6, and performs better than MOPSO for ZDT2. Only for ZDT4, did DSMOPSO share the same victory with DMOPSO and MOPSO. For the case of DTLZ2, although the I_H values of DSMOPSO are lower than the rest of MOPSOs, the difference in I_H values are very small. In addition, the results in Table 7.2 indicate that there are no significant difference between the solutions found by DSMOPSO and the rest of the MOPSOs. Overall, both box plots and results in Table 7.2 clearly show that the performance of DSMOPSO is significantly better than cMOPSO. Lastly, the low standard deviation for all test problems in Figure 7.13 shows DSMOPSO can produce reliable solutions.

The results for additive binary ε-indicator for all test functions are presented in Figure 7.14. The results for each test function are summarized into two box plots, $I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$, in which A denotes DSMOPSO, while $B_{1\text{-}3}$ corresponds to algorithms DMOPSO, MOPSO, and cMOPSO, respectively. Both $I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$ have to be taken into account to decide whether DSMOPSO dominates (or is better than) any of the selected MOPSOs. Wilcoxon rank-sum test is applied to evaluate the distribution of the $I_{\varepsilon+}$ values, and the results are presented in Table 7.3. Combine box plots of Figure 7.14 and results from Table 7.3, the following analysis is given: DSMOPSO weakly dominates DMOPSO and MOPSO for ZDT1, ZDT2, ZDT3, and ZDT6 because $I_{\varepsilon+}(A, B_{1-2}) \approx 0$ and $I_{\varepsilon+}(B_{1-2}, A) > 0$. Only for ZDT4 and DTLZ2, does DSMOPSO share the same success as DMOPSO, especially indicated in Table 7.3 where there are no significant differences between DSMOPSO and DMOPSO. Also, same level

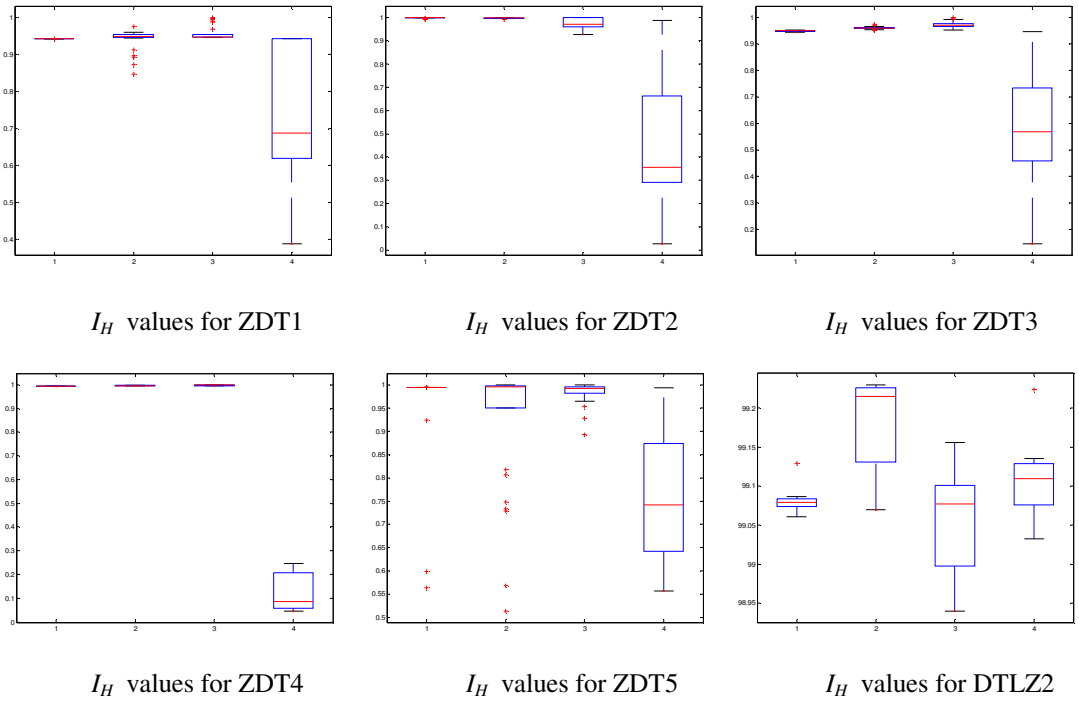


Figure 7.13 Box plot of hypervolume indicator (I_H values) for all test functions (Start from top left) by algorithms 1-4 represented (in order): DSMOPSO, DMOPSO, MOPSO, and cMOPSO.

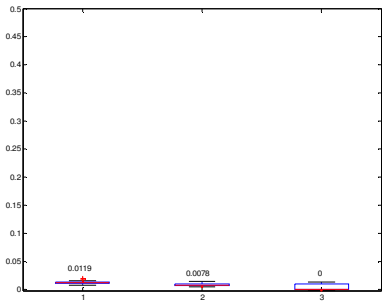
of performance is achieved by DSMOPSO and MOPSO for ZDT4 with the verification in Figure 7.14 and Table 7.3. DSMOPSO strictly dominates cMOPSO for ZDT1, ZDT2, ZDT3, ZDT4, and DTLZ2 because the $I_{\varepsilon+}(A, B_3) \leq 0$ and $I_{\varepsilon+}(B_3, A) > 0$, except for ZDT6 in which DSMOPSO weakly dominates cMOPSO. Overall, DSMOPSO shows a better performance compared to the selected MOPSOs.

For qualitative comparison, the resulting Pareto fronts generated by the selected MOPSOs from a single run given the same initial population are presented in Figures 7.15-7.20. The resulted Pareto fronts obtained by DSMOPSO are comparatively well expanded and near optimal Pareto fronts. DMOPSO and MOPSO are able to find satisfactory Pareto fronts. However, cMOPSO either has difficulty converging to the true Pareto front for ZDT1, ZDT4, and ZDT6, or it partially obtains part of the solutions of

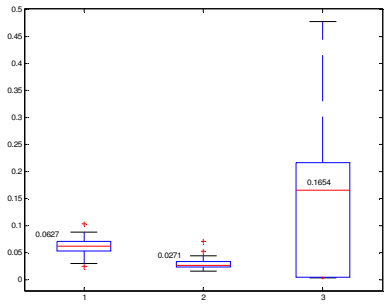
the optimal Pareto front. Only for DTLZ2, the resulting Pareto front of DSMOPSO is slightly not better than DMOPSO's. In general, both quantitative and qualitative results conclude that the performance of DSMOPSO is highly competitive with respect to the selected state-of-the-art MOPSOs for a selected set of test functions.

Table 7.2 The distribution of I_H values tested using Wilcoxon rank-sum test. The table presents the z values and p-values, i.e., presented in the brackets as (z value, p-value), with respect to the alternative hypothesis (i.e., p-value < α=0.05) for each pair of DMOPSO and a selected MOPSO. Note that the distribution of DMOPSO is significantly difference or better than those selected MOPSO unless stated.

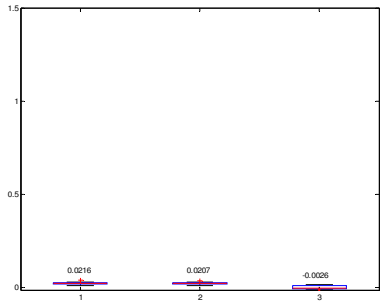
Test Functions	I_H (DSMOPSO) AND		
	I_H (DMOPSO)	I_H (MOPSO)	I_H (cMOPSO)
ZDT1	(-3.6283, 2.9E-04)	(-6.0537, 1.9E-09)	(3.9776, 6.9E-05)
ZDT2	(2.8620, 4.0E-02)	(2.8620, 4.2E-02)	(4.6455,3.4E-06)
ZDT3	(-6.5569, 5.5E-11)	(-6.6318, 3.3E-11)	(6.5421, 6.1E-11)
ZDT4	(3.8575,>0.05) *no difference*	(0.290,>0.05) *no difference*	(4.6455, 3.4E-06)
ZDT6	(-2.7837, 5.0E-03)	(0.8087,>0.05) *no difference*	(5.5674, 2.6E-06)
DTLZ2	(0.5127,>0.05) *no difference*	(0.5681,>0.05) *no difference*	(0.5589,>0.05) *no difference*



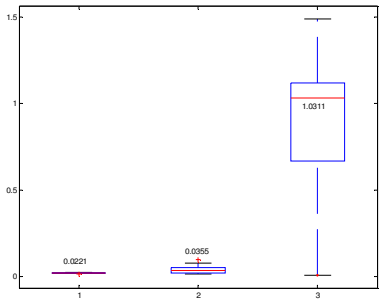
$I_{\varepsilon+}(A, B_{1-3})$ for ZDT1



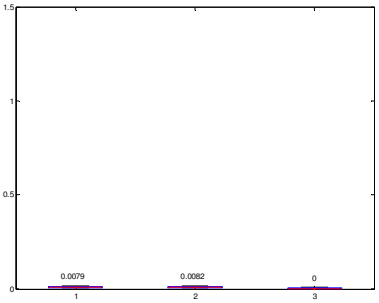
$I_{\varepsilon+}(B_{1-3}, A)$ for ZDT1



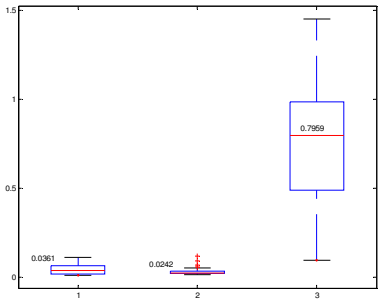
$I_{\varepsilon+}(A, B_{1-3})$ for ZDT2



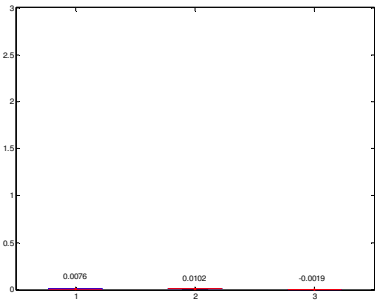
$I_{\varepsilon+}(B_{1-3}, A)$ for ZDT2



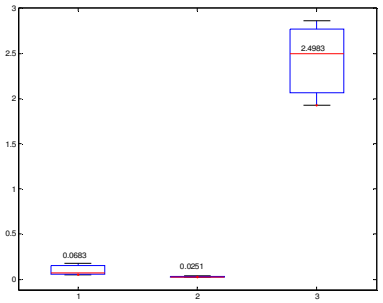
$I_{\varepsilon+}(A, B_{1-3})$ for ZDT3



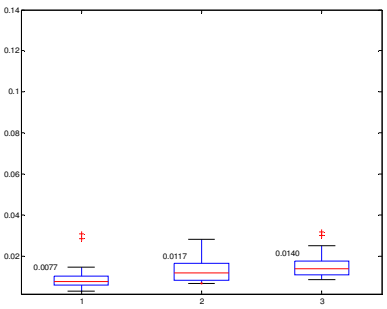
$I_{\varepsilon+}(B_{1-3}, A)$ for ZDT3



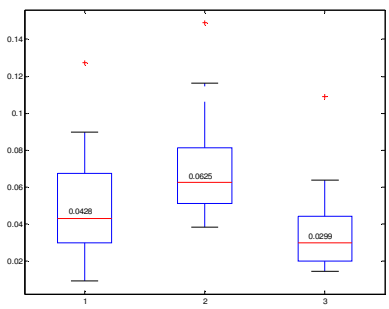
$I_{\varepsilon+}(A, B_{1-3})$ for ZDT4



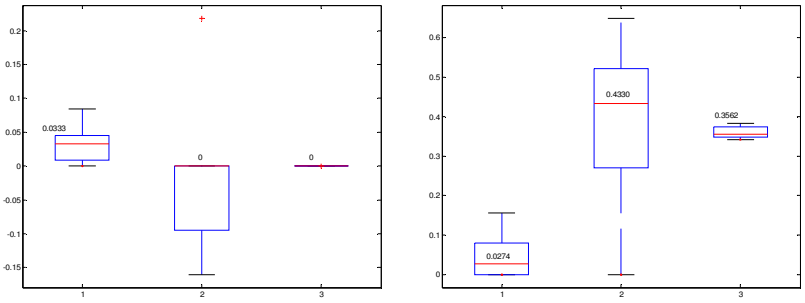
$I_{\varepsilon+}(B_{1-3}, A)$ for ZDT4



$I_{\varepsilon+}(A, B_{1-3})$ for ZDT6



$I_{\varepsilon+}(B_{1-3}, A)$ for ZDT6



$I_{\varepsilon+}(A, B_{1-3})$ for DTLZ2 $I_{\varepsilon+}(B_{1-3}, A)$ for DTLZ2

Figure 7.14 Box plot based upon multiplicative binary epsilon indicator ($I_{\varepsilon+}$ values) all test functions (Start from top left) (algorithm *A* refer to DSMOPSO; algorithms 1-3 are referred to as DMOPSO, MOPSO, and cMOPSO, respectively).

Table 7.3 The distribution of $I_{\varepsilon+}$ values tested using Wilcoxon rank-sum test. The table presents the z values and p-values with respect to the alternative hypothesis (i.e., p-value < α=0.05) for each pair of DMOPSO and a selected MOPSO. In each cell, both values are presented in a bracket like this: (z value, p-value). For simplicity in naming, DSMOPSO is represented by *A*, and algorithms *B1* to *B3* are referred to as DMOPSO, MOPSO, and cMOPSO, respectively. The distribution of DMOPSO is significantly difference or better than those selected MOPSO unless stated.

Test Functions	$I_{\varepsilon+}$ *(A,B1)* and $I_{\varepsilon+}$ *(B1,A)*	$I_{\varepsilon+}$ *(A,B2)* and $I_{\varepsilon+}$ *(B2,A)*	$I_{\varepsilon+}$ *(A,B3)* and $I_{\varepsilon+}$ *(B3,A)*
ZDT1	(-6.0537, 1.5E-04)	(-6.0537, 1.4E-04)	(-4.3553, 1.3E-05)
ZDT2	(0,>0.05) *no difference*	(-2.7790, 5.5E-03)	(-4.4382, 9.1E-06)
ZDT3	(-6.2464, 4.2E-04)	(-6.5717, 5.0E-06)	(-6.6979, 2.1E-06)
ZDT4	(-0.001,>0.05) *no difference*	(-0.015,>0.05) *no difference*	(-4.4055, 3.4E-04)
ZDT6	(-6.0028, 1.9E-06)	(-6.5315, 6.5E-06)	(-5.3496, 8.8E-06)
DTLZ2	(0,>0.05) *no difference*	(-5.2337, 3.1E-04)	(-5.7395, 1.6E-04)

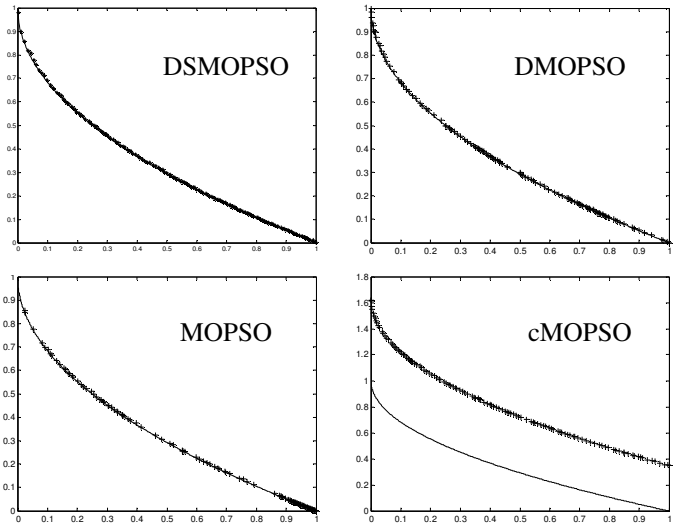


Figure 7.15 Pareto fronts produced by (a) DSMOPSO, (b) DMOPSO, (c) MOPSO, and (d) cMOPSO for ZDT1. The continuous line depicts the true Pareto front.

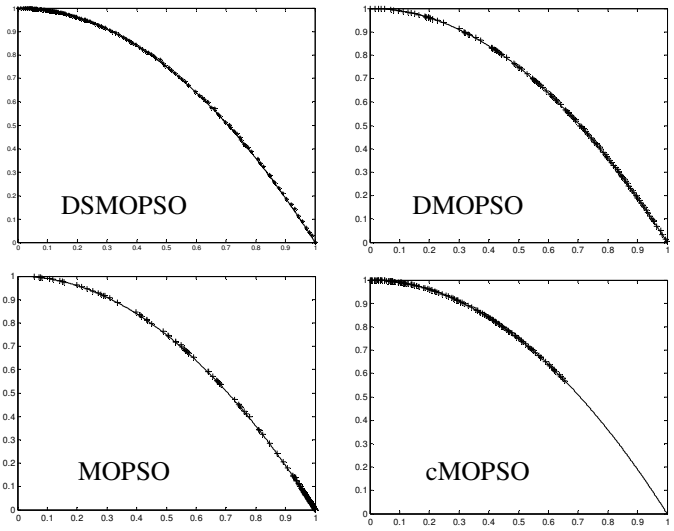


Figure 7.16 Pareto fronts produced by (a) DSMOPSO, (b) DMOPSO, (c) MOPSO, and (d) cMOPSO for ZDT2.

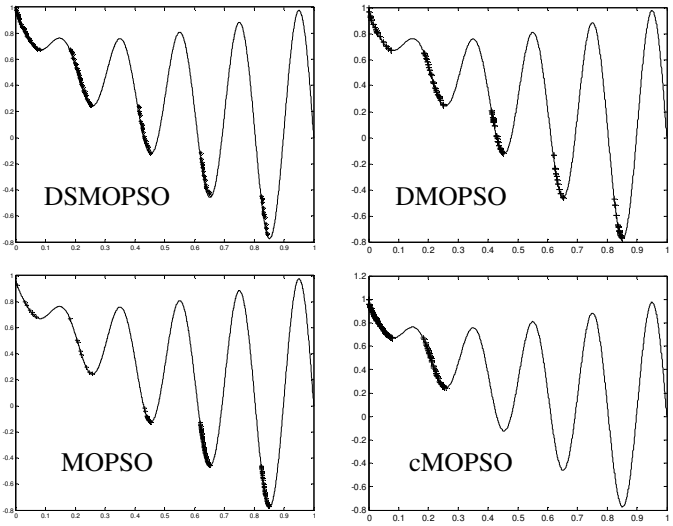


Figure 7.17 Pareto fronts produced by (a) DSMOPSO, (b) DMOPSO, (c) MOPSO, and (d) cMOPSO for ZDT3.

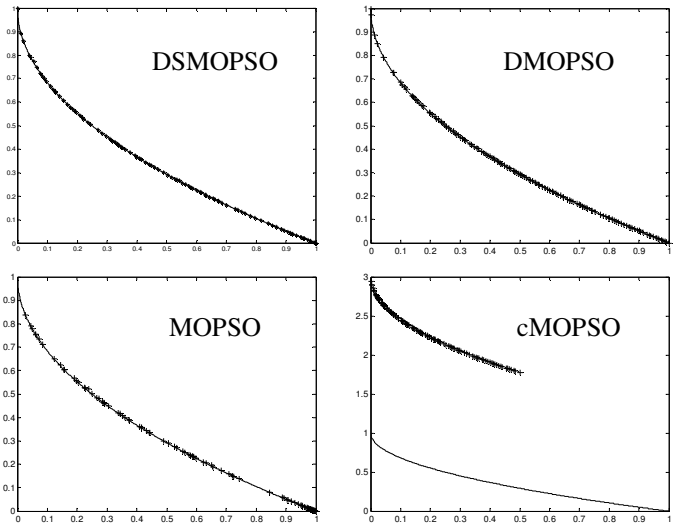


Figure 7.18 Pareto fronts produced by (a) DSMOPSO, (b) DMOPSO, (c) OMOPSO, and MOPSO for ZDT4.

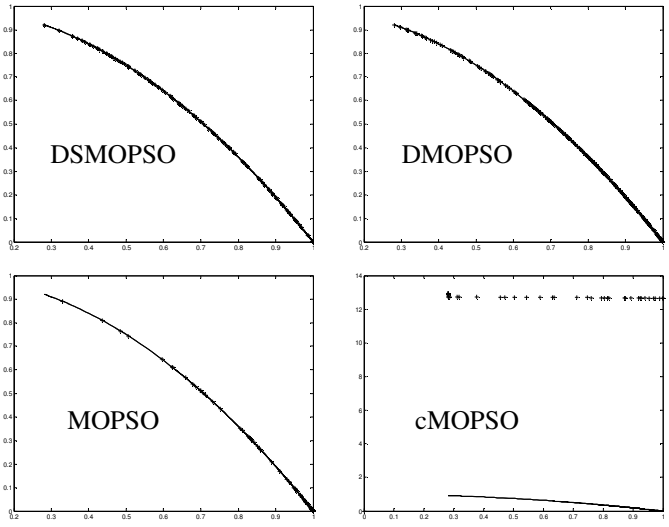


Figure 7.19 Pareto fronts produced by (a) DSMOPSO, (b) DMOPSO, (c) MOPSO, and (d) cMOPSO for ZDT6.

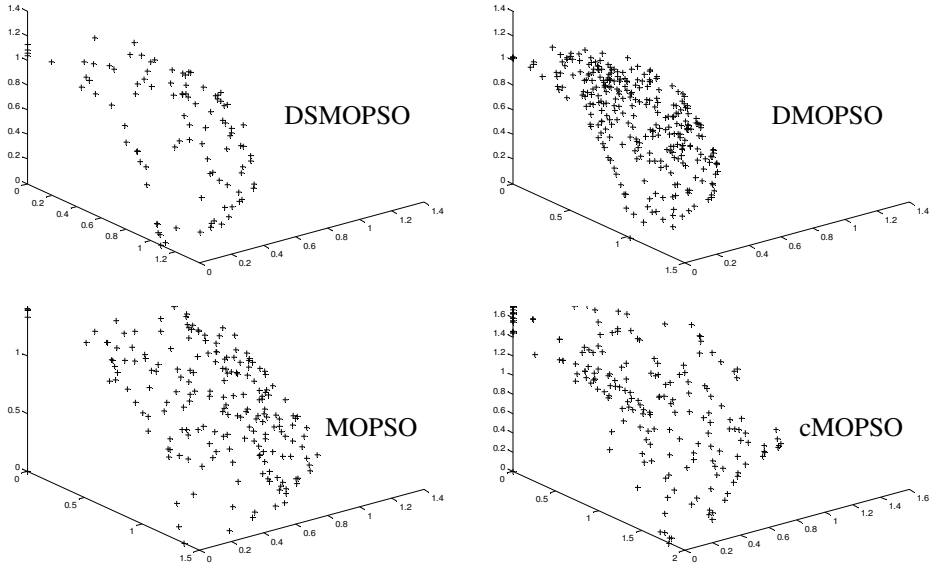


Figure 7.20 Pareto fronts produced by (a) DSMOPSO, (b) DMOPSO, (c) MOPSO, and (d) cMOPSO for DTLZ2.

7.4.4 Comparison in Number of Fitness Evaluation

Table 7.4 Average number of evaluations computed for the test problems to achieve GD =0.001.

	GD =0.001	*DSMOPSO	DMOPSO	MOPSO	cMOPSO
ZDT1	Average No. Evaluations	**20,252**	464.8	3,267	3,264
ZDT2	Average No. Evaluations	**10,003**	998.9	2,900	500,000
ZDT3	Average No. Evaluations	**25,886**	500,000	500,000	500,000
ZDT4	Average No. Evaluations	**851**	12,781.6	22,149	500,000
ZDT6	Average No. Evaluations	**1,490**	2,584	5,340	500,000
DTLZ2	Average No. Evaluations	83,568	**18234**	500000	59840

In this experiment, the computational cost of DSMOPSO is compared with the selected MOPSOs. In [148], an algorithm called ParEGO has shown efficient in solving nine relatively low-dimensional, real-valued test functions using a very low number of function evaluations. A targeted generational distance [45], *GD* value, is set to 0.001 for all of the test problems. A limit of 500,000 evaluations is used as stopping criteria. Each MOPSO performs 50 independent runs and the total number of evaluations to reach the targeted *GD* value is recorded for each run. All parameter configurations for MOPSO are shown in Table 7.1. The average number of evaluations is recorded for each MOPSO and is presented in Table 7.4. Overall, MOPSO designs coupled with dynamic population concepts show saving in computational cost compared to the standard MOPSOs. DMOPSO demands less computational cost for ZDT1, ZDT2, and DTLZ2 than DSMOPSO. However, DSMOPSO is able to achieve less computational cost for more challenging problems in ZDT3, ZDT4, and ZDT6 compared to the rest of the MOPSOs. From observations, it seems that DSMOPSO needs less computational cost for test problems with disconnected Pareto front and with multiple local optima. For connected

Pareto fronts, the results seem to indicate that DSMOPSO requires more evaluations than DMOPSO, MOPSO, and cMOPSO, most likely due to the process involved in adapting the number of swarms needed.

7.4.5 Sensitivity Analysis

Analysis of how sensitive DSMOPSO is with respect to the setting of the design parameters is presented herein. The same six test functions are used and hypervolume indicator is adopted. A limit of 30,000 evaluations is used as stopping criterion, and 30 independent runs are performed. Five experiments are conducted. In every experiment one parameter is varied while the rest of the parameter configurations remain the same as shown in Table 7.1. Box plots of hypervolume indicator for all test functions are showed for the five experiments.

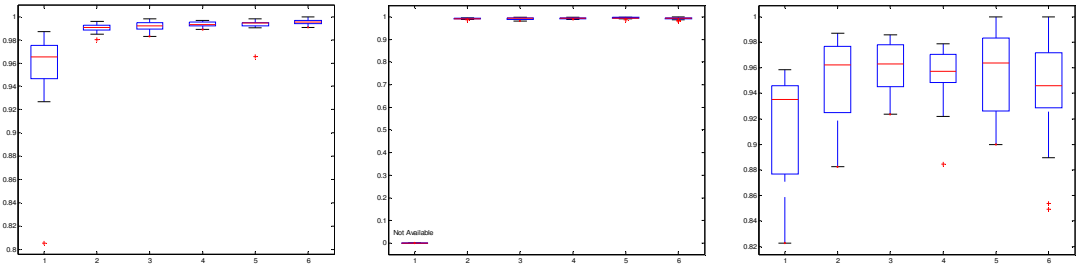
1) *Effect of Varying the Swarm Size*: Figure 7.21 presents the box plots of hypervolume indicator for swarm size varied from 2 to 20. By observation, the I_H values obtained from all test functions are relatively high with small variations except for ZDT2, no result is obtained for swarm size equal to 2 because it failed to obtain the Pareto front. In overall, the figure shows that swarm size between 4 to 6 yield high I_H values for all test functions except for ZDT4 and DTLZ2, the I_H values is slightly lesser than the I_H value obtained by swarm size equal to 2 (the difference is about 0.01 to 0.05). Hence, we recommend setting the swarm size between 4 and 6.
2) *Effect of Varying the Grid Scale*: Figure 7.22 shows the impact of the grid scale on the performance of DSMOPSO. Looking at the figure, the results yielded indicate that the performance of DSMOPSO is not affected. However, if compare

the results among the test problems, ZDT3, ZDT6, and DTLZ2 do not show any patterns associated with different grid scales; while ZDT1, ZDT2, and ZDT4 shows good results for different grid scales in terms of high I_H values and very low standard deviations. Hence, based on this observation, the algorithm is believed to be insensitive to the grid scale.

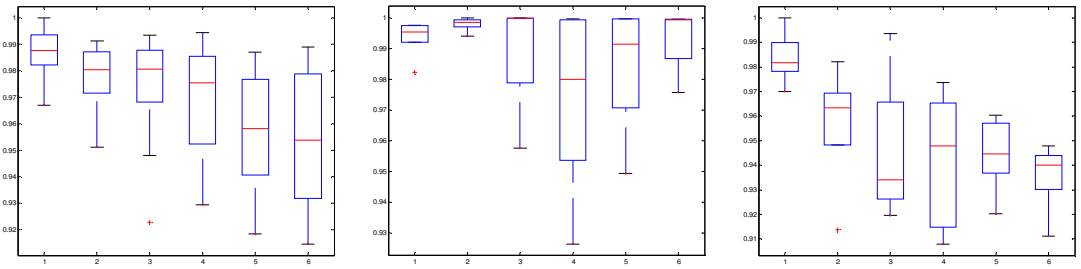
3) *Effect of Varying the Population Size Per Cell*: The population size per cell varies from 3 to 25. Figure 7.23 shows DSMOPSO is able to achieve high I_H values with small standard deviation (i.e. the maximum deviation is about 0.1) for all test functions regardless of any population size per cell. Among those values, it is shown that I_H values are consistently highest for all test functions when the population size per cell equals 3. Based on the results, we recommend using population size per cell equal to 3.

4) *Effect of Varying the δ Parameter*: Refer to Figure 7.24, the I_H values remains very close to 1 as δ increases for each test function except for ZDT3 and DTLZ2, which have relatively lower I_H values. The results also show the standard deviations are generally low in overall, indicating δ parameter does not significantly influence the reliability of the solutions. Based on the results, any settings for δ will work for those test functions.

5) *Effect of Varying the Age Threshold A_{th}*: The experimental results of varying the age threshold between 3 and 25 are presented in Figure 7.25. Again, the result shows any setting for age threshold is able to deliver good performance, i.e., high I_H values varied between 0.94 to 1. Hence, any age threshold settings are allowed.

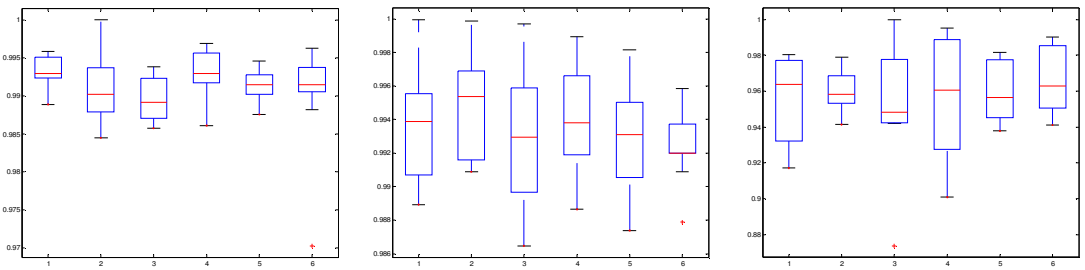


I_H values for ZDT1 I_H values for ZDT2 I_H values for ZDT3

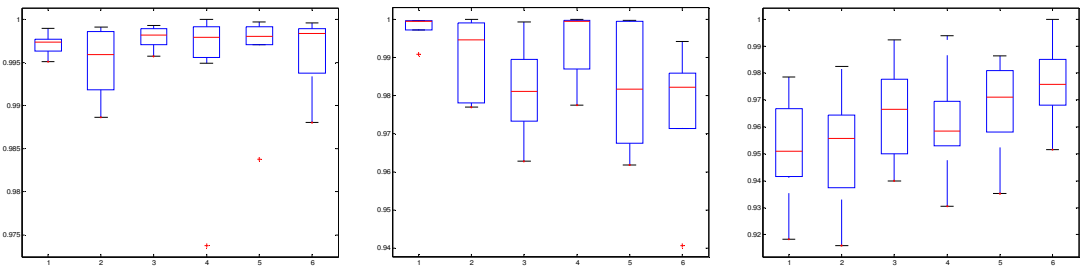


I_H values for ZDT4 I_H values for ZDT5 I_H values for DTLZ2

Figure 7.21 Box plot of hypervolume indicator (I_H values) for experiment with varying the swarm size. Note that 1-6 on x-axis represented (in order): swarm size of 2, 4, 6, 8, 12, and 20.

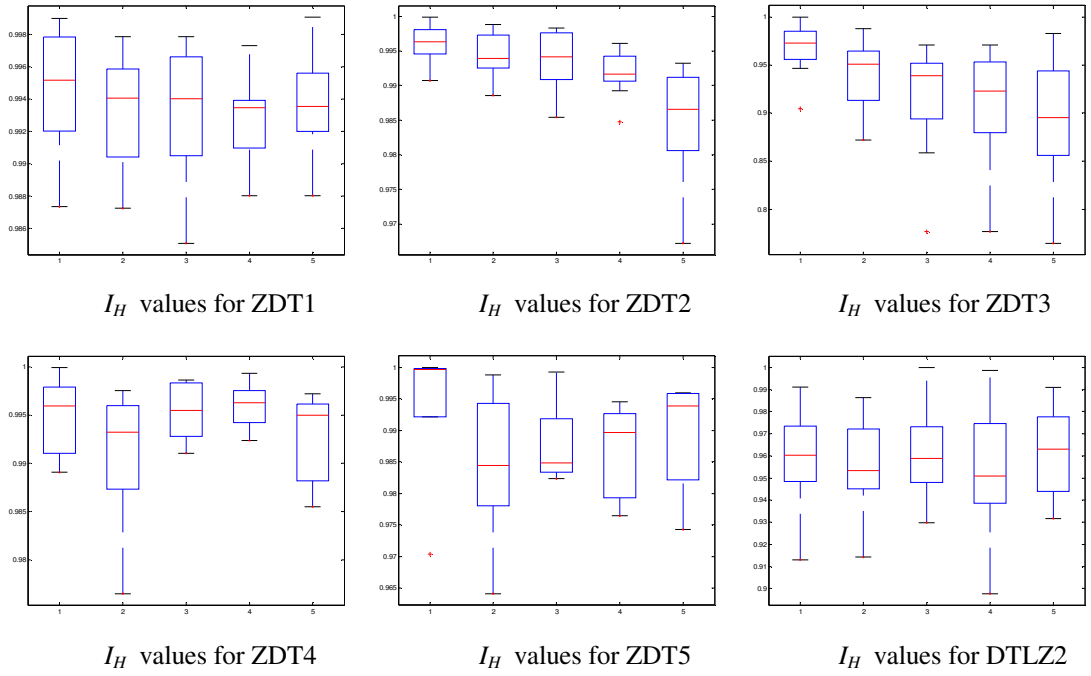


I_H values for ZDT1 I_H values for ZDT2 I_H values for ZDT3



I_H values for ZDT4 I_H values for ZDT5 I_H values for DTLZ2

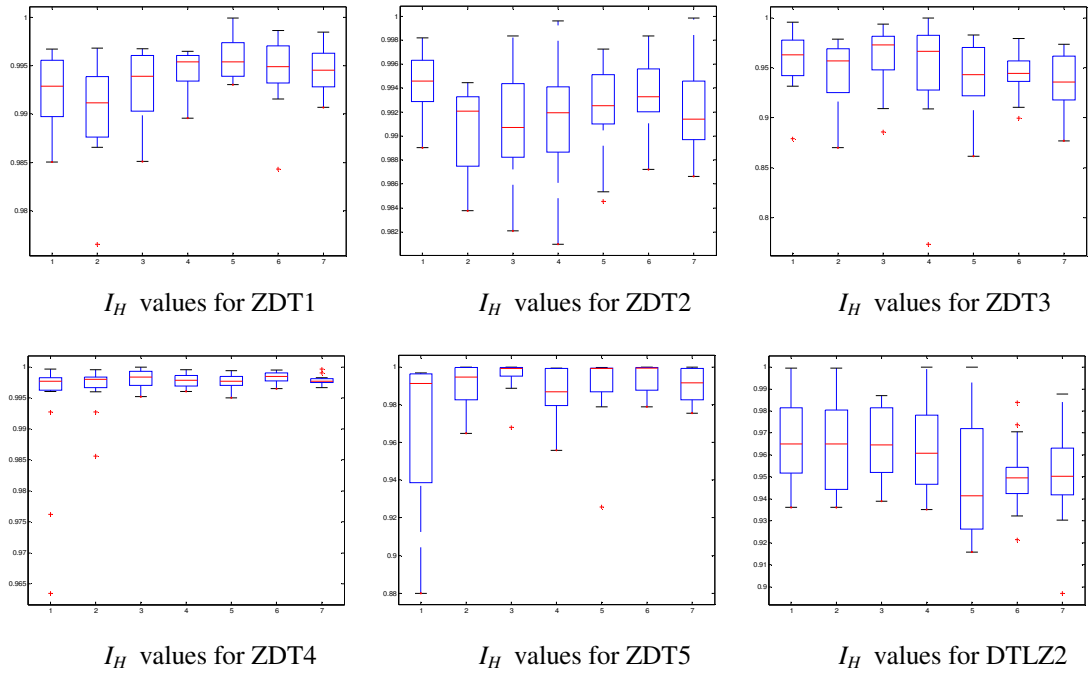
Figure 7.22 Box plot of hypervolume indicator (I_H values) for experiment with varying the grid scale (K_i). Note that 1-6 on x-axis represented (in order): K_i equals to 4, 5, 6, 7, 10, and 15.



I_H values for ZDT1 I_H values for ZDT2 I_H values for ZDT3

I_H values for ZDT4 I_H values for ZDT5 I_H values for DTLZ2

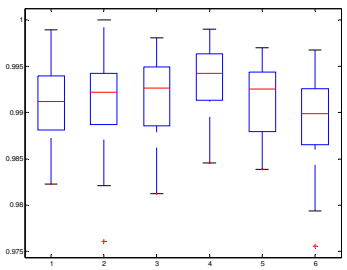
Figure 7.23 Box plot of hypervolume indicator (I_H values) for experiment with varying the population size per cell (*ppv*). Note that 1-5 on x-axis represented (in order): *ppv* equal to 3, 5, 8, 12, and 25.



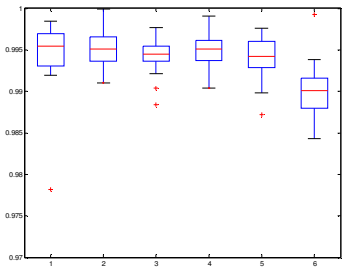
I_H values for ZDT1 I_H values for ZDT2 I_H values for ZDT3

I_H values for ZDT4 I_H values for ZDT5 I_H values for DTLZ2

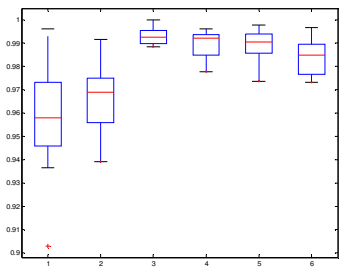
Figure 7.24 Box plot of hypervolume indicator (I_H values) for experiment with varying the δ parameter. Note that 1-7 on x-axis represented (in order): δ is equal to 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, and 0.9.



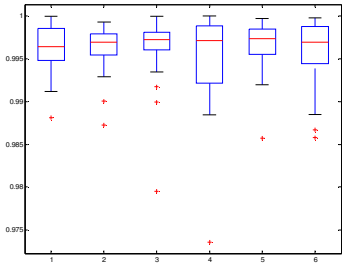
I_H values for ZDT1



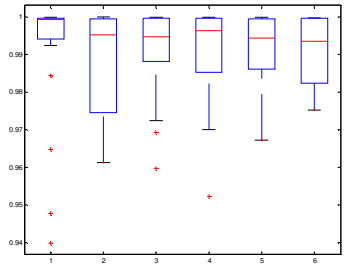
I_H values for ZDT2



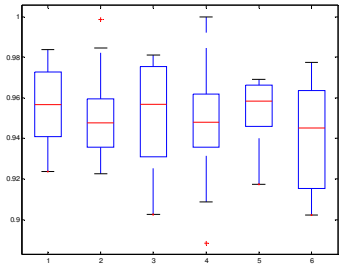
I_H values for ZDT3



I_H values for ZDT4



I_H values for ZDT5



I_H values for DTLZ2

Figure 7.25 Box plot of hypervolume indicator (I_H values) for experiment with varying the age threshold (A_{th}). Note that 1-6 on x-axis represented (in order): A_{th} is equal to 3, 4, 5, 6, 10, and 25.

CHAPTER 8

PROPOSED PSO AND MOPSO FOR CONSTRAINED OPTIMIZATION

Chapters 6 and 7 incorporated the dynamic population concept into swarm population to solve for unconstrained MOPs. However, in real world applications, most of the optimization problems involve constraints. To optimize the constrained problems is challenging since the optimum solution(s) must be feasible or else the solutions found will be useless. In this chapter, the goal is to design a constrained MOPSO to solve for constrained multiobjective optimization problems. In order to achieve this goal, a constrained PSO is designed to solve for constrained optimization problems, as a basic step. Then, the proposed constrained PSO is extended into a constrained MOPSO. Details on the two proposed algorithms are elaborated and experiments are conducted to evaluate the performance of the proposed algorithms.

8.1 Introduction

In real world applications, most optimization problems are subjected to different types of constraints. These problems are known as the constrained optimization problems (COPs) or if more than one objective functions are involved, it is called constrained multiobjective optimization problems (CMOPs). Although publication records have proven that evolutionary algorithms (EAs) are effective tools in solving different types of optimization problems, EAs in their original design are unable to solve constrained

optimization problems effectively. Hence, in the past decade, many researchers have developed a variety of constraint handling techniques to counter this deficiency. These techniques are mainly incorporated within evolutionary algorithm designs (EAs), particularly genetic algorithm, to solve COPs [155,156] and CMOPs [157-166]. Recently, EA based on multiobjective optimization formulation for COPs gains much attention since it requires neither penalty factors that need heuristic tuning nor the need to balance the right proportion of feasible and infeasible solutions in the population via selection criteria [157,164].

PSO has advantages over evolutionary algorithms, which are its simplicity, easy implementation, and rapid convergence capability. So, the number of researches on PSO design to solve for unconstrained SOPs and MOPS has gradually gain momentum in the past few years. However, there are relatively fewer works that apply PSO for COPs [167-175]. Similar to EAs, the original PSO design also lacks a mechanism to handle constraints in order to solve COPs. Most of the proposed PSO designs adopted the popular constraint handling techniques that are build for EAs. Evidence shows in recent publications on constraint handling with PSO including penalty methods [167], selection criteria based on feasible and constraint violation [168-170], lexicographic order [171], and multiobjective constraint handling method [172-175], to name a few.

Nevertheless, many real world problems are often multiobjective in nature. The ultimate goal is to develop multiobjective particle swarm optimization algorithms (MOPSOs) that effectively solve CMOPs. In addition to this perspective, the recent successes of MOPSOs in solving unconstrained MOPs have further motivated us to

design a constrained MOPSO to solve CMOPs. To achieve this goal, the constrained PSO is proposed to deal with COPs as a basic step towards the design of constrained MOPSO.

In this design, we integrate the multiobjective optimization techniques in PSO for constraint handling. Three main design elements are incorporated to the proposed PSO: 1) the updating personal best archive procedure has two separate conditions. The first condition targets the infeasible personal best in the archive. For this case, a simple formula based on the particles' Pareto rank and their constraint violation is designed to update the infeasible personal best. The second condition aims for the feasible personal best and the updating rule is based on Pareto ranks or fitness. 2) Feasible and infeasible global best archives are used to store both feasible and infeasible nondominated solutions. The purpose is to make use of these solutions to guide the particles to feasibility and then towards the global optimum solution. Procedure to maintain these archives are discussed. 3) The acceleration constants in the PSO equation are controlled by a feasibility ratio and the constraint violations of the personal best and global best archives' members. This will encourage the particles to search for feasible regions and the global optimum solution. These design elements are adopted to solve for CMOPs. Different mutation procedures are incorporated in both proposed constrained PSO and MOPSO.

8.2 Related Works

In this section, relevant works of PSO adopting multiobjective optimization formulation to solve for COPs are reviewed first. Then, a brief review on the various constraint handling techniques designed for multiobjective evolutionary algorithms

(MOEAs) is presented, as there exist no prominent constrained MOPSO design in literature, to our best knowledge.

Multiobjective constraint handling formulation (also called multiobjective optimization techniques to handling constraints) are based on multiobjective optimization concepts. The idea is to convert the constraints into one or more unconstrained objective functions and handle them via Pareto dominance relation. From the comprehensive survey conducted by Mezura-Montes and Coello Coello [176], they grouped the techniques available into two main categories.

For the first category, a COP is converted into an unconstrained bi-objective optimization problem where the first objective is the original objective function and the second objective is the sum of the constraint violations. The following works fall into this category: Lu and Chen [172] proposed a novel constraint handling technique, called dynamic objective method (DOM), which can be easily incorporated into a variety of PSO algorithms. DOM does not apply Pareto dominance relation, but incorporates a threshold to control when to start the process of optimizing the objective function from the process of minimizing the sum of constraint violations. The threshold is used to update personal and global bests. In addition, the same authors also proposed a restricted velocity particle swarm optimization (RVPSO), in which the PSO equation is modified to incorporate the impact of feasible region on the velocity equation. Experiment results show DOM is efficient in handling constraints and the combined algorithm (DOM+RVPSO) shows competitive in solving COPs. However, how sensitive for the choice of thresholds to impact the performance over different test problems is not discussed. Li *et al*. [173] incorporated goal oriented programming concept to guide the

search towards the global optimum (feasible) solution. A feasible tolerance parameter is defined to determine how minimum does the constraint violations should allow. Selection rules based on Pareto dominance relationship and comparison of constraint violations are proposed to update personal and neighborhood bests. Perturbation with minor probability is applied for diversity maintenance. Simulation results show competitive performance but require prior knowledge of true solution and user settings of feasible tolerance parameter.

For the second category, a COP is converted into an unconstrained multiobjective optimization problem (MOP), i.e., original objective function and each constraint is treated as a separate objective function. Hence, we will have p+1 unconstrained objective functions and the parameter p refers to the total number of constraints (see Equations (2.2) and (2.3)). Liang and Suganathan [174] proposed a dynamic particle multi-swarm optimization (DMS-PSO). In their design, a sub-swarm or several subswarms are assigned to optimize one objective selected from the objective functions and constrained functions. The assignment of these subswarms changes adaptively and the assignment depends on the complexity of the constraints, e.g. more number of subswarms will be assigned to work on difficult constraints. In addition, the authors applied a local search with sequential quadratic programming (SQP) on a set of five randomly chosen particles' personal best (pbest). Twenty-four benchmark functions are tested and the algorithm is able to obtain the global (feasible) solution efficiently. The drawback is that the user-defined parameters need to be tuned heuristically. In [175], the COP is converted into p+1 unconstrained objective functions and optimizes these functions as MOPs. The author exploited the information of the "worst" solutions by adding a global worst term

on the original velocity equation. The idea is to inform the particles to slightly move away from the center of the least feasible solutions found so far and head towards the direction of global best. The initial experimental results show this approach can produce good results for certain benchmark functions. The drawback is the 'global worst' is defined in terms of the solution with the worst constraint violation and the author plans to incorporate the worst objective function values in the future.

During the past decade, researchers are interested in the design of MOEAs for unconstrained MOPs and only a handful of MOEA designs is specifically for handling constraints.

In [158], the constraint handling is incorporated within a decision making framework based on goals and priority, in which the constraints are given higher priority than the objective functions during the search process. Hence, emphasis is given to searching for feasible solutions first then to only searching for global solution next.

Coello Coello and Christiansen [159] developed two new MOEAs based on the notion of min-max optimum to solve CMOPs. These MOEAs only optimize feasible solutions since only feasible solutions will survive to the next generation and the crossover and mutation operators are designed in such only to produce feasible solutions. However, their algorithms may face difficulty in producing a set of feasible solutions at the initialization step and require large computational time if the feasible region is small.

In [160], Multiobjective Evolutionary Strategy (MOBES) is proposed. This design includes dividing the infeasible individuals into different classes according to their "nearness" to the feasible region, ranking the infeasible individuals based on the class, computing fitness values according to proportion of feasible and/or infeasible individuals

in the population, and incorporating a mechanism to maintain a set of feasible Pareto optimum solutions in every generation. Experimental results on some benchmark functions indicate MOBES is efficient in handling constraints in CMOPs.

Deb *et al.* [31] introduced a constrained domination principle to handle constraint in their NSGA-II. An individual *i* is said to constrained-dominate an individual *j* 1) if individual *i* is feasible and individual *j* is infeasible; 2) if both individuals *i* and *j* are infeasible and individual *i* has smaller constraint violations; and 3) if both individuals *i* and *j* are feasible and individual *i* dominates individual *j*. All feasible individuals are ranked via usual Pareto dominance relationship while all infeasible individuals are ranked according to their amount of constraint violation. This constraint handling technique is also adopted in micorgenetic algorithm (microGA) by Coello Coello and Pulido [177], and a MOPSO that is proposed by Coello Coello *et al.* [120].

Another novel MOEAs called Evolutionary Algorithm of Non-dominated Sorting with Radial Slots (ENORA) is proposed by Jimenéz *et al.* [161]. Their proposed constraint handling technique involves allowing feasible solutions to evolve towards optimality while infeasible solutions to evolve towards feasibility using the min-max formulation. The diversity mechanism divides the decision space into a set of radial slots along with the successive populations generated. Ray and Won [162] also employ standard min-max formulation for constraint handling and divides the objective space into a predefined number of radial slots where the solutions will compete with members in the same slot for existence.

Harada *et al.* proposed Pareto Descent Repair Operator (PDR) to repair the infeasible solution by searching for feasible solution closest to the infeasible solutions in

the constraint function space [163]. Their idea is to reduce all violated constraints simultaneously.

Geng *et al.* [164] proposed a new constraint handling strategy to address the deficiency of Deb's constrained domination principle in NSGA-II [31]. In their proposal, infeasible elitists are kept to act as a bridge connecting any isolated feasible regions during the evolution process. In addition, they adopted the stochastic ranking [155] to obtain a good balance in selecting between the feasible and infeasible elitists. Their idea is applied to NSGA-II and compared the performance with the original NSGA-II on six benchmark CMOPs. Their proposed strategy shows significant improvement in terms of distributions and quality of the Pareto fronts on benchmark problems with disconnected feasible regions.

In [165], the authors proposed two algorithms to solve CMOPs. For the first algorithm, Objective Exchange Genetic Algorithm for Design Optimization (OEGADO), each single-objective GA optimizes one objective or constraint function with independent population. Since there are several objectives and constraint functions, several GAs will run concurrently. At certain generations, the solutions found by all GAs will exchange information with each other. On the contrary, for the second algorithm, Objective Switching Genetic Algorithm for Design Optimization (OSGADO), a single-objective GA optimizes several objective functions in a sequential order, in which, one objective is optimized for a certain number of fitness evaluations, then switch to the next objective to optimize for a certain number of fitness evaluations, and this continues until the fitness evaluation for the last objective is completed. The process is repeated starting from the first objective to the last objective until the maximum number of fitness evaluations is

reached. Based on the experimental study, OEGADO shows better and consistent performance.

Recently, Woldesenbet *et al.* [166] proposed an adaptive penalty function [27] that exploits the information of the solutions to guide the solutions towards the feasible region and search for optimum solution. They proposed a modified objective function value that consists of two key components: distance measure and adaptive penalty. Then, the dominance relation of the solutions is checked using the modified objective function values. Their idea is incorporated in NSGA-II, but can be easily extended to any MOEAs. Simulation results show the superiority of their proposed algorithm in performance compared to the selected MOEAs.

8.3 Proposed Approach

Based on what we learn, the proposed approaches involve adopting an existing constraint handling technique and modify the mechanism in the original PSO to simultaneously handle constraints as well as optimize the objective functions. For the following subsections, the design elements of the proposed constrained PSO are discussed first, and then the designs are extended into a MOPSO.

8.3.1 Transform a COP into an Unconstrained Bi-objective Optimization Problem

In Chapter 2, the general form of the multiobjective optimization problem (MOPs) is defined by Equations (2.1)-(2.4). From these equations, by setting k equal to 1 (since there is only one objective function), we defined a general single constrained optimization problem (COP). To transform a COP into unconstrained bi-objective

optimization problem, both the inequality and equality constraints (i.e., $g_j(\mathbf{x})$ and $h_j(\mathbf{x})$ respectively) are treated as one objective and the other objective is the original objective function $F(\mathbf{x})$. Hence, Equations (2.1), (2.2) and (2.3) are transformed into the following general form of an unconstrained bi-objective optimization problem:

$$\text{Minimize}\,\mathbf{F}(\mathbf{x}) = \left[cv(\mathbf{x}), F(\mathbf{x}) \right], \tag{8.1}$$

where $cv(\mathbf{x})$ is the scalar constraint violation of a decision vector $\mathbf{x}$ (or particle) and it is mathematically formulated as below:

$$cv(\mathbf{x}) = \frac{1}{p} \sum_{j=1}^{p} \frac{cv_j(\mathbf{x})}{cv_{\max}^{j}} \tag{8.2}$$

$$\text{where} \quad cv_j(\mathbf{x}) = \begin{cases} \max\left(0, g_j(\mathbf{x})\right), & j = 1, \ldots, m \\ \max\left(0, \left|h_j(\mathbf{x})\right| - \delta\right), & j = m+1, \ldots, p \end{cases}, \tag{8.3a}$$

$$cv_{\max}^{j} = \max_{\mathbf{x} \in \boldsymbol{S}} cv_j(\mathbf{x}). \tag{8.3b}$$

Parameter δ is the tolerance allowed for equality constraints, usually δ is set to 0.001 or 0.0001. If a particle or solution ($\mathbf{x}$) satisfies the *j*th constraints, then $cv_j(\mathbf{x})$ is set to zero, otherwise it is greater than zero. Finally, $c_{\max}^{j}$ represents the maximum constraint violation of each constraint in the swarm population. The goal of computing Equation (8.2) is to treat each constraint equally and the $cv(\mathbf{x})$ value lies between 0 and 1 [178].

In solving MOPs, our final goal is to find a set of optimum solutions or the Pareto optimal set. Although the Pareto dominance relation is used to solve the bi-objective optimization problem in Equation (8.1), in this case, we only need to find one global optimum (feasible) solution. This is because if the solution found is infeasible (i.e., $cv(\mathbf{x}) > 0$), it is unacceptable no matter how optimal is the fitness value ($F(\mathbf{x})$). Only the

solutions that are landed on the feasible region (i.e., $cv(\mathbf{x}) = 0$) are considered potential solutions. Figure 8.1 illustrates the feasible region, Pareto front, search space, and the global optimum solution.

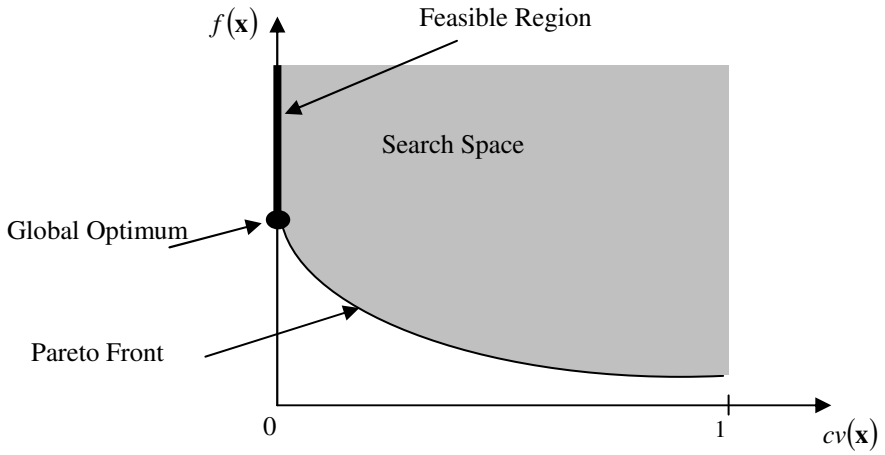


Figure 8.1. Illustration of bi-objective optimization problem ($\mathbf{F}(\mathbf{x})$). The feasible region is mapped to the solid segment. The shaded region represents the search space. The global optimum (black circle) is located beat the intersection of the Pareto front and the solid segment [155].

8.3.2 Proposed PSO Algorithm to Solve COPs

All of the existing constraint handling techniques have two goals: 1) to search for feasible solutions and to guide infeasible solutions towards feasibility; and 2) to converge to the global optimal solution or Pareto front. In view of this fact, we have proposed the essential design elements to achieve these goals: 1) updating personal best procedure based on the rank in the swarm population and constraint violation; 2) maintaining feasible and infeasible global best archive to preserve both feasible and infeasible nondominated solutions, respectively; 3) the acceleration constants in the PSO equation are adjusted based on the feasibility ratio and the constraint violations correspond to the members in personal best archive and global best archive; 4) a mutation procedure is applied in such the range of each decision variable covered for mutation is adaptively

reduced as the number of iterations increases to encourage exploration in early iterations and promote exploitation via fine tuning in the later iterations.

Figure 8.2 presents the pseudocode of the proposed PSO algorithm, where t represents the iteration count and the parameter r_f is the feasibility ratio of the particles' personal best (*pbest*). In the following, the key procedures (highlighted in boldface in Figure 8.2) are elaborated in the following subsections.

```
Begin
/*Initialization
Initialize swarm population and velocity
Set Maximum iterations (tmax)
Set iteration t = 0
Store Personal Best (pbest)

While t < tmax
    Calculate Fitness and Constrain violation
    Apply Pareto dominance Concept
    Update Personal Best (pbest)
    Calculate r_f
    Update Feasible and Infeasible Global Best Archive
    Particle Update Mechanism
    Mutation Operator
    t = t + 1
EndWhile
Report optimum solution in Feasible Global Best Archive
End
```

Figure 8.2 Pseudocode of the proposed PSO algorithm to solve for COPs.

8.3.2.1 Update Personal Best (Pbest) Archive

In [173], the personal best is updated based on the two selection rules: 1) nondominated particles are better than dominated ones; and 2) a particle with lower constraint violation is better than a particle with higher constraint violation. The drawback of these rules is to determine which rule should be prioritized first. If rule one is given higher priority, the progress of searching for feasible regions may slow down since personal best indirectly influence the particles' search behavior in the swarm

population. On the contrary, if rule two is given higher priority, all infeasible solutions will quickly land on the feasible regions but this will indirectly degrade the diversity in the swarm population and may results in premature convergence. Hence it is important to update personal best using both rules at the same time to maintain a balance between convergence to fitter particles and search of feasible regions.

In this study, we propose the following equation to incorporate the rank value and scalar constraint violation of a particle (with decision vector $\mathbf{x}$) to update the personal best if the latest recorded personal best of a particle is in infeasible region.

$$RC(\mathbf{x}) = \left(1 - \frac{1}{rank(\mathbf{x})}\right) + cv(\mathbf{x}), \tag{8.4}$$

where $RC(\mathbf{x})$ is the rank-constraint violation indicator of particle with decision vector $\mathbf{x}$, $rank(\mathbf{x})$ represents the current rank value, while $cv(\mathbf{x})$ refers to the scalar constraint violation of the particle with decision vector $\mathbf{x}$. The rank values are obtained from applying the Pareto ranking [25] to the swarm population. Refer to Equation (8.4), the first term indicates the dominant relationship of the particles comparing the others and it is mapped between zero and one, where zero indicates non-dominated particles and any values greater than zero indicates particle is dominated in various degrees. The purpose is to search for the non-dominated solutions regardless of if the solutions are infeasible, and these solutions will possibly indirectly influence the improvement of the particles in the next iterations in terms of convergence. However, this does not guarantee that the particles will move towards the feasible regions easily since most of the time the searching is spent in the infeasible regions [155]. So, the second term is added to Equation (8.4) to emphasize the current state of the particles in terms of their feasibility

or the degree of infeasibility in the current population. Note that the range of RC is between 0 and 2, and a particle with smaller RC value indicates better solution in terms of its convergence and feasibility status.

The updating procedures, perform in every iteration, are summarized in Figure 8.3.

Function UpdatePbestArchive(*particle*,*Pbest_Archive*, $F(\mathbf{x})$, $cv(\mathbf{x})$)

```
/* particle = a particles (with decision vector x) in the swarm population
/* Pbest_Archive = Previously recoded personal best
/* F(x) = a Particle's fitness value
/* cv(x) = scalar constraint violation of a particle
Begin
    If Pbest_Archive={ }          /*an empty set
        Compute RC value (Equation 8.4)
        Record RC value ( pbest_RC(x) )
        Record particle's position ( pbest(x) )
        Record particle's fitness value ( pbest_fitness(x) )
        Record particle's constraint violation (Pbest_cv or
    cv(pbest(x)) )
    Else
        If cv(pbest(x)) > 0   /*infeasible
            If RC(x) ≤ pbest_RC(x)
                Update pbest_RC(x)
                Update pbest(x)
                Update pbest_fitness(x)
                Update Pbest_cv or cv(pbest(x))
            EndIf
        Else    /*If Pbest_i is feasible
            If cv(x) = 0 and {(RC(x) ≤ pbest_RC(x)) or (F(x) ≤ Pbest_fitness(x))}
                Update pbest_RC(x)
                Update pbest(x)
                Update pbest_fitness(x)
                Update Pbest_cv or cv(pbest(x))
            EndIf
        EndIf
    EndIf
End
```

Figure 8.3 Pseudocode of updating the particles best archive.

Refer to Figure 8.3, when a recorded personal best is infeasible, then the updating step depends on Equation (8.4). On the other hand, if it is feasible, then the updating is done by comparing the rank values (because $cv(\mathbf{x}) = 0$) or fitness values of the particles in the swarm population with those recorded in the personal best archive. The updating of infeasible recorded personal best is based on RC values in order to emphasize both convergence and feasibility; while the updating of feasible recorded personal best is based on dominance relationship in order to support the convergence towards the global optimum in the feasible region.

Once the updating procedure is completed, the feasibility ratio of the particles' personal best (r_f) is updated via the following equation:

$$r_f = \frac{number\ of\ \ particles'\ personal\ best\ that\ are\ feasible}{swarm\ population\ size}. \tag{8.5}$$

8.3.2.2 Update Feasible and Infeasible Global Best Archive

Recent studies have realized the advantage of using infeasible solutions to search for global optimum solution [165,166,179]. One purpose is to promote diversity during the search process through a balance between feasible and infeasible solutions [157,165]. Another purpose is to use the infeasible solutions as the bridge to explore isolated feasible regions in order to search for better feasible solutions and to deal with the case where the proportional feasible region is relatively smaller compared to the entire search space. Hence we propose a fixed size global best archive that stores only the best feasible solution found so far and the infeasible nondominated solutions that have minimum scalar constraint violation found so far.

There are two separate procedures to update the global best archive and they are summarized here:

- **Procedure to update the best feasible solution:** If there is no feasible solution in the archive, the best feasible solution (feasible solution with minimum fitness) is immediately accepted in the archive. If the achieve has recorded the best feasible solution in the pervious iteration, then the new best feasible solution in the current iteration is compared. If the recorded one has larger fitness value, then the current best feasible solution will replace the recorded one, otherwise the current one is removed.
- **Procedure to update the infeasible nondominated solutions**: If the archive is empty or has no infeasible members, the infeasible nondominated solutions are accepted to fill up the archive. If there are infeasible members in the archive, the scalar constraint violation of the new infeasible nondominated solutions is compared with the particle with the largest scalar constraint violation stored in the archive. Those new infeasible nondominated solutions with scalar constraint violation exceed the largest scalar constraint violation stored in the archive are removed. Then, the remaining new infeasible nondominated solutions are compared with respect to any infeasible members in the archive. If any infeasible new solutions are not dominated by any archive infeasible members, they are accepted into the archive. Similarly, any archive infeasible members dominated by any new infeasible solutions are removed from the archive.

Once the two procedures are completed and if the archive size exceeds the allocated size, then Harmonic distance [180] is applied to remove the crowded members and to maintain diversity among the archive members. Afterwards, the crowded

tournament selection operator is applied to select the global best leaders (*gbest*) from achieve to update the particle velocity and position equations (see Equations (8.6) and (8.7)). Note also that the archive will be used on mutation operator procedure.

8.3.2.3 Particle Update Mechanism

In the original PSO design, the particles modeled the swarm behavior and flew through the hyperdimensional space to search for possible optimal solutions. The movement of particles is influenced by their past experiences, i.e., their personal past experience and successful experience attained by their peers. Cushman [175] added a global worst term (Gworst) with a very low acceleration constant (suggested 0.0001) to nudge the particles away from the center of the least feasible solution. Lu and Chen [172] replaced the inertial term with personal and global bests in order to restrict the velocity term so that those feasible particles (solutions) will not be moved away from the feasible regions.

In our design, we make use of the scalar constraint violation and the feasibility ratio (r_f) (i.e., Equation 8.5) to adjust the acceleration constants. The purpose is to guide the particles towards feasibility first and then influence them to search for global optimal solution. The scalar constraint violation belongs to the members in personal best and global best archives. The new PSO equation and its new acceleration constants are formulated as follow:

$$v_{i,j}(t+1) = w \times v_{i,j}(t) + c_1 \times r_1 \times \left(pbest_{i,j}(t) - x_{i,j}(t)\right) + c_2 \times r_2 \times \left(gbest_j - x_{i,j}(t)\right) \tag{8.6}$$

$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1) \tag{8.7}$$

where

$$c_1 = 0.5 \times \left(\left(1 - r_f\right) + \left(1 - pbest\_cv_i\left(t\right)\right) \right) \tag{8.8a}$$

$$c_2 = 0.5 \times \left(r_f + \left(1 - gbest\_cv\right) \right) \tag{8.8b}$$

$v_{i,j}(t)$ is the *j*th dimensional velocity of particle *i* in iteration *t*; $x_{i,j}(t)$ is the *j*th dimensional position of particle *i* in iteration *t*; $pbest_{i,j}(t)$ denotes the *j*th dimensional personal best position of the particle *i* in iteration *t*; $pbest\_cv_i(t)$ is the scalar constraint violation of the personal best of particle *i* in iteration *t*, $gbest_j$ is the *j*th dimensional global best selected from the global best archive; $gbest\_cv$ represents the scalar constraint violation of the selected $gbest$; r_1 and r_2 are a random numbers within $[0,1]$ that are regenerated every time they occur; w is the inertial weight, set to varied between 0.1 to 0.7 (as suggested in [121] to eliminate the difficulty of fine tuning the inertial weight); and c_1 and c_2 are the acceleration constants. Please note the PSO flight equations stress the update mechanism of a particle *i* (so the variable *t* and subscript *i* are used). On the other hand, Equations (8.1)-(8.3) emphasizes the scalar constraint violation of a particle with decision vector $\mathbf{x}$. Specifically, $pbest\_cv_i(t)$ refers to $cv(pbest(\mathbf{x}))$ in iteration *t*.

Adjustment of acceleration constants: Refer to Equations (8.8a) and (8.8b), the values of c_1 and c_2 are influenced by the feasibility ratio of the particles' personal best (r_f) and the amount of constraint violations of the $pbest$ and $gbest$. In general, if c_1 is larger than c_2, the second term in Equation (8.6), i.e., $c_1 \times r_1 \times \left(pbest_{i,j}(t) - x_{i,j}(t)\right)$, is emphasized, in which, the movement of a particle depends more on their personal past

experience than the global experiences attained by the whole swarm population. Table 8.1 briefly summarizes the effect of r_f, $pbest\_cv$, and $gbest\_cv$ on the second and third terms in Equation (8.6). Observing Table 1, we can generally conclude that small r_f will influence the particles to favor on searching for feasible regions instead of optimum solution, while with small $gbest\_cv$ and large r_f, the particles are inclined to search for optimum solution. However, both $pbest\_cv$ and $gbest\_cv$ will also guide the particles towards feasibility but in an indirect manner.

Table 8.1 Brief summary of the effects of r_f, $pbest\_cv$, and $gbest\_cv$ on the second and third terms in Equation (8.6)

r_f	$pbest\_cv$	$gbest\_cv$	**Comments**
small	small	small	$c_1 > c_2$; slightly emphasize on the second term (Both terms will guide the particle towards feasibility)
small	small	large	$c_1 >> c_2$; emphasize on the second term (Second term guides the particle towards feasibility)
small	large	small	$c_1 \approx c_2$; both terms may have equal emphasis (Both terms will guide the particle towards feasibility and find better solutions)
small	large	large	$c_1 > c_2$; emphasize on the second term (Second term guides the particle towards feasibility)
large	small	small	$c_1 < c_2$; emphasize on the third term (Third term guides the particle to find better solutions)
large	small	large	$c_1 \approx c_2$; both terms may have equal emphasis (Both terms will guide the particle towards feasibility and find better solutions)
large	large	small	$c_1 << c_2$; emphasize on the third term (Third term guides the particle to find better solutions)
large	large	large	$c_1 < c_2$; slightly emphasize on the third term (Third term guides the particle to find better solutions)

8.3.2.4 Mutation Operator

In this chapter, the mutation procedure proposed in [120] is applied. The procedure is performed in the randomly selected dimension(s) of the decision variables in

order to bring the particles from being trapped in the local optima. However, in [120], the mutation covers the full range (upper and lower bound of the decision variables in Equation (2.4)). In our approach, the range covered for mutation is adaptively reduced as the number of iterations increases. The idea is as follow: Initially, the mutation operator covers the full range to allow the particles to explore the whole search space, hoping to search for feasible region or better solutions. As the number of iteration increases, the range of the search space covered reduces via a nonlinear equation to reduce the effect of mutation operation in the sense of global search, and encourage fine tuning of the local search. The following presents the nonlinear equation to control the range covered:

$$\%Range\_reduced = \begin{cases} (1-lb)\times\left(1-2\times(2T)^2+\dfrac{lb}{1-lb}\right), & 0 \le T \le 0.25 \\ (1-lb)\times\left(2\times(2T-1)^2+\dfrac{lb}{1-lb}\right), & 0.25 < T \le 0.4, \\ lb\times(1-(T-0.4))^{\alpha}, & 0.4 \le T \le 1 \end{cases} \quad (8.9)$$

$$\text{where} \quad lb = \frac{\beta}{number\ of\ decision\ \text{var}iables} \quad \text{and} \quad (8.10)$$

$$T = \frac{Current\ iteration}{maximum\ number\ of\ iterations}. \quad (8.11)$$

lb is the lowest percentage allowable parameter to narrow the search space with a quick pace, before entering the finely narrowing the search space and β is user defined parameter; while α is another user defined parameter control how fast to finely narrow the range covered. Figure 8.4 depicts the Equation (8.9). Observing in Figure 8.4, the range covered is slowly decreases from 0 to 25% of the maximum iterations, provide opportunity for the particles to explore the entire search space. Next, the mutation range covered decrease quickly within 25% and 40% of the maximum iterations to narrow

down the exploration in the search space since at this stage the global particles will either closer or in the feasible region. Lastly, for the remaining iteration count the mutation range covered is slowly reduced to create a path way to provide changes for the mutated particles to explore locally and towards the global optimum.

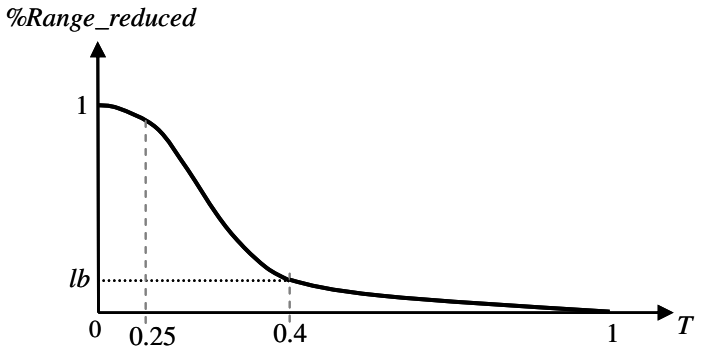


Figure 8.4 Graph for percentage range to be reduced against *T*.

```
Function MutateParticles(particles, current and previous best
solutions)
```

```
   /* particles = current particles in the swarm population
   /*Note: Mutation operator procedure see [120].
   Begin
      Randomly select particles from half of the swarm population
      Calculate %Range_reduced from Equations (8.9–8.11)
      For each selected particle
             If Previous best solution equals to current best solution
                 If rand<0.8  /* mutation rate of 0.8
                    Apply mutation operator on best solution
                 Else
                    Apply mutation operator on the current particle
                 EndIf
                  Replace the current particle with the mutated one
            Else
                 Apply mutation operator on the current particle
                  Replace the current particle with the mutated one
              EndIf
       EndFor
   End
```

Figure 8.5 Pseudocode of mutation operator applies to the swarm population.

The mutation procedure is applied to half of the population size. The particles to be mutated are chosen randomly. In addition, during the later stage of the search process,

the particles may trap in the local optimal. Hence, the best solution from previous iteration and the best solution from current iteration are compared. If they are the same, then with a higher probability, the mutation operator is applied to the best solution and replaces the mutated one with a selected particle in the swarm population. The idea is to push the particles to advance towards the global optimum solution. The following pseudocode (Figure 8.5) shows how the mutation operator is applied to the swarm population.

8.3.3 Proposed Constrained MOPSO to Solve CMOPs

In this section, we extended the proposed PSO to a constrained MOPSO for CMOPs. In the proposed MOPSO, the technique of converting a COP into an unconstrained bi-objective optimization problem is applied to transform the CMOPs into an unconstrained tri-objective optimization problem. Note that we use the term 'tri-objective optimization problem' since in this study, CMOPs with two objective functions are considered. For this technique, both the inequality and equality constraints (i.e., Equations (2.2) and (2.3), respectively) are treated as one objective and the other objectives are the original two objective functions. Hence, Equations (2.1)-(2.3) are transformed into the following general form of an unconstrained tri-objective optimization problem:

$$\text{Minimize } \mathbf{F}(\mathbf{x}) = \left[cv(\mathbf{x}), F_1(\mathbf{x}), F_2(\mathbf{x})\right], \tag{8.12}$$

where $cv(\mathbf{x})$ is the scalar constraint violation of a decision vector $\mathbf{x}$ (or particle) and it is mathematically formulated as below:

$$cv(\mathbf{x}) = \frac{1}{p}\sum_{j=1}^{p}\frac{cv_j(\mathbf{x})}{cv_{\max}^{j}} \tag{8.13}$$

where $cv_j(\mathbf{x}) = \begin{cases} \max\left(0, g_j(\mathbf{x})\right), & j = 1, \ldots, m \\ \max\left(0, \left|h_j(\mathbf{x})\right| - \delta\right), & j = m+1, \ldots, p \end{cases}$, (8.14a)

$$cv_{\max}^{j} = \max_{\mathbf{x} \in \boldsymbol{S}} cv_j(\mathbf{x}). \tag{8.14b}$$

Parameter δ is the tolerance allowed for equality constraints, usually δ is set to 0.001 or 0.0001. If a particle or solution ($\mathbf{x}$) satisfies the *j*th constraints, then $cv_j(\mathbf{x})$ is set to zero, otherwise it is greater than zero. In solving MOPs, our final goal is to find the Pareto optimum set. Although the Pareto dominance relation is used to solve the tri-objective optimization problem in Equation (8.12), in this case, we only need to find the Pareto front of two objective functions. This is because if the set of nondominated solutions found is infeasible (i.e., $cv(\mathbf{x}) > 0$), it is unacceptable no matter how high quality the Pareto front of the three objective functions is produced. Only the set of nondominated solutions that are landed on the feasible regions (i.e., $cv(\mathbf{x}) = 0$) are considered potential Pareto front.

The general design procedure of the proposed constrained MOPSO is the extension of the proposed constrained PSO. The same designs elements are employed with exception of the mutation procedure. The brief summary of the essential design elements in the proposed MOPSO are given: 1) updating personal best procedure based on the rank in the swarm population and constraint violation; 2) maintaining feasible and infeasible global best archive to preserve both feasible and infeasible nondominated solutions, respectively; 3) the acceleration constants in the PSO equation are adjusted based on the feasibility ratio and the constraint violations correspond to the members in personal best archive and global best archive; and 4) the mutation rate is adaptively updated based on the feasibility ratio, in which a higher frequency of applying mutation

operator to the swarm population if there are few feasible particles to promote exploration, otherwise a lower frequency on activating the mutation operator if there are many feasible particles. Mutation operators that support global and local searches are used.

Although the same design elements are adopted but some modifications on these design elements are needed to accommodate the nature of MOPs. Figure 8.6 presents the pseudocode of the proposed constrained MOPSO. Procedures that are slightly modified are highlighted in bold and are discussed below.

```
Begin
/*Initialization
Initialize swarm population and velocity
Set Maximum iterations (tmax)
Set iteration t = 0
Store Personal Best (pbest)

While t < tmax
    Calculate Fitness and Constrain violation
    Apply Pareto dominance Concept
    Update Personal Best (pbest)
    Calculate r_f
    Update Feasible and Infeasible Global Best Archive
    Global Best Selection
    Particle Update Mechanism
     Mutation Operator
    t = t + 1
EndWhile
Report optimal Pareto front in Feasible Global Best Archive
End
```

Figure 8.6 Pseudocode of the proposed constrained MOPSO algorithm.

8.3.3.1 Updating Personal Best Archive

The updating of personal best archive procedures are slightly modified and are summarized below. Note that the procedures are done in every iteration.

- If the personal best archive is empty, record all computed RC values of all particles, including their corresponding positions ($pbest$) and their degree of constraint violations ($pbest\_cv$).
- If the personal best archive is nonempty, then for those recorded personal best that are *infeasible*, their recorded RC values are compared with the RC values of their corresponding particles in the swarm population. Any of the current particles with smaller RC values will replace the recorded ones, including updating the corresponding RC values, $pbest$, and $pbest\_cv$. However, for those recorded personal best that are *feasible* (i.e., $cv(x) = 0$), pure Pareto ranking [25] is applied to these personal best and their corresponding particles in the swarm population. If the current particle dominates their corresponding personal best, then the current one will replace the recorded one. If both do not dominate each other, one of them is randomly chosen to update the personal best archive. Similarly, the updating of personal best archive includes updating the RC values, $pbest$, and $pbest\_cv$.

The updating of infeasible recorded personal best is based on RC values in order to emphasize both convergence and feasibility; while the updating of feasible recorded personal best is based on dominance relationship in order to support the convergence towards the Pareto front in the feasible region.

8.3.3.2 Updating Feasible and Infeasible Global Best Archive

For the proposed constrained MOPSO, we propose two fixed size global best archives, i.e., feasible and infeasible global best archives. Feasible global best archive stores only the best feasible solution found so far, while infeasible global best archive

stores the infeasible solutions that have minimum scalar constraint violation found so far. The solutions in both archives serve as potential global best candidates (*gbest)* for the particle flight update.

To maintain the archive, first the new nondominated particles from the swarm population are found. Then, these new nondominated particles are divided into new feasible nondominated solutions and infeasible nondominated solutions. The procedures to maintain both global best archives are summarized below:

- **Maintaining Feasible Global Best Archive**: At each iteration count, new *feasible* nondominated solutions are compared with respect to any members in the archive. If new feasible solutions are not dominated by any archive members, they are accepted into the archive. Similarly, any archive members dominated by any new feasible solutions are removed from the archive. If the archive population size exceeds the allocated archive size, then harmonic distance [180] is applied to remove the crowded members and to maintain diversity among the archive members.
- **Maintaining Infeasible Global Best Archive:** In the first procedure, the scalar constraint violation of the new infeasible nondominated solutions is compared with the largest scalar constraint violation stored in the archive. Those new infeasible nondominated solutions with scalar constraint violation exceed the largest scalar constraint violation stored in the archive are removed. Then, in the second procedure, the remaining new infeasible nondominated solutions are compared with respect to any members in the archive. If any new solutions are not dominated by any archive members, they are accepted into the archive. Similarly, any archive members dominated by any new solutions are removed from the archive. If the archive

population size exceeds the allocated archive size, then harmonic distance [180] is applied to remove the crowded members and to maintain diversity among the archive members.

8.3.3.3 Global Best Selection

As mentioned in previous subsection, the infeasible global best archive plays a vital role in finding the global optimum because the infeasible members will lead the particles towards the feasible regions especially if the feasible region is very small compared to the entire objective space or act as a bridge to bring the particles from infeasible regions to other isolated feasible regions. The members from the feasible global best archive will guide the particles to search for global optimum in the feasible regions. With equal probability, *gbest* is selected either from feasible global best archive or infeasible global best archive. This is to give equal probability of utilizing the feasible and infeasible *gbest* to guide the particles. Unless one of the archives is empty, then by default the *gbest* is selected from remaining nonempty archive. Once which archive is chosen, the crowding distance values of the archive members are used to guide the particles to select their *gbest*.

8.3.3.4 Mutation Operator

In this approach, two mutation operators are applied, i.e., uniform and Gaussian mutation operators. Uniform mutation aims to encourage exploration in the swarm population and is presented in Equation (8.15), while Gaussian mutation in Equation

(8.16) promotes exploitation among the particles in the swarm population via local search characteristics.

$$x_{i,j}(t) = r_u\left(x_{i,j}^U - x_{i,j}^L\right) \tag{8.15}$$

$$x_{i,j}(t) = x_{i,j}(t) + \beta_i \tag{8.16}$$

where $x_{i,j}(t)$ is the *j*th dimensional position of particle *i* in iteration *t*; r_u is a random number within $[0,1]$; $x_{i,j}^U$ and $x_{i,j}^L$ are the *j*th dimensional upper and lower bound of particle *i*; and β_i represents a random number in which it is drawn from the Gaussian distribution, $Gaussian\left(0, P_m\left(x_{i,j}^U - x_{i,j}^L\right)\right)$. Parameter P_m is computed using Equations (8.17) and (8.18) [135].

$$lb = \frac{0.1}{n},\ n = \text{number of decision variables} \tag{8.17}$$

$$P_m = \begin{cases} (0.5 - lb) \times \left(1 - 2 \times r_f^2 + \dfrac{lb}{0.5 - lb}\right), & 0 \le r_f \le 0.5 \\ (0.5 - lb) \times \left(2 \times (r_f - 1)^2 + \dfrac{lb}{0.5 - lb}\right), & 0.5 < r_f \le 1 \end{cases} \tag{8.18}$$

The P_m parameter represents the mutation rate and is adaptively determined by the feasibility ratio of the particles' personal best (r_f). The idea is to allow for a higher mutation rate when there are fewer feasible particles (r_f is small) or vice versa. Figure 8.7 is the illustration of Equation (8.18). lb represents the minimum allowable mutation rate and is determined from Equation (8.14). If $r_f = 1$, mutation rate will remain lb. For simplicity, a random number r_n with uniform distribution between $[0,1]$ is generated to decide which mutation operator is applied. If $r_n < 0.5$, uniform mutation is applied, otherwise Gaussian mutation is applied.

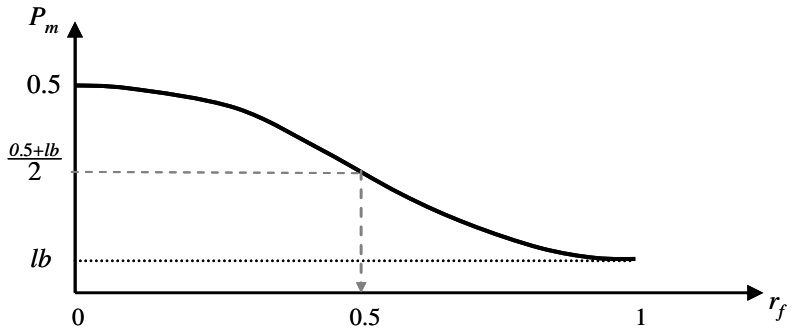


Figure 8.7 Mutation rate (P_m) versus feasibility ratio of the particles' personal best (r_f).

8.4 Comparative Study

Two experiments are performed. The first experiment evaluates the performance of the proposed PSO for COPs and compares the results against the selected constrained approaches, while the second experiment evaluates the performance of the proposed constrained MOPSO against two state-of-the-art constrained MOEAs.

8.4.1 Experiment 1: Performance Evaluation of the Proposed PSO for COPs

8.4.1.1 Experimental Framework

Thirteen well-known benchmark functions [181] are used to test the performance of the proposed constrained PSO. Table 8.2 presents the summary of the main characteristics of all teat functions. It provides the type of objective functions (i.e., linear, nonlinear, cubit, quadratic) and their types of constraint functions (i.e., linear inequality (LI), nonlinear inequality (NI), linear equality (LE), and nonlinear equality (NE)). The parameter **n** represents the number of decision variables, and parameter **a** represents the number of inequality constraints that are active. The parameter ρ is called feasibility ratio. This ratio is determined by calculating the percentage of feasible solutions out of

1,000,000 randomly generated solutions in the entire search space [157]. If the feasibility ratio is very small, this challenges the algorithms to search for feasible solutions.

Table 8.2 Summary of main characteristics of the 19 benchmark functions.

Problems	n	Type of function	ρ	LI	NI	LE	NE	a
g01	13	Quadratic	0.0111%	9	0	0	0	6
g02	20	Nonlinear	99.9971%	1	1	0	0	1
g03	10	Nonlinear	0.0000%	0	0	0	1	1
g04	5	Quadratic	52.1230%	0	6	0	0	2
g05	4	Cubic	0.0000%	2	0	0	3	3
g06	2	Cubic	0.0066%	0	2	0	0	2
g07	10	Quadratic	0.0003%	3	5	0	0	6
g08	2	Nonlinear	0.8560%	0	2	0	0	0
g09	7	Nonlinear	0.5121%	0	4	0	0	2
g10	8	Linear	0.0010%	3	3	0	0	3
g11	2	Quadratic	0.0000%	0	0	0	1	1
g12	3	Quadratic	4.7713%	0	9^3	0	0	0
g13	5	Nonlinear	0.0000%	0	0	0	3	3
g14	10	Nonlinear	0.0000%	0	0	3	0	3
g15	3	Quadratic	0.0000%	0	0	1	1	2
g16	5	Nonlinear	0.0204%	4	34	0	0	4
g17	6	Nonlinear	0.0000%	0	0	0	4	4
g18	9	Quadratic	0.0000%	0	13	0	0	6
g19	15	Cubic	33.1761%	0	5	0	0	0

Table 8.3. Parameter configurations for the proposed PSO.

Test Problems	Parameter settings	
	β	α
g02, g10, g19	1	10
g04, g12, g16	1	15
g11	1	16
g01, g06	1	20
g03, g08	0.1	10
g18	0.1	11
g07, g13, g14, g15, g17	0.1	15
g09	0.1	20
g05	0.1	25

Parameter configurations of the proposed PSO for each test function are presented in Table 8.3. For each test function, we perform 300,000 fitness function evaluations and conduct 30 independent runs [155]. The experiment is implemented in Matlab software.

8.4.1.2 Simulation Results and Analysis

Table 3 presents the best, median, worst, and mean results obtained by the proposed constrained PSO for each test function. Among the nineteen test problems, the proposed PSO is able to obtain the optimal for g01, g04, g06, g08, g11, g12, g15, and g16. The following test problems: g02, g03, and g18 have the best results that are very close to the optimum. From Table 8.4, we can note that only g17 has one infeasible run among the 30 independent runs. This means among 30 runs, only run that the proposed constrained PSO is unable to find any feasible solutions. This case is rare and considered an extreme case. In addition, the constrain violation for the infeasible run is extremely low, which is 1.7859E-05.

The proposed constrained PSO is compared against the four selected approaches. They are: Stochastic Ranking method (SR) [155], dynamic-objective method and restricted velocity particle swarm optimization (DOM+RVPSO) [172], master-slave particle swarm optimization (MSPSO) [179], and feasibility tournament and perturbing the particle's memory (PESO) [182]. The experiment results for the thirteen test problems are listed in Table 8.5. Observed Table 8.5, our algorithm can achieve the same or better performance than some of the selected approaches for the following test problems, g01, g03, g04, g06, g08, g11, and g12. The proposed algorithm is unable to obtain the best performance compared to some of the selected approaches for the test problems: g02, g05, g07, g09, g10, and g13. However, the proposed algorithm is able to performance better than MSPSO for test problems g05, g07, and g09, and obtains better performance than DOM+RVPSO for test problems g02 and g13. For test problem g13,

the proposed algorithm obtains the better results for mean and worst when compared to SR.

Table 8.4 Experimental results on the 19 benchmark functions with 50 independent runs. Note that the first column presents the test problem and its global optimal.

Problems/ optimal	Best	Median	Worst	Mean	Std	Infeasible Runs
g01/ -15.000000	**-15.000000**	-15.000000	-13.000000	-14.840000	5.54E-01	0
g02/ -0.803619	-0.803618	-0.793081	-0.772503	-0.792893	7.45E-03	0
g03/ -1.0005001	-1.0004927	-1.0004739	-1.0003476	-1.0004554	4.42E-05	0
g04/ -30665.539	**-30665.539**	-30665.539	-30665.538	-30665.539	8.39E-05	0
g05/ 5126.4981	5126.5753	5137.4183	5199.3823	5143.8872	1.82E+01	0
g06/ -6961.8138	**-6961.8138**	-6961.8132	-6961.8078	-6961.8128	1.12E+03	0
g07/ 24.3062091	24.346823	24.734355	25.248314	24.765756	2.26E+01	0
g08/ -0.095825	**-0.095825**	-0.095825	-0.095825	-0.095825	9.65E-13	0
g09/ 680.63006	680.63260	680.65159	680.73852	680.65835	2.07E-02	0
g10/ 7049.248	7086.0745	7427.0046	7627.0983	7408.8877	1.12E+02	0
g11/ 0.7499000	**0.7499000**	0.7499001	0.7499051	0.7499006	1.19E-06	0
g12/ -1.000000	**-1.000000**	-1.000000	-1.000000	-1.000000	0	0
g13/ 0.0539498	0.0539645	0.0568509	0.1121465	0.0607371	1.19E-01	0
g14/ -47.764888	-47.326592	-45.232864	-43.439452	-45.357137	9.91E-01	0
g15/ 961.71502	**961.71502**	961.71759	961.99117	961.73767	5.69E-02	0
g16/ -1.905155	**-1.905155**	-1.905155	-1.905154	-1.905155	3.35E-07	0
g17/ 8853.53967	8854.0298	8927.6184	8975.5617	8912.7110	4.96E+01	1
g18/ -0.866025	-0.866021	-0.865341	-0.858935	-0.864311	2.25E-03	0
g19/ 32.655592	33.707518	36.240328	40.272815	36.329895	1.81E+00	0

Table 8.5 Comparison of the proposed algorithm with respect to SR[155], DOM+RVPSO [172], MSPSO [179], and PESO [182] on 13 benchmark functions. Note that the first column presents the test problem and its global optimal.

Problems/ optimal		SR[153]	DOM+RVPSO [170]	MSPSO [177]	PESO [180]	Proposed
g01/ -15.000	Best	**-15.000**	**-15.000**	**-15.000**	**-15.000**	**-15.000**
	Mean	**-15.000**	-14.419	**-15.000**	**-15.000**	-14.840
	Worst	**-15.000**	-12.453	**-15.000**	**-15.000**	-13.000
	St. dev	1.3E-13	8.5E-01	4.12E-04	**0**	5.54E-01
g02/ -0.803619	Best	**-0.803619**	-0.664028	-0.803020	**-0.803619**	-0.803618
	Mean	-0.772078	-0.413257	-0.800418	**-0.801320**	-0.792893
	Worst	-0.683055	-0.259980	**-0.799342**	-0.786566	-0.772503
	St. dev	2.6E-02	1.2E-01	**1.51E-03**	4.59E-03	7.45E-03
g03/ -1.001	Best	**-1.001**	-1.005	1.000	**-1.001**	**-1.001**
	Mean	**-1.001**	-1.003	0.998	**-1.001**	**-1.001**
	Worst	**-1.001**	-0.933	0.996	-1.000	-1.000
	St. dev	**6.0E-09**	1.3E-02	1.58E-03	3.15E-07	4.42E-05
g04/ -30665.539	Best	**-30665.539**	**-30665.539**	-30665.537	**-30665.539**	**-30665.539**
	Mean	**-30665.539**	**-30665.539**	-30663.010	**-30665.539**	**-30665.539**
	Worst	**-30665.539**	**-30665.539**	-30658.300	**-30665.539**	-30665.538
	St. dev	2.2E-11	1.2E-11	2.79E+00	**0**	8.39E-05
g05/ 5126.4981	Best	5126.497	5126.4842	5126.6051	**5126.4981**	5126.5753
	Mean	5126.497	5241.0549	5129.8001	**5126.4981**	5143.8872
	Worst	5126.497	5708.2250	5157.2247	**5126.4981**	5199.3823
	St. dev	6.2E-12	1.8E+02	1.25E+01	**0**	1.82E+01
g06/ -6961.814	Best	**-6961.814**	**-6961.814**	-6961.830	**-6961.814**	**-6961.814**
	Mean	**-6961.814**	**-6961.814**	-6957.760	**-6961.814**	-6961.813
	Worst	**-6961.814**	**-6961.814**	-6954.650	**-6961.814**	-6961.808
	St. dev	6.4E-12	4.6E-12	2.69E+00	**0**	1.12E+03
g07/ 24.306	Best	**24.306**	24.306	24.373	**24.306**	24.347
	Mean	**24.306**	24.317	24.180	**24.306**	24.766
	Worst	24.308	24.385	23.750	**24.306**	25.248
	St. dev	2.7E-04	2.4E-02	2.53E-01	**3.34E-06**	2.26E+01
g08/ -0.095825	Best	**-0.095825**	**-0.095825**	**0.095825**	**0.095825**	**-0.095825**
	Mean	**-0.095825**	**-0.095825**	**0.095825**	**0.095825**	**-0.095825**
	Worst	**-0.095825**	**-0.095825**	**0.095825**	**0.095825**	**-0.095825**
	St. dev	4.2E-17	1.4E-17	**0**	**0**	2.07E-02
g09/ 680.630	Best	**680.630**	**680.630**	680.660	**680.630**	680.633
	Mean	**680.630**	**680.630**	681.002	**680.630**	680.658
	Worst	**680.630**	**680.630**	684.113	**680.630**	680.739
	St. dev	4.6E-13	**5.4E-13**	1.48E+00	**0**	2.07E-02
g10/ 7049.248	Best	**7049.248**	**7049.2480**	7051.690	**7049.248**	7086.075
	Mean	**7049.249**	7049.2701	7054.710	**7049.249**	7408.888
	Worst	7049.296	7049.5969	7059.280	**7049.264**	7627.098
	St. dev	**4.9E-03**	7.9E-02	2.84E+00	3.61E-03	1.12E+02
g11/ 0.750	Best	**0.750**	0.749	**0.750**	**0.750**	**0.750**
	Mean	**0.750**	0.749	**0.750**	**0.750**	**0.750**
	Worst	**0.750**	0.749	**0.750**	**0.750**	**0.750**
	St. dev	1.8E-15	2.4E-12	**0**	**0**	1.19E-06
g12/ -1.000	Best	**-1.000**	**-1.000**	NA	**-1.000**	**-1.000**
	Mean	**-1.000**	**-1.000**	NA	**-1.000**	**-1.000**
	Worst	**-1.000**	**-1.000**	NA	**-1.000**	**-1.000**
	St. dev	9.6E-10	**0**	NA	**0**	**0**
g13/ 0.0539498	Best	0.053942	0.0538666	NA	**0.053950**	0.0539645
	Mean	0.096276	0.0681124	NA	**0.053950**	0.0607371
	Worst	0.438803	2.0428924	NA	**0.053965**	0.1121465
	St. dev	1.2E-01	4.0E-01	NA	**2.76E-06**	1.19E-01

8.4.2. Experiment 2: Performance Evaluation of the Proposed Constrained MOPSO

8.4.2.1. Experimental Framework

Table 8.6 Parameter configurations for testing algorithms.

Algorithms	Parameter Settings
NSGA-II [31]	Population size =100; crossover probability = 0.9; mutation probability = $1/n$; SBX crossover parameter = 20; polynomial mutation parameter = 20.
GZHW [164]	Population size =100; crossover probability = 0.9; mutation probability = $1/n$; SBX crossover parameter = 20; polynomial mutation parameter = 20; comparison probability = 0.45; penalty parameters, $w_j = 1$, $\beta = 1$.
WTY [166]	Population size =100; Test Functions BNH. CTP1-CTP8, Crossover probability = 0.9; mutation probability = $1/n$; SBX crossover parameter = 10; polynomial mutation parameter = 20. Test Functions SRN, TNK, OSY, CONSTR, and Welded Beam Crossover probability = 0.9; mutation probability = $1/n$; SBX crossover parameter = 5; polynomial mutation parameter = 5.
Proposed MOPSO	Population size =100; feasible and infeasible Gbest archive size = 100.

As stated earlier these exist no prominent constrained MOPSO for CMOPs. Instead, three state-of-the-art constrained MOEAs are chosen for performance comparison. They are NSGA-II [31], Geng *et al.* [164], (indicated by GZHW), and Woldesenbet *et al.* [166] (indicated by WTY). Each algorithm is set to perform 50,000 fitness function evaluations. The parameter configurations for all testing algorithms are summarized in Table 8.6. The fourteen benchmark problems are chosen to evaluate the performance of the proposed MOPSO with the selected MOEAs. All the benchmark problems are two objectives minimization problems and they are listed in Tables 8.7 and 8.8: BNH [160], SRN [161], TNK [183], OSY [184], CTP1-CTP8 [185], CONSTR [1], and Welded Beam [165]. Similar to Table 8.2, the summary of the main characteristics of these benchmark problems are presented in Table 8.9. All algorithms use a real-number representation for decision variables. For each experiment, 50 independent runs were

conducted to collect the statistical results, and the results are illustrated by statistical box plots.

Table 8.7 The 14 benchmark CMOPs used in this study. All objective functions are to be minimized.

Problems	Objective Functions	Constraints	Variable Bounds
BNH [160]	$F_1(\mathbf{x}) = 4x_1^2 + 4x_2^2$ $F_2(\mathbf{x}) = (x_1 - 5)^2 + (x_2 - 5)^2$	$g_1(\mathbf{x}) = (x_1 - 5)^2 + x_2^2 \le 25$ $g_2(\mathbf{x}) = (x_1 - 8)^2 + (x_2 + 3)^2 \ge 7.7$	$x_1 \in [0,5]$ $x_2 \in [0,3]$
SRN [161]	$F_1(\mathbf{x}) = 2 + (x_1 - 2)^2 + (x_2 - 1)^2$ $F_2(\mathbf{x}) = 9x_1 - (x_2 - 1)^2$	$g_1(\mathbf{x}) = x_1^2 + x_2^2 \le 225$ $g_2(\mathbf{x}) = x_1 - 3x_2 + 10 \le 0$	$x_i \in [-20,20]$ $i = 1,2$
TNK [183]	$F_1(\mathbf{x}) = x_1$ $F_2(\mathbf{x}) = x_2$	$g_1(\mathbf{x}) = x_1^2 + x_2^2 - 1 - 0.1\cos\left(16\tan^{-1}\left(\frac{x_1}{x_2}\right)\right) \ge 0$ $g_2(\mathbf{x}) = (x_1 - 0.5)^2 + (x_2 - 0.5)^2 \le 0.5$	$x_i \in [0,\pi]$ $i = 1,2$
OSY [184]	$F_1(\mathbf{x}) = -\left[25(x_1 - 2)^2 + (x_2 - 2)^2 + (x_3 - 1)^2 + (x_4 - 4)^2 + (x_5 - 1)^2\right]$ $F_2(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2$	$g_1(\mathbf{x}) = x_1 + x_2 - 2 \ge 0$ $g_2(\mathbf{x}) = 6 - x_1 - x_2 \ge 0$ $g_3(\mathbf{x}) = 2 - x_2 + x_1 \ge 0$ $g_4(\mathbf{x}) = 2 - x_1 + 3x_2 \ge 0$ $g_5(\mathbf{x}) = 4 - (x_3 - 3)^2 - x_4 \ge 0$ $g_6(\mathbf{x}) = (x_5 - 3)^2 + x_6 - 4 \ge 0$	$x_i \in [0,10]$ $i = 1,2,6$ $x_j \in [1,5]$ $j = 3,5$ $x_4 \in [0,6]$
CTP1 [185]	$F_1(\mathbf{x}) = x_1$ $F_2(\mathbf{x}) = c(\mathbf{x})\exp\left(-\frac{F_1(\mathbf{x})}{c(\mathbf{x})}\right)$ $c(\mathbf{x}) = 1 + x_2$	$g_1(\mathbf{x}) = F_2(\mathbf{x}) - 0.858\exp(-0.541F_1(\mathbf{x})) \ge 0$ $g_2(\mathbf{x}) = F_2(\mathbf{x}) - 0.728\exp(-0.295F_1(\mathbf{x})) \ge 0$	$x_1 \in [0,1]$ $i = 1,2$
CTP2-CTP8 [185]	$F_1(\mathbf{x}) = x_1$ $F_2(\mathbf{x}) = c(\mathbf{x})\left(1 - \frac{F_1(\mathbf{x})}{c(\mathbf{x})}\right)$ $c(\mathbf{x}) = 1 + x_2$	$g(\mathbf{x}) = \cos(\theta)[F_2(\mathbf{x}) - e] - \sin(\theta)F_1(\mathbf{x}) \ge a\left\lvert\sin\left\{b\pi[\sin(\theta)(F_2(\mathbf{x}) - e) + \cos(\theta)F_1(\mathbf{x})]^c\right\}\right\rvert^d$ **See Table 8.8 for parameter setting for CTP test problems	$x_1 \in [0,1]$ $i = 1,2$ Note for CTP8: $x_1 \in [0,1]$ $x_2 \in [0,10]$
CONSTR [1]	$F_1(\mathbf{x}) = x_1$ $F_2(\mathbf{x}) = \frac{1 + x_2}{x_1}$	$g_1(\mathbf{x}) = x_2 + 9x_1 \ge 6$ $g_2(\mathbf{x}) = -x_2 + 9x_1 \ge 1$	$x_1 \in [0.1,1]$ $x_2 \in [0,5]$
Welded Beam Design [165]	$F_1(\mathbf{x}) = 1.10471x_1^2x_3 + 0.04811x_4x_2(14 + x_3)$ $F_2(\mathbf{x}) = \frac{2.1952}{x_4^3x_2}$	$g_1(\mathbf{x}) = 13600 - \tau(\mathbf{x}) \ge 0$ $g_2(\mathbf{x}) = 30000 - \sigma(\mathbf{x}) \ge 0$ $g_3(\mathbf{x}) = x_2 - x_1 \ge 0$ $g_4(\mathbf{x}) = P_c(\mathbf{x}) - 6000 \ge 0$ $\tau(\mathbf{x}) = \sqrt{(\tau')^2 + (\tau'')^2 + \frac{x_3\tau'\tau''}{\sqrt{0.25(x_3^2 + (x_1 + x_4)^2)}}}$ $\tau'(\mathbf{x}) = \frac{6000}{\sqrt{2}x_1x_3}$ $\tau''(\mathbf{x}) = \frac{6000(14 + 0.5x_3)\sqrt{0.25(x_3^2 + (x_1 + x_4)^2)}}{2\sqrt{2}x_1x_3\left(\frac{x_3^2}{12} + 0.25(x_1 + x_4)^2\right)}$ $\sigma(\mathbf{x}) = \frac{504000}{x_4^2x_2}$ $P_c(\mathbf{x}) = 64746.022(1 - 0.0282346x_4)x_4x_2^3$	$x_1 \in [0.125,5]$ $x_2 \in [0.125,5]$ $x_3 \in [0.1,10]$ $x_4 \in [0.1,10]$

Table 8.8 Parameter setting for CTP2-CTP8 [183].

Problems	θ	a	b	c	d	e
CTP2	-0.2π	0.2	10	1	6	1
CTP2	-0.2π	0.1	10	1	0.5	1
CTP2	-0.2π	0.75	10	1	0.5	1
CTP2	-0.2π	0.1	10	2	0.5	1
CTP2	0.1π	40	0.5	1	2	-2
CTP7	-0.05π	40	5	1	6	0
CTP8	0.1π -0.05π	40 40	0.5 2	1 1	2 6	-2 0

Table 8.9 Summary of main characteristics of the 14 benchmark functions.

Problems	Objective Functions	n	ρ	LI	NI	LE	NE	a
BNH	2	2	93.61%	0	2	0	0	0
SRN	2	2	16.18%	1	1	0	0	0
TNK	2	2	5.09%	0	2	0	0	1
OSY	2	6	3.25%	4	2	0	0	6
CTP1	2	2	99.58%	0	2	0	3	1
CTP2	2	2	78.65%	0	1	0	0	1
CTP3	2	2	76.85%	0	1	0	0	1
CTP4	2	2	58.17%	0	1	0	0	1
CTP5	2	2	77.54%	0	1	0	0	1
CTP6	2	2	0.40%	0	1	0	0	1
CTP7	2	2	36.68%	0	1	0	0	0
CTP8	2	2	17.86%	0	2	0	0	1
CONSTR	2	2	52.52%	2	0	0	0	1
Welded Beam	2	4	18.67%	1	3	0	0	0

8.4.2.2 Selected Performance Metrics

All comparisons are based on both quantitative and qualitative measures. Quantitative comparison is based on the plots of the final Pareto fronts in a given run. For quantitative comparison, two performance metrics are taken into consideration to measure the quality of algorithms with respect to dominance relations. The results are illustrated by statistical box plots. The performance metrics used here are the same as given in Subsection 6.4.3: hypervolume indicator (S Metric) and additive binary epsilon indicator.

8.4.2.3 Performance Evaluation

The box plots of hypervolume indicator (the I_H values) are summarized in Figure 8.8. The algorithm with higher I_H values indicates the ability to dominate a larger region in the objective space and with better diversity. In Figure 8.8, the I_H values are normalized for each test problem. So, the highest I_H value will equal one. The figure shows that in general, the proposed MOPSO has high I_H values, with at least higher than 0.9. The proposed MOPSO has the highest I_H value for test problem CTP5. The proposed MOPSO obtains a higher I_H values than GZHW for test problems SRN, CTP1, CTP2, CTP8, and Welded Beam, while it has higher I_H values than NSGA-II for test problem OSY. In addition, the proposed MOPSO has comparable I_H values with WTY for test problems CTP3, CTP4, CTP7, and Welded Beam since they attain the relative close I_H values. Hence, the Mann-Whitney rank-sum test is used to examine the distribution of the I_H values. The tested results are presented in Table 8.10. From Table 8.10, we concluded that the proposed MOPSO and WTY share the same victory for test problem CTP4 while the results also show there is no difference in performance on test function OSY for the proposed MOPSO and GZHW. Refer to Figure 8.8, the box plots indicate that the proposed MOPSO and NSGA-II show comparable I_H values for test problems CTP7 but it is not confirmed by the Mann-Whitney rank-sum test in Table 8.10. Although we observed in Figure 3 that the proposed MOPSO shows the lowest I_H values for test problems BNH, SRN, TNK, CTP2, CTP6, CTP8, and CONSTR compared to the selected MOEAs, the proposed MOPSO does not fall short in terms of performance because it has I_H values higher than 0.99 and the difference between its I_H values compared to those achieved by the selected MOEAs are very small. From the analysis, we concluded that

the proposed MOPSO is competitive in terms of performance compared to the selected MOEAs. In addition, Figure 8.8 shows that the standard deviations for the proposed MOPSO are consistently low; this indicates its ability of producing reliable solutions for the benchmark problems.

Figure 8.9 illustrates the results (summarized in box plots) of additive binary ε-indicator. For each test problem, there are two box plots, i.e., $I_{\varepsilon+}\left(A, B_{1-3}\right)$ and $I_{\varepsilon+}\left(B_{1-3}, A\right)$, in which the proposed MOPSO is represented by A and the algorithm B_{1-3} represent NSGA-II, GZHW, and WTY, respectively. Observe Figure 4, it seems the proposed MOPSO performs slightly better with respect to dominance relation than all of the MOEAs for test functions CTP5 since the $I_{\varepsilon+}\left(A, B_{1-3}\right) \approx 0$ and $I_{\varepsilon+}\left(B_{1-3}, A\right) > 0$. Similarly, it performs better than GZHW for test function Welded Beam. However, for test functions OSY, CTP4, and CTP7, the performance of the proposed MOPSO slightly fell short than one or some of the selected MOEAs. Otherwise, the proposed MOPSOs does not strictly dominate the rest of the MOEAs for test problems BNH, SRN, TNK, CTP1, CTP2, CTP6, CTP8, and CONSTR since box plots seem to show $I_{\varepsilon+}\left(A, B_{1-3}\right) > 0$ and $I_{\varepsilon+}\left(B_{1-3}, A\right) > 0$. The results in Table 8.11 also indicate the following conclusions: the proposed MOPSO performs equally well as NSGA-II for test problems SRN, CTP2, CTP7, and Welded beam; it also shares the same performance with GZHW for test problems TNK, OSY, CTP2, CTP6, and CTP8, and finally WTY performs equally well as the proposed MOPSO on test problem CTP1. In summary, we conclude that the proposed MOPSO performs equally well as the selected MOEAs.

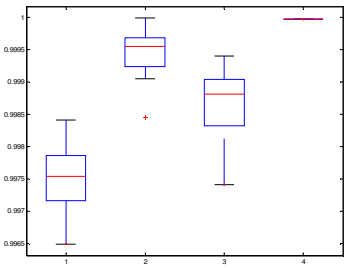
For qualitative comparison, the resulted Pareto fronts generated by all the algorithms from a single run given the same initial population are presented in Figure

Table 8.10 The distribution of I_H values tested using Mann-Whitney rank-sum Test. The table presents the z values and p-values with respect to the alternative hypothesis (i.e., p-value < α=0.05) for each pair of the proposed MOPSO and a selected constrained MOEAs. In each cell, both values are presented in a bracket: (z value, p-value). The distribution of the proposed MOPSO is significantly different than those selected constrained MOEAs unless stated.

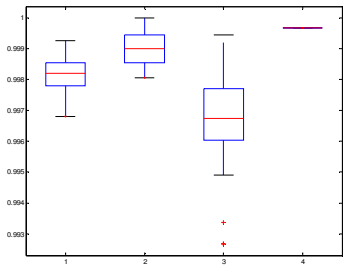
Test Functions	I_H (Proposed) AND		
	I_H (NSGA-II)	I_H (GZHW)	I_H (WTY)
BNH	(-8.6139, 7.1E-18)	(-7.7038, 1.3E-14)	(7.6529, 2.0E-14)
SRN	(-4.8715, 1.1E-06)	(3.7774, 1.6E-04)	(-7.1040, 1.2E-12)
TNK	(-6.6458, 3.0E-11)	(-3.2156, 1.3E-03)	(-7.1040, 1.2E-12)
OSY	(2.3612, 1.8E-02)	(0.3027, >0.05) *no difference*	(3.3644, 7.6E-04)
CTP1	(-7.7454, 9.5E-15)	(1.9751, 4.8E-02)	(7.0403, 1.9E-12)
CTP2	(-8.6072, 7.5E-18)	(6.0011, 2.0E-19)	(7.6529, 2.0E-14)
CTP3	(-6.0771, 1.2E-09)	(-8.6138, 7.1E-18)	(3.9771, 7.0E-05)
CTP4	(-8.6140, 7.1E-18)	(-8.4415, 3.1E-17)	(0.7633, >0.05) *no difference*
CTP5	(7.3900, 1.5E-13)	(6.5569, 5.5E-11)	(7.1040, 1.2E-12)
CTP6	(-8.5450, 1.2E-17)	(-4.7602, 1.9E-06)	(7.6529, 2.0E-14)
CTP7	(-3.2505, 1.2E-03)	(8.6138, 7.1E-18)	(3.3644, 7.7E-04)
CTP8	(-7.0904, 1.3E-12)	(3.9329, 8.4E-05)	(7.6529, 2.0E-14)
CONSTR	(-8.6140, 7.1E-18)	(-8.6138, 7.1E-18)	(7.6324, 2.3E-14)
Welded Beam	(-3.8073, 1.4E-04)	(6.6456, 3.0E-11)	(-6.6299, 3.6E-11)

8.10. For every test problem, four plots are presented and the labels (a)-(d) represent the following algorithms: the proposed MOPSO, NSGA-II, GZHW, and WTY respectively. Figure 8.10 shows the proposed MOPSO is able to produce equal quality Pareto fronts compared to the selected MOEAs for most of the test problems except for test problems OSY and Welded beam. In such cases, the proposed MOPSO produces worse Pareto fronts than NSGA-II and WTY due to the characteristic of the Pareto optimal region. The Pareto optimal front for OSY constitutes by five separate regions, in which there is at least one active constraint in each region. For Welded Beam, the difficulty lies on the nonlinear constraints and the curve of the Pareto front consists of extreme regions. That is

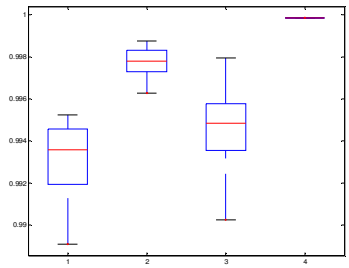
why it is difficult to obtain good distribution on those regions. However, for those two test problems, the proposed MOPSO shares the same performance with GZHW.



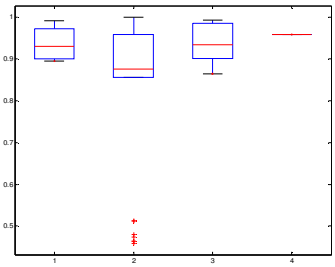
I_H values for BNH



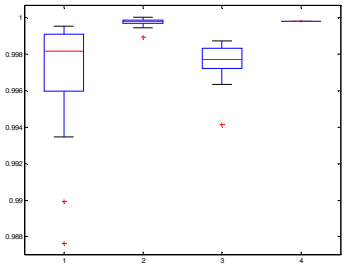
I_H values for SRN



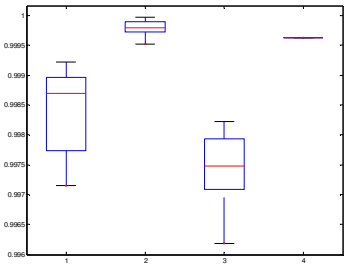
I_H values for TNK



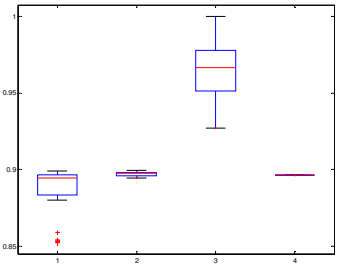
I_H values for OSY



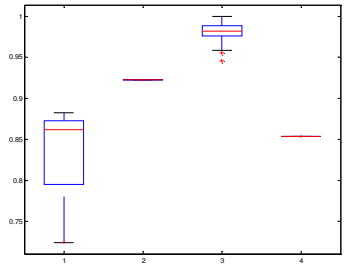
I_H values for CTP1



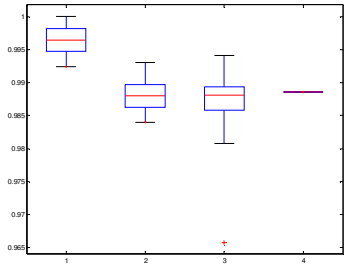
I_H values for CTP2



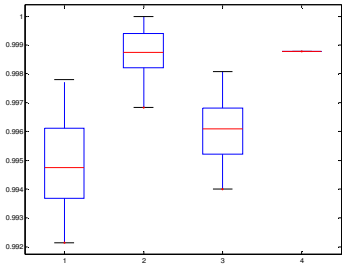
I_H values for CTP3



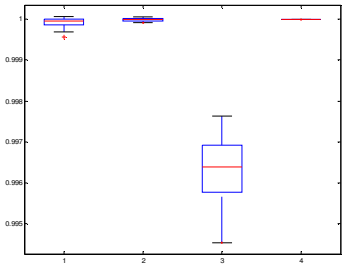
I_H values for CTP4



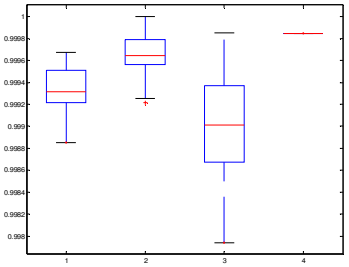
I_H values for CTP5



I_H values for CTP6



I_H values for CTP7



I_H values for CTP8

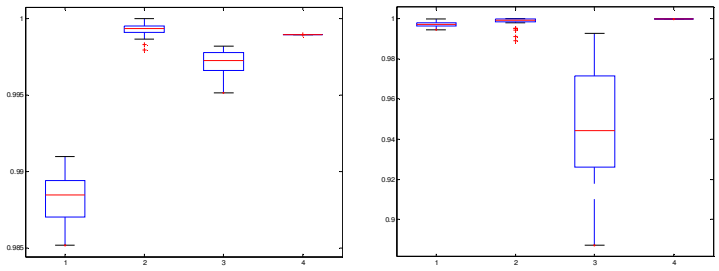
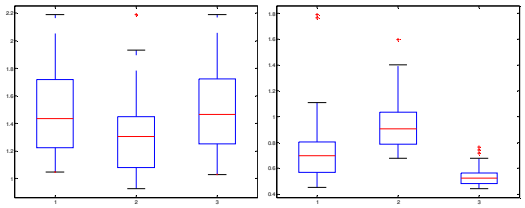


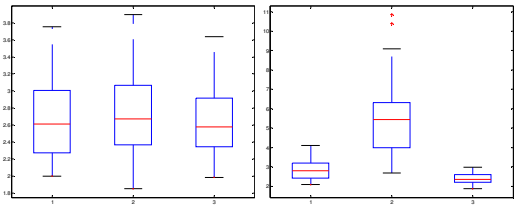
Figure 8.8 Box plot of hypervolume indicator (I_H values) for all test functions by algorithms 1-4 represented (in order): Proposed MOPSO, NSGA-II, GZHW, and WTY.

Table 8.11 The distribution of $I_{\varepsilon+}$ values tested using Mann-Whitney rank-sum Test. The table presents the z values and p-values with respect to the alternative hypothesis (i.e., p-value < α=0.05) for each pair of the proposed MOPSO and a selected constrained MOEAs. In each cell, both values are presented in a bracket: (z value, p-value). The proposed MOPSO is represented by *A*, and algorithms B_1, B_2, and B_3 are referred to as NSGA-II[31], GZHW[164] and WTY[166] respectively. The distribution of the proposed MOPSO is significantly difference than those selected constrained MOEAs unless stated.

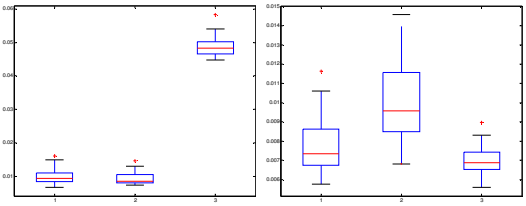
Test Functions	$I_{\varepsilon+}$ *(A,B1)* and $I_{\varepsilon+}$ *(B1,A)*	$I_{\varepsilon+}$ *(A,B2)* and $I_{\varepsilon+}$ *(B2,A)*	$I_{\varepsilon+}$ *(A,B3)* and $I_{\varepsilon+}$ *(B3,A)*
BNH	(5.8916, 3.8E-09)	(5.0637, 4.1E-07)	(6.6456, 3.0E-11)
SRN	(0.2905, >0.05) *no difference*	(-6.1281, 8.9E-10)	(2.2398, 2.5E-02)
TNK	(4.3244, 1.5E-05)	(0.3183, >0.05) *no difference*	(6.6456, 3.0E-11)
OSY	(-3.4078, 6.5E-04)	(0.2340, >0.05) *no difference*	(4.3984, 1.1E-05)
CTP1	(4.7532, 2.0E-06)	(-2.0772, 3.8E-02)	(0.9565, >0.05) *no difference*
CTP2	(0.1413, >0.05) *no difference*	(0.0933, >0.05) *no difference*	(6.1577, 7.4E-10)
CTP3	(6.6456, 3.1E-11)	(6.6456, 3.0E-11)	(6.6457, 3.0E-11)
CTP4	(6.6508, 2.9E-11)	(6.6456, 3.0E-11)	(6.6457, 3.0E-11)
CTP5	(-6.4386, 1.2E-10)	(-6.6012, 4.1E-11)	(-6.4386, 1.2E-10)
CTP6	(5.6994, 1.2E-08)	(-1.2345, >0.05) *no difference*	(6.4978, 8.2E-11)
CTP7	(0.2957, >0.05) *no difference*	(4.4279, 9.5E-06)	(6.6456, 3.0E-11)
CTP8	(4.3984, 1.1E-05)	(-1.7224, >0.05) *no difference*	(2.4320, 1.5E-02)
CONSTR	(5.4333, 5.5E-08)	(2.8016, 5.1E-03)	(6.2095, 5.3E-10)
Welded Beam	(-1.5154, >0.05) *no difference*	(-6.6456, 3.0E-11)	(5.5072, 3.7E-08)



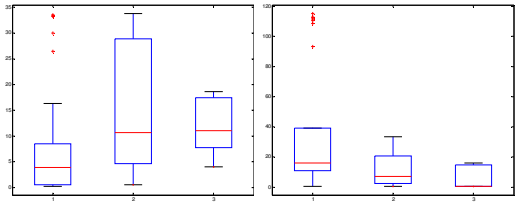
$I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$ for BNH



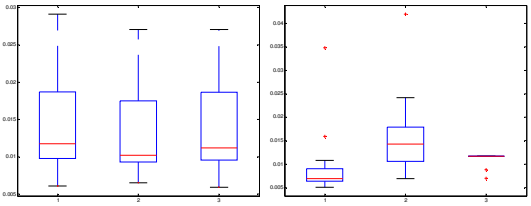
$I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$ for SRN



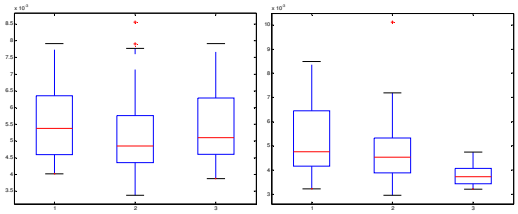
$I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$ for TNK



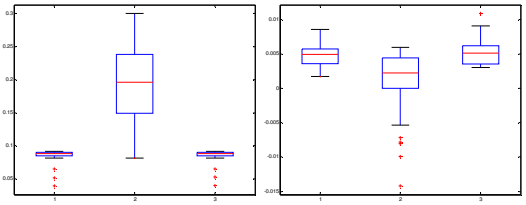
$I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$ for OSY



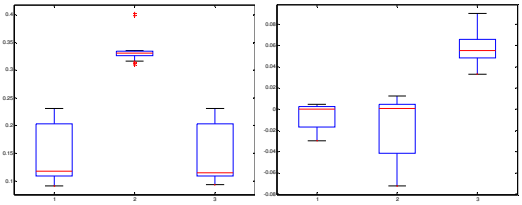
$I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$ for CTP1



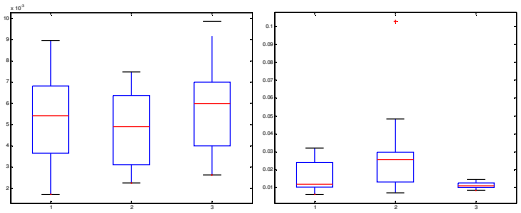
$I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$ for CTP2



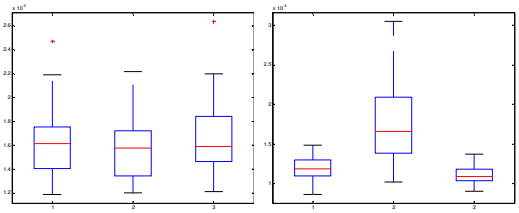
$I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$ for CTP3



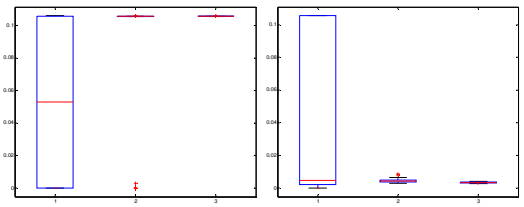
$I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$ for CTP4



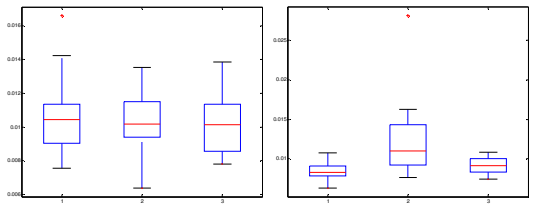
$I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$ for CTP5



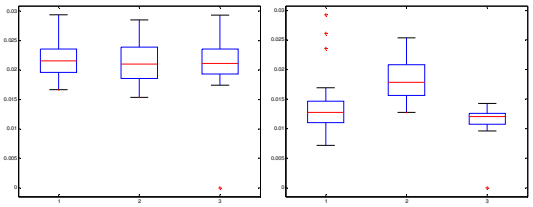
$I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$ for CTP6



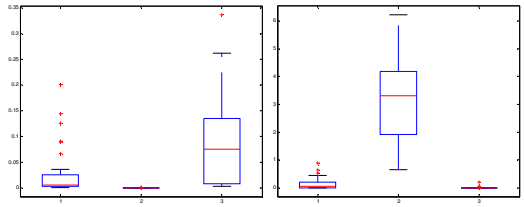
$I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$ for CTP7



$I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$ for CTP8

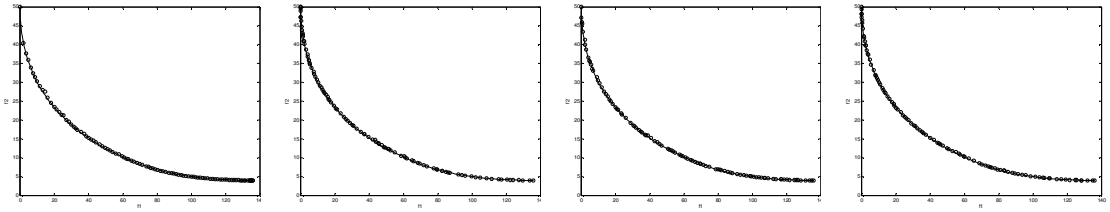


$I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$ for CONSTR



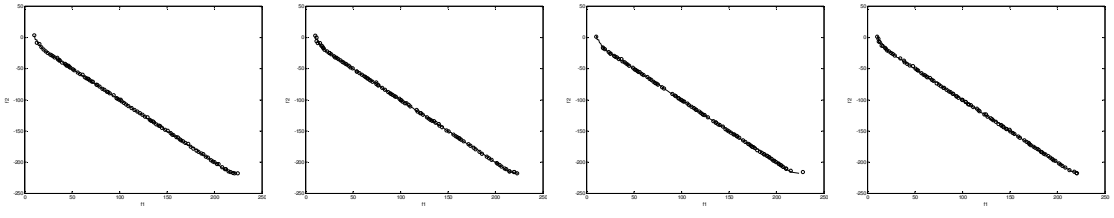
$I_{\varepsilon+}(A, B_{1-3})$ and $I_{\varepsilon+}(B_{1-3}, A)$ for Welded Beam

Figure 8.9 Box plot of additive binary epsilon indicator ($I_{\varepsilon+}$ values) for all test functions (algorithm *A* refers to the proposed MOPSO; algorithms B_{1-3} are referred to as NSGA-II, GZHW, and WTY, respectively).



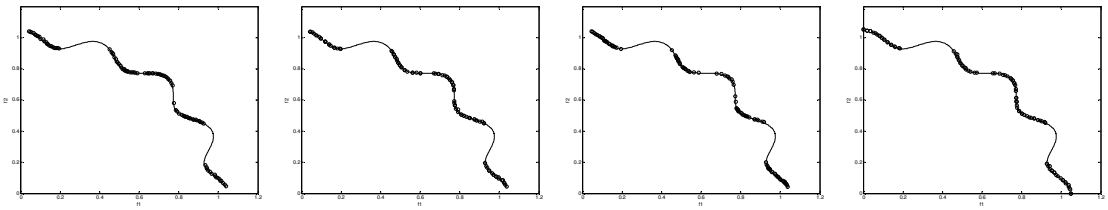
(a) (b) (c) (d)

Pareto fronts for BNH



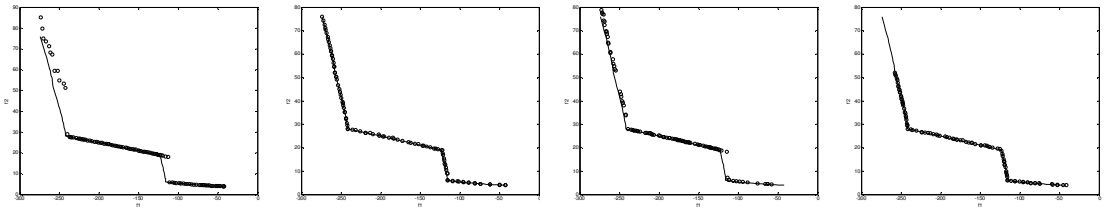
(a) (b) (c) (d)

Pareto fronts for SRN



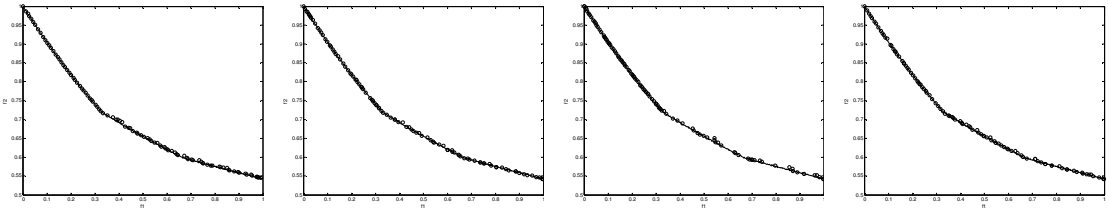
(a) (b) (c) (d)

Pareto fronts for TNK



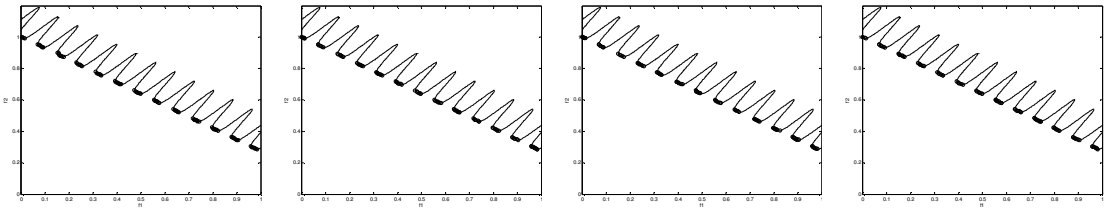
(a) (b) (c) (d)

Pareto fronts for OSY



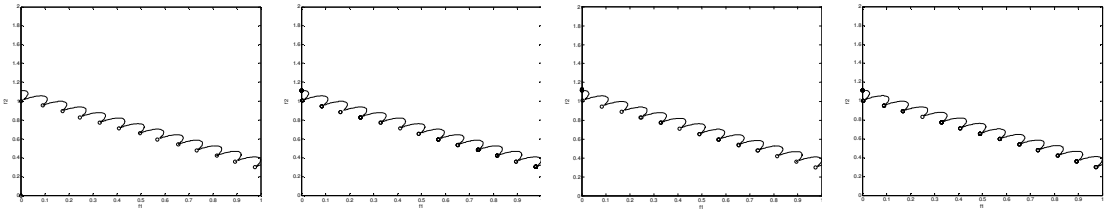
(a) (b) (c) (d)

Pareto fronts for CTP1



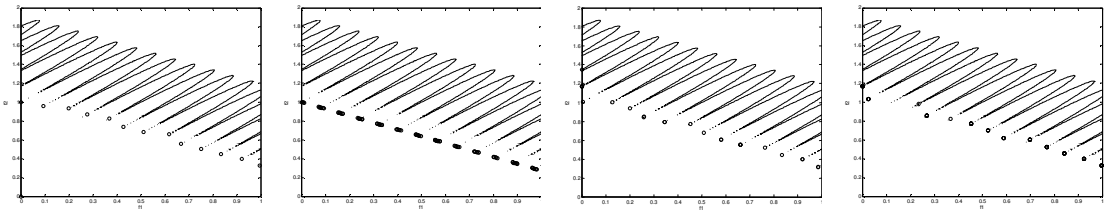
(a) (b) (c) (d)

Pareto fronts for CTP2



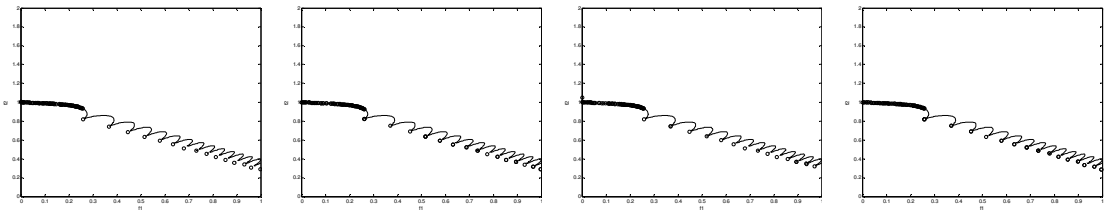
(a) (b) (c) (d)

Pareto fronts for CTP3



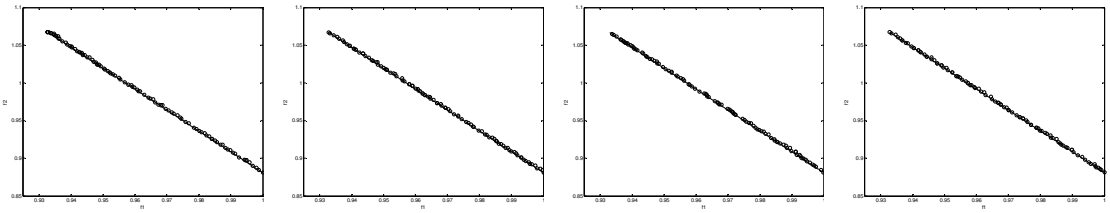
(a) (b) (c) (d)

Pareto fronts for CTP4



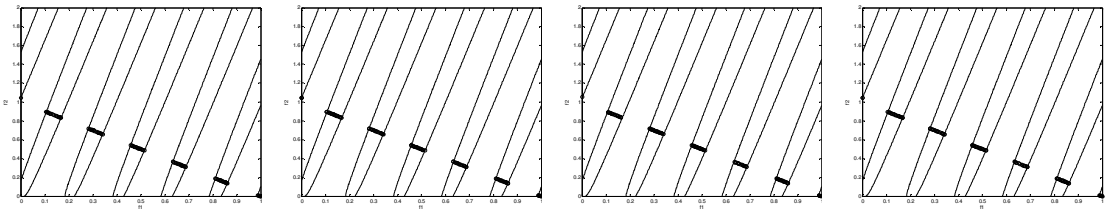
(a) (b) (c) (d)

Pareto fronts for CTP5



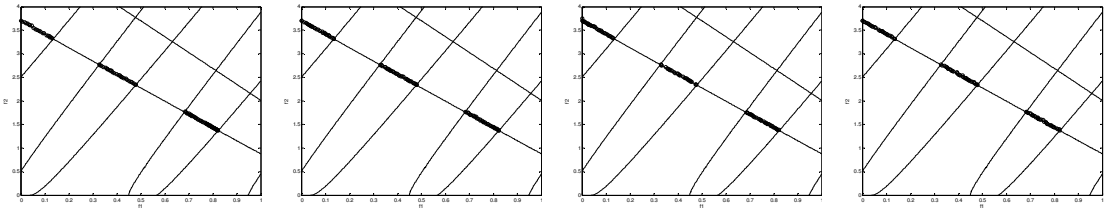
(a) (b) (c) (d)

Pareto fronts for CTP6



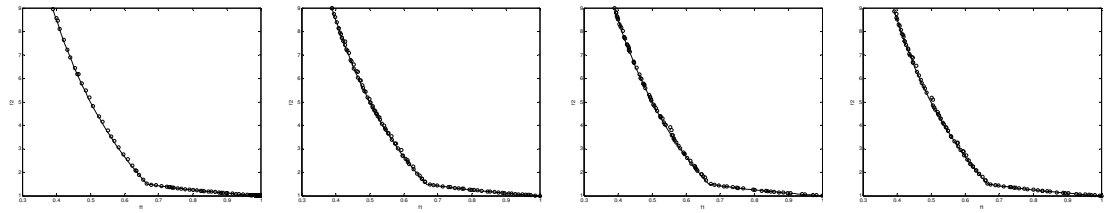
(a) (b) (c) (d)

Pareto fronts for CTP7



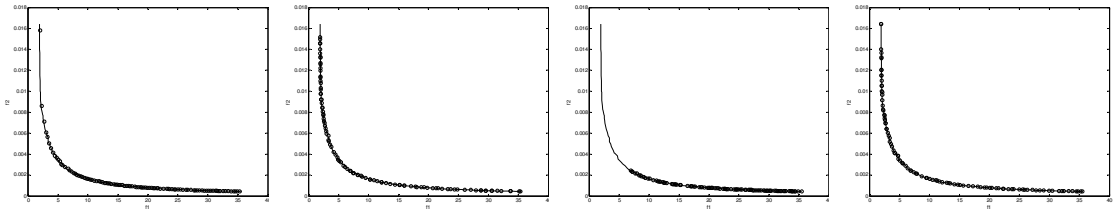
(a) (b) (c) (d)

Pareto fronts for CTP8



(a) (b) (c) (d)

Pareto fronts for CONSTR



(a) (b) (c) (d)

Pareto fronts for Welded Beam

Figure 8.10 Pareto fronts produced by the following algorithms a-d represented (in order): proposed MOPSO, NSGA-II, GZHW and WTY.

CHAPTER 9

CONCLUSION AND FUTURE WORKS

Many real world problems are often multiobjective in nature. Most of them are subjected by a set of constraints. In this thesis, the objective is to develop multiobjective particle swarm optimization algorithms to deal with both unconstrained and constrained multiobjective optimization problems. In addition to the objective, the inherited mechanisms of particle swarm optimization are studied to exploit the key mechanism to solve for unconstrained multiobejctive optimization problems and handle constraints.

9.1 Dynamic Population Size and Multiple-swarm Concepts

Recently, various multiobjective particle swarm optimization (MOPSO) algorithms have been developed to efficiently and effectively solve unconstrained multiobjective optimization problems. However, the existing MOPSO designs generally adopt the need to "estimate" a fixed population size sufficiently to explore the search space without incurring excessive computational complexity. Existing works of MOEAs that adopted the idea of adaptively adjust the population size during the course of finding the optimal Parato front have inspired the designs of the two MOPSO algorithms that proposed in this study. Coupled with the idea of dynamic population size, the two proposed MOPSO algorithms also integrate the multiple swarms concept to exploit the "swarm-like" characteristic of the PSO and to enhance their potential to achieve better

performance. Although both proposed algorithms integrate dynamic population size and multiple swarms concepts, the design perspectives are different.

In the first proposed algorithm, the number of particles or swarm population size is dynamically changed but the number of swarms is user defined. Hence, the number of particles in each swarm (or swarm size) depends on the swarm population size. The basic skeleton of this algorithm, multiobjective particle swarm optimization algorithm (DMOPSO), is based on the design of cMOPSO [128]. Additionally, three proposed features are incorporated into the algorithm: 1) a cell-based rank density estimation scheme to quickly update the location of the new particles in the objective space and to provide easy access to the rank and density information of the particles; 2) a population growing strategy that adaptively grows new particles with enhanced exploration and exploitation capabilities; 3) a population declining strategy to balance and control the dynamic population size; and 4) adaptive local archives to improve the selection of group leaders to produce a better distributed Pareto front associated with each swarm. A comparative study of DMOPSO with five selected state-of-the-art MOPSOs on six benchmark test problems is presented. The results of applying two performance metrics clearly indicate that DMOPSO is highly competitive and even outperforms most of the selected MOPSOs. In addition, qualitative results also show that DMOPSO has the ability to produce relatively better Pareto fronts compared to most of the selected MOPSOs for all six benchmark test functions In fact, dynamic population strategy has contributed in improved performance. The reasons are as follows: First, dynamic population strategy provides some flexibility in preserving good particles and removing those that will not contribute to the search process for the following iterations. Second, the design

guarantees to grow new potential particles that will either improve the search process or land on unexplored regions to discover better solutions. This will speed the process in finding better solutions and indirectly save computational cost. Third, multiple swarms approach provides some degrees of local search. This greatly enhances the quality of convergence toward optimal Pareto front. To avoid the excessive use of local search, adaptive local archive is incorporated to maintain the diversity within each swarm, at least in a local sense.

However, there are two disadvantages in DMOPSO. Clustering algorithm is applied to group the leaders in the archive according to the predefined number of swarms. This adds additional computational complexity to the algorithm, especially if the number of swarms is set too high. Another weakness of DMOPSO is the parameter settings and dealing with the question of how to optimally choose the parameters. We suggest fixing some of the parameters such as $lnp = 1$; $rud = 0.7$; $rld = 0.02$ and $ppv = 10$. For grid scale, K, we suggest starting at 100 first and then tuning the value up or down depending on the resolution of the resulting Pareto front needed. The setting of parameter, K_a, depends on the number of sub-swarms. If the number of swarms is high, then parameter, K_a, can be tuned down and vice versa. Lastly, parameters r_b, unp and selection ratio are interdependent. Currently, these parameters are selected *ad hoc*.

Due to the disadvantages of DMOPSO discussed above, a new multiobjective particle swarm optimization, called DSMOPSO is proposed. This algorithm, however, dynamically adjusts the number of swarms instead of the swarm population size, and fixes the swarm size for each swarm. The design of this algorithm involves three main contributions. First, the swarm growing and declining strategies are developed to

dynamically grow new potential swarms and to remove swarms with least contribution to the search process. These strategies promote diversity by placing new swarms into unexplored areas and by eliminating swarms that reside in crowded regions. Second, PSO updating rule is modified to improve the interaction among swarms and particles within a swarm. Third, the objective space compression and expansion strategy is proposed to allow adjustment of the size of the objective space to ensure the swarms progressively find the true Pareto front. Experiments show DSMOPSO is competitive in terms of performance, compared to selected MOPSOs, in both qualitative and quantitative measures for the selected test functions. In the study investigating the computational cost exerted by DSMOPSO, it appears that DSMOPSO demands less computational cost for test problems with disconnected Pareto front or with multiple local optima. In a future study, which types of problem characteristics work best for DSMOPSO in terms of performance and computational cost will be further investigated. Lastly, sensitivity analysis is conducted to study the impact of the tuning parameters on DSMOPSO. From the results, we have recommended various parameter settings that will deliver good performance for the selected test functions. There are advantages of DSMOPSO over the first proposed algorithm, DMOPSO. Only two user-defined parameters are in the swarm growing and declining strategies as opposed to six user-defined parameters in DMOPSO. With lesser user-defined parameters, this reduces the difficulty of tuning the parameters and the dependency among the parameters, which reduces the impact of the tuning parameters on the algorithm's performance. In addition, the objective space compression and expansion strategy will reduce the dependency on setting the grid scale parameter, K. With multiple swarms concept directly applies to the search process, both local and

global searches are encouraged during process of searching for the optimal Pareto front. Hence, clustering algorithm is no longer needed. Despite of the advantages, DSMOPSO has its limitations. Through observation, one of its limitation is the search progression is slower, which render larger computational cost. This may due to two possible reasons. The growth rate for number of swarms is not high enough and the lacking of good strategy to enhance the communication among and within swarms. Another problem is the need to effectively trigger the objective space compression and expansion routine in order to reduce its frequency.

In the near future works, study to deal with the disadvantages of DSMOPSO is highly desired and investigation on the performance of DSMOPSO for test functions more than two objectives is required. For DMOPSO, It will be interesting to study how well DMOPSO will handle the combinatorial optimization problems since several publications have proved successful in applying PSO to solve for combinatorial optimization problems like multiobjective knapsack or TSP [149-151].

9.2 Constraint Handling

For constraint optimization, the main challenges are to optimize the objective function(s) and simultaneously handle constraints. The design of constrained MOPSO is achieved in two steps. First the constrained PSO with key design elements is proposed for COPs then with the design elements, it is extended to a MOPSO to solve for CMOPs. This proposed constrained PSO adopts a multiobjective constraint handling technique, in which the COP is converted into an unconstrained bi-objective optimization problem. It incorporates the following design features: 1) separate procedures to update the infeasible

and feasible personal best in the personal best archive in order to guide the infeasible particles towards the feasible regions while promote search for better solutions; 2) an infeasible global best archive is adopted to make use of the infeasible nondominated solutions for searching possible isolated feasible regions or a very small feasible region while the feasible global best archive aims to guide the particles to find better solutions; and 3) the adjustment of the accelerated constants in the PSO equation is based on the number of feasible personal best in the personal best archive and the constraint violations of personal best and global best. The adjustment will influence the search process either to find more feasible solutions (particles) or to search for better solutions; and the frequency of applying the mutation operators are based on the feasibility ratio of the particles' personal best. This feasibility ratio is exploited to encourage more exploration characteristic to search possible feasible regions when there are few feasible particles' personal best, while reduce the exploration rate when most of the particles' personal best are feasible to support convergence toward Pareto optimal front. In addition, a mutation operator with the mutated range covered is narrowed overtime to encourage global search in early iterations and fine tune local search in later iterations. From the simulation study, the proposed constrained PSO is capable to obtain quality feasible solutions for most of the test problems, while the performance achieved is competitive when compared with selected state-of-the-art approaches. In our future work, further improvement is considered to improve the solution quality and to solve for those problems that occasionally do not find feasible optima, such as test problem g17.

For the proposed constrained MOPSO, same design as the constrained PSO is incorporated except that the mutation operator is modified. In this design, both uniform

and Gaussian mutation operators are used to encourage local and global search. Furthermore, the mutation rate for both mutation operators is adaptively determined by the feasibility ratio of the particles' personal best, in which the frequency of applying the mutation operators depends on the number of feasible personal best in the archive. A comparative study of the proposed MOPSO and three state-of-the-art constrained MOEAs on 14 benchmark test problems are presented. The simulation results show the proposed constrained MOPSO is highly competitive and able to obtain quality Pareto fronts for most of the test problems. However, the proposed constrained MOPSO is still fail in solving test problems OSY and Welded Beam by observing the simulation results. Several suggestions for future works: improve the diversity mechanism in the design elements, apply the proposed constrained MOPSO to other CMOPs, e.g., problems with equality constraints, and incorporate dynamic population concept or multiple swarms approach in the proposed constrained MOPSO.

BIBIOGRAPHY

[1] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*, John Wiley & Sons, Ltd, England, 2001.

[2] C.A. Coello Coello, "Recent trends in evolutionary multiobjective optimization," Ajith Abraham, Lakhimi Jain, Robert Goldberg (Editors), *Evolutionary Multiobjective Optimization Theoretical Advances and Application*, Springer-Verlag London Limited, USA, 2005.

[3] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by simulated annealing," *Science*, Vol 220, No. 4598, pp. 671-680, 1983.

[4] D. Cvijovic and J. Klinowski, "Taboo search: An approach to the multiple minima problem," *Science*, Vol. 267, No. 5198, pp. 664-666, 1995.

[5] Joshua D. Knowles, "Local-search and hybrid evolutionary algorithms for Pareto optimization," PhD thesis, The University of Reading, Reading, UK, January 2002.

[6] C.A. Coello Coello, "Evolutionary multi-objective optimization: A historical view of the field," *IEEE Computational Intelligence Magazine*, February, pp.29-36, 2006.

[7] K.C. Tan, E.F. Khor, and T.H. Lee, *Multiobjective Evolutionary Algorithms and Application*, Springer-Verlag London Limited, 2005.

[8] A. Abraham and L. Jain, "Evolutionary multiobjective optimization," Ajith Abraham, Lakhimi Jain, Robert Goldberg (Editors), *Evolutionary Multiobjective Optimization Theoretical Advances and Application*, Springer-Verlag London Limited, USA, 2005.

[9] *M. Schniederjans,* Goal Programming: Methodology and Applications*, Kluwer Academic Publishers, Mar 1995.*

[10] *Saul I. Gass,* Linear Programming: Methods and Applications*, Fifth Edition, Dover Publications, 2003.*

[11] E. Papadopoulou and D.T. Lee, "The Min-Max Voronoi diagram of polygons and applications in VLSI manufacturing," *Proceedings of 13th Annual International*

Symposium on Algorithms and Computation*, Vancouver, Canada, pp. 511-522, 2002.*

[12] C.L. Hwang and A.S.M. Masud, *Multiple Objective Decision Making – Methods and Applications*, Springer-Verlag, NY, 1979.

[13] Haiming Lu, "State-of-the-art multiobjective evolutionary algorithms - Pareto ranking, density estimation and dynamic population," PhD Thesis, Oklahoma State University, Stillwater, Oklahoma, August 2002.

[14] A. Charnes and W.W Cooper, *Management models and industrial applications of linear programmin*g, Wiley, New York, 1961.

[15] Y. Ijiri, *Management Goals and Accounting for Control*, Rand-McNally, Chicago, 1965.

[16] M. Tamiz and D.F. Jones, "Expanding the flexibility of goal programming via preference modelling techniques", *Omega - The International Journal of Management Science*, Vol. 23, No 1, pp. 41-48, 1995.

[17] D.F. Jones and M. Tamiz Goal programming in the period 1990-2000, *in Multiple Criteria Optimization: State of the art annotated bibliographic surveys*, M. Ehrgott and X.Gandibleux (Eds.), pp. 129-170. Kluwer, 2002.

[18] C. Romero, *Handbook of critical issues in goal programming*, Pergamon Press, Oxford, 1991.

[19] C.A. Coello Coello and A.D. Christiansen, "Two new GA-based methods for multiobjective optimization," *Civil Engineering Systems*, Vol. 15, No. 3, pp. 207-243, 1998.

[20] N.A. Barricelli, "Esempi numerici di processi di evoluzione, " *Methodos*, pp. 45-68, 1954.

[21] T. Bäck, *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, Inc., New York, 1996.

[22] D.E. Goldberg, *The Design of Innovation Lessons from and for Competent Genetic Algorithm*, Kluwer Academic Publishers, USA, 2002.

[23] W.B. Langdon, R. Poli, *Foundations of Genetic Programming*, Springer-Verlag, 2002.

[24] J.D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithm," *Genetic Algorithms and Their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pp. 93-100, 1985.

[25] D.E. Goldberg, *Genetic Algorithm in Search, Optimization and Machine Learning*, Addison-Westley Publishing Company, Reading, Massachusetts, 1989.

[26] C. Fonseca and P. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization," *Proceedings of 5th International Conference in Genetic Algorithms*, Urbana-Champaign, IL, pp. 416-423, 1993.

[27] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithm: A comparative case study and the strength pareto approach," *IEEE Transactions on Evolutionary Computations*, Vol.3, No. 4, pp. 257-271, Nov, 1999.

[28] H. Lu and G.G. Yen, "Rank-density-based multiobjective genetic algorithm and benchmark test function study," *IEEE Transactions on Evolutionary Computations*, Vol. 7, No. 4, pp. 325- 343, 2003.

[29] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: improving the strength Pareto evolutionary algorithm," Swiss Federal Institute of Technology, Zurich, Switzerland, Technical Report, TIK-Rep. 103, 2001.

[30] J.D. Knowles and D.W. Corne, "Approximating the nondominated front using the Pareto archived evolution strategy," *Evolutionary Computation,* Vol. 8, No. 2, pp. 149-172, 2000.

[31] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," ," *IEEE Transactions on Evolutionary Computations*, Vol. 6, no. 2, pp. 182-197, 2002.

[32] D.W. Corne, J.D. Knowles, and M.J. Oates, "The pareto envelope-based selection algorithm for multiobjective optimization," M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, and H-P. Schwefel (Editors), *Proceedings of the Parallel Problem Solving from Nature VI Conference,* Paris, France, Vol. 1917, pp. 839-848, 2000.

[33] C.A. Coello Coello and G. Toscazo Pulido, "A micro-genetic algorithm for multiobjective optimization," E. Zitzler, K. Deb, L. Thiele, C.A. Coello Coello, and D. Corne (Editors), *Proceedings of the First International Conference on Evolutionary Multi-criterion Optimization*, Zurich, Switzerland, Vol. 1993, pp. 126-140, 2001.

[34] J.D. Schaffer, "Some experiments in machine learning using vector evaluated genetic algorithm," Ph.D. Thesis, Vanderbilt University, 1984.

[35] F. Kursawe, "A variant of evolution strategies for vector optimization," *Parallel Problem Solving in Nature I*, Dortmund, Germany, pp. 193-197, 1990.

[36] C. Poloni, A. Giurgevich, L. Onesti, and V. Pediroda, "Hybridization of a multiobjective genetic algorithm, a neural network and a classical optimizer for complex design problem in fluid dynamic," *Computer Methods in Applied Mechanics and Engineering*, Vol. 186, No. 2-4, pp. 403-420, 2000.

[37] C.M. Fonseca and P.J, Fleming, "An overview of evolutionary algorithms in multiobjective optimization," *Evolutionary Computation Journal*, Vol. 3, No. 1, pp. 1-16, 1995.

[38] K. Deb, "Multi-objective genetic algorithms: Problem difficulties and construction of test problems," *Evolutionary Computation Journal*, Vol. 7, No.3, pp. 205-230, 1999.

[39] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary Computation Journal*, Vol. 8, No.2, pp. 173-195, 2000.

[40] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable test problems for evolutionary multiobjective optimization," *Evolutionary Computation Based Multi-Criteria Optimization: Theoretical Advances and Applications*. Springer, pp. 105-145, 2005.

[41] T. Okabe, Y. Jin, M. Olhofer, and B. Sendhoff, "On test functions for evolutionary multi-objective optimization," *Proceedings of Parallel Problem Solving from Nature-VIII*, Birmingham, UK, Vol. 3242, pp. 792-802, 2004.

[42] S. Huband, P. Hingston, L. Barone, and L. While, "A review of multiobjective test problems and a scalable test problem toolkit," *IEEE Transactions on Evolutionary Computatio*n, Vol. 10, No. 5, pp. 477-506,Oct, 2006.

[43] D.A. Van Veldhuizen and G.B. Lamount, "Multiobjective evolutionary algorithm research: A history and analysis," Department of Electrical Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Technical Report TR-98-03, 1998.

[44] J.R. Schott, "Fault tolerant design using single and multicriteria genetic algorithm optimization," Master Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, May, 1995.

[45] D.A. Van Veldhuizen, "Multiobjective evolutionary algorithms: Classifications, analyses, and new innovations," Ph.D. Thesis, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH, May 1999.

[46] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca and V.G. da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review,"

IEEE Transactions on Evolutionary Computation, Vol. 7, No. 2, pp.117-132, April 2003.

[47] E. Zitler, "Evolutionary Algorithms for multiobjective optimization: methods and applications," Ph.D dissertation, Shaker Verlag, Aachen, Germany, 1999.

[48] J. Kennedy, and R. C. Eberhart, Swarm Intelligence, ISBN 1-55860-595-9, Academic Press, 2001.

[49] M. Dorigo, *Optimization, Learning and Natural Algorithms*, PhD thesis, Politecnico di Milano, Italy, 1992.

[50] J. Kennedy and R.C. Eberhart, "Particle swarm optimization," *Proceedings of IEEE International Conference of Neural Networks*, Perth, Australia, pp. 1942-1948, 1995.

[51] M. Dorigo and L.M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, pp. 53–66, 1997.

[52] V. Maniezzo, A.Colorni, and M.Dorigo, "The ant system applied to the quadratic assignment problem," *Tech. Rep. IRIDIA/94-28*, Université Libre de Bruxelles, Belgium, 1994.

[53] H. Weimerskirch. (December, August,16). Bird flight explained. *BBC News World Edition*. Available: http://news.bbc.co.uk/2/hi/science/nature/1608251.stm

[54] C.W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model, in computer graphics," *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, Vol. 21, No.4, pp. 25-34, 1987.

[55] C.W. Reynolds. Boids: Background and update. Available: http://www.red3d.com/cwr/boids/

[56] F. Heppner, and U. Grenander, "A stochastic nonlinear model for coordinated bird flocks**,"** The Ubiquity of Chaos, ed. S. Krasner, AAAS Publications, Washington, DC, 1990.

[57] H. Lorek and M. White, "Parallel Bird Flocking Simulation," *Proceedings of BCS Conference on Parallel Processing for Graphics and Scientific Visualization*, Edinburgh, May 1993.

[58] J. Toner and Y. Tu, "Flocks, herds, and schools: A quantitative theory of flocking," *Physical Review E*, Vol. 58, No. 4, pp. 4828-4858, 1999.

[59] L. Spector, J. Klein, C. Perry, M. Feinstein, "Emergence of Collective Behavior in Evolving Populations of Flying Agents," *Genetic Programming and Evolvable Machines*, Vol. 6, No. 1, pp. 111-125, 2005.

[60] H. G. Tanner, A. Jadbabaie and G. J. Pappas, "Flocking Agents with Varying Interconnection Topology", *Automatica*, 2004.

[61] Paul Pomeroy, An Introduction to Particle Swarm Optimization, *AdpativeView.com*, March, 2003.

[62] Y. Shi and R.C. Eberhart, "A modified particle swarm optimizer," *Proceedings of IEEE International Conference on Evolutionary Computatio*n, Anchorage, Alaska, pp. 303-308, 1997.

[63] S. Mikki and A. Kishk, "Improved particle swarm optimization technique using hard boundary conditions," *Microwave and Optical Technology Letters*, Vol. 46, No.5, pp.422-426, 2005.

[64] L. Zhang, H. Yu, and S. Hu, "A new approach to improve particle swarm optimization," *Proceedings of the Genetic and Evolutionary Computation Conference,* Chicago, Il, Vol. 2723, pp. 134-142, 2003.

[65] Y. Shi and R.C. Eberhart, "Empirical study of particle swarm optimization," *Proceedings of International Congress on Evolutionary Computation*, Piscataway, NJ, Vol. 3, pp. 101-106, 1999.

[66] Z. Qin, F. Yu, Z.W. Shi, and Y. Wang, "Adaptive inertia weight particle swarm optimization," *Proceedings of Artificial Intelligence and Soft Computing,* Zakopane, Poland, Vol. 4029, pp. 450-459, 2006.

[67] A. Ratnaweera, S. K. Halgamuge, and H. C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration constants," *IEEE Transactions on Evolutionary Computations*, Vol. 8, No. 3, pp. 240- 255, 2004.

[68] Z.H. Cui, J.C. Zeng, and G.J. Sun, "Adaptive velocity threshold particle swarm optimization," *Proceedings of Rough Sets and Knowledge Technology,* Chongquing, China, Vol. 4062, pp. 327-332, 2006.

[69] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, Vol. 6, pp. 58-73, 2002.

[70] J. Kennedy, "Bare bones particle swarms," *Proceedings of Swarm Intelligence Symposium*, Indianapolis, Indiana, pp. 80-87, 2003.

[71] R. A. Krohling, "Gaussian particle swarm with jumps," *Proceedings of Congress on Evolutionary Computation*, Vol. 2, Edinburgh, UK, pp. 1226- 1231, 2005.

[72] N. Higashi and H. Iba, "Particle swarm optimization with Gaussian mutation," *Proceedings of Swarm Intelligence Symposium*, Indianapolis, Indiana, pp. 72-79, 2003.

[73] S. Das, A. Konar, U. K. Chakraborty, "Improving particle swarm optimization with differentially perturbed velocity," *Proceedings of Genetic and Evolutionary Computation Conference*, Washington, DC, pp. 177-184, 2005.

[74] P.N. Suganthan, "Particle swarm optimiser with neighborhood operator," *Proceedings of Congress on Evolutionary Computation*, Washington, DC, pp. 1958-1962, 1999.

[75] J. Kennedy and R. Mendes, "Topological structure and particle swarm performance," *Proceedings of the Fourth Congress on Evolutionary Computation*, Honolulu, Hawaii, pp. 1671–1676, 2002.

[76] R. Mendes, J. Kennedy, J. Neves, "Watch thy neighbor or how the swarm can learn from its environment," *Proceedings of Swarm Intelligence Symposium*, Indianapolis, Indiana, pp. 88- 93, 2003.

[77] James Kennedy and R. Mendes, "Neighborhood topologies in fully informed and best-of-neighborhood particle swarm," *IEEE Transaction on Systems, Man and Cybernetics–Part C: Applications and Reviews* , Vol. 36, No. 4, pp. 515-519, 2006.

[78] E.S. Peer, F. van den Bergh, A.P. Engelbrecht, "Using neighborhoods with guaranteed convergence PSO," *Proceedings of Swarm Intelligence Symposium*, Indianapolis, Indiana, pp. 235- 242, 2003.

[79] A.S. Mohais, R. Mendes, C. Ward, and C. Posthoff, "Neighborhood Re-structuring in Particle Swarm Optimization," *Proceedings of 18th Australian Joint Conference on Artificial Intelligence,* Vol. 3809, pp. 776-785, 2005.

[80] R. Brits, A. Engelbrecht, and F. van den Bergh, "Scalability of Niche PSO," *Proceedings of Swarm Intelligence Symposium*, Indianapolis, IN, pp. 228-234, 2003.

[81] S. Bird and X. Li, "Enhancing the robustness of a speciation-based PSO," *Proceedings of Congress on Evolutionary Computation*, Vancouver, Canada, pp. 3185-3192, 2006.

[82] X. Li, "Adaptively choosing neighborhood bests using species in a particle swarm optimizer for multimodal function optimization," *Proceedings of Genetic and Evolutionary Computation Conference*, Seattle, WA, pp.105-116, 2004.

[83] M. Iwamatsu, "Multi-species particle swarm optimizer for multimodal function optimization," *IEICE Transactions on Information and Systems*, Vol. E89D, No. 3, pp. 1181-1187, 2006.

[84] M. Iwamatsu, "Locating all the global minima using multi-species particle swarm optimizer: the inertial weight and the constriction factor variants," *Proceedings of Congress on Evolutionary Computation*, Vancouver, Canada, pp. 3158-3164, 2006.

[85] A. Passaro and A. Starita, "Clustering particles for multimodal function optimization," *Proceedings of ECAI Workshop on Evolutionary Computation*, Riva del Garda, Italy, pp. 124-131, 2006.

[86] J.H. Seo, C.H. Im, C.G. Heo, J.K. Kim, H.K. Jung, and C.C. Lee, "Multimodal function optimization based on particle swarm optimization," *IEEE Transactions on Magnetics*, Vol. 42, No. 4, pp. 244-252, 2006.

[87] J. Zhang, D.S, Huang, T.M. Lok, and M.R. Lyu, "A novel adaptive sequential niche technique for multimodal function optimization," *Neurocomputing*, Vol. 69, pp. 2396-2401, 2006.

[88] T. Blackwell and J. Branke, "Multiswarms, exclusion, and anti-convergence in dynamic environments," *IEEE Transactions on Evolutionary Computation*, Vol. 10, No. 4, pp. 459-472, 2006.

[89] D. Parrott and X. Li, "Locating and tracking multiple dynamic optima by a particle swarm model using speciation," *IEEE Transactions on Evolutionary Computations*, Vol. 10, No. 4, pp. 440-458, 2006.

[90] J. Kennedy, "Stereotyping: Improving particle swarm performance with cluster analysis," *Proceedings of Congress on Evolutionary Computation*, San Diego, CA, pp. 1507-1512, 2000.

[91] F. van den Bergh and A.P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computations*, Vol. 8, No. 3, pp. 225-239, 2004.

[92] G. Chen and J. Yu, "Two sub-swarms particle swarm optimization algorithm," *Proceeding of International Conference on Natural Computation*, Changsha, China, pp. 515-524, 2005.

[93] B. Niu, Y. Zhu, and X. He, "Multi-population cooperative particle swarm optimization," *Proceeding of European Conference on Artificial Life*, Canterbury, UK, pp. 874-883, 2005.

[94] M. El-Abd and M. Kamel, "Information exchange in multiple cooperating swarms," *Proceedings of Swarm Intelligence Symposium*, Pasadena, CA, pp. 138-142, 2005.

[95] M. El-Abd and M.S. Kamel, "On the convergence of information exchange methods in multiple cooperating swarms," *Proceedings of Congress on Evolutionary Computation*, Vancouver, Canada, pp. 3797-3801, 2006.

[96] G.G. Yen and M. Daneshyari, "Diversity-based information exchange among multiple swarms in particle swarm optimization," *Proceedings of Congress on Evolutionary Computation*, Vancouver, Canada, pp. 1686-1693, 2006.

[97] B. Niu, Y.L. Zhu, K.Y. Hu, S.F. Li, X.X. He, "A novel particle swarm optimizer using optimal foraging theory," *Proceedings of Computational Intelligence and Bioinformatics, Part 3,* Kunming, China , Vol. 4115, pp. 61-71, 2006.

[98] J. Sun, B. Feng, and W. Xu, "Particle swarm optimization with particles having quantum behavior," *Proceedings of Congress on Evolutionary Computation*, Portland, OR, pp. 325-331, 2004.

[99] G. Yang and R. Zhang, "An emotional particle swarm optimization," *Proceedings of First International Conference of Advances in Natural Computation, Part 3,* Changsha, China, Vol. 3612, pp. 553-561, 2005.

[100] Z. Wang, G.L. Durst, R.C. Eberhart, D.B. Boyd, and Z. Ben Miled, "Particle swarm optimization and neural network application for QSAR," *Proceedings of Third IEEE International Workshop on High Performance Computational Biology*, Santa Fe, NM, in HiCOMB 2004 Online proceeding, 2004.

[101] M. P. Wachowiak, R. Smolikova, Y. Zheng, J. M. Zurada, and A. S. Elmaghraby, "An approach to multimodal biomedical image registration utilizing particle swarm optimization," IEEE Transaction on Evolutionary Computation, Vol. 8, No. 3, pp. 289-301, June 2004.

[102] J. Tillett, S.J. Yang, R. Rao, and F. Sahin, "Application of particle swarm techniques in sensor network configuration," *Proceedings of SPIE International Society for Optical Engineering*, Vol. 5796, pp. 363-373, 2005.

[103] H. Liu, S. Sun, and A. Abraham, "Particle swarm approach to scheduling work-flow applications in distributed data-intensive computing environments," *Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications*, Jinan. Shandong, China, Vol. 2, pp. 661-666, 2006.

[104] K.C. Lee and J.Y. Jhang, "Application of particle swarm algorithm to the optimization of unequally spaced antenna arrays," *Journal of Electromagnetic Waves and Applications*, Vol. 20, No. 14, pp. 2001-2012, 2006.

[105] X. Hu, R.C. Eberhart and Y. Shi, "Particle swarm with extended memory for multiobjective optimization," *Proceedings of Swarm Intelligence Symposium*, Indianapolis, IN, pp. 193-198, 2003.

[106] J.E. Fieldsand and S. Singh, "A multi-objective algorithm based upon particle swarm optimization, an efficient data structure and turbulence," *Proceedings of UK Workshop on Computational Intelligence*, Birmingham, UK, pp. 37-44, 2002.

[107] S. Mostaghim and J. Teich, "The role of ε–dominance in multi objective particle swarm optimization methods," *Proceedings of Congress on Evolutionary Computation*, Canberra, Australia, pp. 1764-1771, 2003.

[108] Hao Jiang, Jin-hua Zheng, and Liang-jun Chen, "Multi-objective particle swarm optimization algorithm based on enhanced ε-dominance," *Proceedings of IEEE International Conference on Engineering of Intelligent Systems*, Islamabad, Pakistan, pp.1-5, April, 2006.

[109] X. Li, "A non-dominated sorting particle swarm optimizer for multiobjective optimization," *Proceedings of Genetic and Evolutionary Computation Conference,* Vol. 2723, pp. 37-48, 2003.

[110] C.A. Coello Coello, and M.S. Lechuga, "MOPSO: A proposal for multiple objective particle swarm optimization," *Proceedings of Congress on Evolutionary Computation*, Honolulu, HI., pp. 1051-1056, 2002.

[111] X. Hu and R.C. Eberhart, " Multiobjective optimization using dynamic neighborhood particle swarm optimization," *Proceedings of Congress on Evolutionary Computation*, Honolulu, HI, pp. 1677-1681, 2002.

[112] L.B. Zhang, C.G. Zhou, X.H. Liu, Z.Q. Ma, M. Ma, and Y.C. Liang, "Solving multi objective problems using particle swarm optimization," *Proceedings of Congress on Evolutionary Computation*, Canberra, Australia, pp. 2400-2405, 2003.

[113] S. Mostaghim and J. Teich, "Strategies for finding good local guides in multi-objective particle swarm optimization," *Proceedings of Swarm Intelligence Symposium*, Indianapolis, IN, pp. 26-33, 2003.

[114] D. Ireland, A, Lewis, S. Mostaghim, and Jun Wei Lu, "Hybrid particle guide selection methods in multi-objective particle swarm optimization," *Proceedings of Second IEEE International Conference on e-Science and Grid Computing*, Amsterdam, Netherlands, pp. 116 – 116. 2006.

[115] M.A. Villalobos-Arias, G.T. Pulido, and C.A. Coello Coello, "A proposal to use stripes to maintain diversity in a multi-objective particle swarm optimizer," *Proceedings of Swarm Intelligence Symposium*, Pasadena, CA, pp. 22-29, 2005.

[116] D.W. Gong, Y. Zhang, and J.H. Zhang, "Multi-objective particle swarm optimization based on minimal particle angle," *Proceedings of International Conference on Intelligent Computing*, Hefei, China, pp. 571-580, 2005.

[117] J.E. Alvarez-Benitez, R.M. Everson, and J.E. Fieldsend, "A MOPSO algorithm based exclusively on Pareto dominance concepts," *Proceedings of Evolutionary Multi-Criterion Optimization Conference*, Guanajuato, Mexico, pp. 459-473, 2005.

[118] Jürgen Branke and Sanaz Mostaghim, "About selecting the personal best in multi-objective particle swarm optimization," *Proceedings of the 9th International Conference Parallel Problem Solving from Nature*, Reykjavik, Iceland, Vol. 4193, pp. 523-532, 2006.

[119] S.L. Ho, Shiyou Yang, Guangzheng Ni, E.W.C. Lo, and H.C. Wong, "A particle swarm optimization-based method for multiobjective design optimizations," *IEEE Transactions on Magnetics*, Vol. 41, No. 5, pp. 1756 – 1759, 2005.

[120] C.A. Coello Coello, G. Toscano Pulido, and M.S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, Vol. 8, No. 3, pp. 256-279, 2004.

[121] M.R. Sierra and C.A. Coello Coello, "Improving PSO-based multi-objective optimization using crowding, mutation and ε–dominance," *Proceedings of Evolutionary Multi-Criterion Optimization Conference,* Guanajuato, Mexico, Vol. 3410, pp. 505-519, 2005.

[122] F. de Toro, J. Ortega, and B. Paechter, "Parallel single front genetic algorithm: performance analysis in a cluster system," *Proceedings of International Parallel and Distributed Processing Symposium*, Nice, France, pp. 143-148, 2003.

[123] S. Xiong and F. Li, "Parallel strength Pareto multi-objective evolutionary algorithm for optimization problems," *Proceedings of Congress on Evolutionary Computation*, Canberra, Australia, pp. 2712- 2718, 2003.

[124] K.C. Tan, T.H. Lee, Y.J. Yang, and D.S. Liu, "A cooperative coevolutionary algorithm for multiobjective optimization," *Proceedings of IEEE International Conference on Systems, Man, and Cybernetic*, The Hague, The Netherlands, pp. 1926-1931, 2004.

[125] K.E. Parsopoulos, D.K. Tasoulis, N.G. Pavlidis, V.P. Plagianakos, and M.N. Vrahatis, "Vector evaluated differential evolution for multiobjective

optimization," *Proceedings of Congress on Evolutionary Computation*, Portland, OR, pp. 204-211, 2004.

[126] S. Ando and E, Suzuki, "Distributed multi-objective GA for generating comprehensive Pareto front in deceptive optimization problems," *Proceedings of Congress on Evolutionary Computation*, Vancouver, Canada, pp. 1569-1576, 2006.

[127] K. Izumi, M.M.A. Hashem, and K. Watanabe, "An evolution strategy with competing subpopulations," *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Monterey, CA, pp. 306-311, 1997.

[128] G. Toscano Pulido and C.A. Coello Coello, "Using clustering techniques to improve the performance of a particle swarm optimizer," *Proceedings of Genetic and Evolutionary Computation Conference,* Vol. 3102, Seattle, WA, pp. 225-237, 2004.

[129] S. Mostaghim and J. Teich, "Covering Pareto-optimal fronts by subswarms in multi-objective particle swarm optimization," *Proceedings of Congress on Evolutionary Computation*, Portland, OR, pp. 1404-1411, 2004.

[130] Ching-Shih Tsou, Hsiao-Hua Fang, Hsu-Hwa Chang, and Chia-Hung Kao, "An improved particle swarm Pareto optimizer with local search and clustering," *Proceedings of the 6th International Conference on Simulated Evolution and Learnin*g, Hefei, China, Vol. 4247, pp. 400-407, 2006.

[131] Hong-yun Meng, Xiao-hua Zhang, and San-yang Liu, "A co-evolutionary particle swarm optimization-based method for multiobjective optimization," *Proceedings of the 18th Australian Joint Conference on Artificial Intelligence*, Sydney, Australia, Vol. 3809, pp. 349-359, 2005.

[132] L.V. Santana-Quintero, N. Ramirez-Santiago, and C.A. Coello Coello, "A new proposal for multiobjective optimization using particle swarm optimization and rough sets theory," *Proceedings of the 9th International Conference Parallel Problem Solving from Nature*, Reykjavik, Iceland, Vol. 4193, pp. 483-492, 2006.

[133] Xiaohua Zhang, Hongyun Meng, and Licheng Jiao, "Intelligent particle swarm optimization in multiobjective optimization," *Proceedings of Congress on Evolutionary Computation*, Edinburgh, Scotland, Vol. 1, pp. 714- 719, 2005.

[134] Xiaohua Zhang, Hongyun Meng, and Licheng Jiao, "Improving PSO-based multiobjective optimization using competition and immunity clonal," *Proceedings of International Conference of Computational Intelligence and Security,* Xi'an, China, Vol. 3801, pp. 839-845, 2005.

[135] K.C. Tan, T.H. Lee and E.F. Khor, "Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, Vol. 5, No. 6, pp. 565-588, 2001.

[136] J. Arabas, Z. Michalewicz and J. Mulawka, "GAVaPS- a genetic algorithm with varying population size," *Proceedings of Congress on Evolutionary Computation*, Orlando, FL, pp. 73-74, 1994.

[137] N. Zhuang, M. Benten and P. Cheung, "Improved variable ordering of BBDS with novel genetic algorithm," *Proceedings of IEEE International Symposium on Circuits and Systems*, Atlanta, GA**,** pp. 414-417, 1996.

[138] J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transaction on Systems, Man and Cybernetics*, Vol. 16, No. 1, pp. 122-128, 1986.

[139] H. Lu and G.G. Yen, "Dynamic population size in multiobjective evolutionary algorithms," *Proceedings of Congress on Evolutionary Computation*, Honolulu, HI, pp 1648-1653, 2002.

[140] G.G. Yen and H. Lu, "Dynamic multiobjective evolutionary algorithm: adaptive cell-based rank and density estimation," *IEEE Transactions on Evolutionary Computations*, Vol. 7, No. 3, pp. 253-274, 2003.

[141] H. Eskandari, L. Rabelo, and M. Mollaghasemi, "Multiobjective simulation optimization using an enhanced genetic algorithm," *Proceedings of the 37th Winter Simulation Conference*, Orlando, FL, pp. 833-841, 2005.

[142] C.A. Coello Coello and E.M. Montes, "Constraint-handling in genetic algorithms through the use of dominance-based tournament selection," *Advanced Engineering Information*, Vol. 16, pp. 193-203, 2002.

[143] J.D. Knowles and D.W. Corne, "M-PAES: A memetic algorithm for multiobjective optimization," *Proceedings of Congress on Evolutionary Computation*, La Jolla, CA, pp. 325-332, 2000.

[144] J.D. Knowles, L. Thiele and E. Zitzler, "A tutorial on performance assessment of stochastic multiobjective optimizers," TIK-Report No. 214 (revised version), Computer Engineering and Network Laboratory, ETH Zurich, Switzerland, pp. 1-35, February 2006.

[145] P.S. Andrews, "An investigation into mutation operators for particle swarm optimization," *Proceedings of Congress on Evolutionary Computation*, Vancouver, Canada, pp 3789-3796, 2006.

[146] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[147] T. Okabe, Y. Jin, B. Sendhoff, and M.Olhofer, "Voronoi-based estimation of distribution algorithm for multi-objective optimization," *Proceedings of Congress on Evolutionary Computation*, Portland, OR, pp. 1594-1601, 2004.

[148] J.D. Knowles, "ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems," *IEEE Transactions on Evolutionary Computation*, Vol. 10, No. 1, pp.50-66, 2005.

[149] C. Grosan, C. Dumitrescu, and D. Lazar, "A particle swarm optimization for solving 0/1 knapsack problem." *Proceedings of International Conference on Computers and Communications*, Oradea, Romania, pp. 200-204, 2004.

[150] H.S. Lope and L.S. Coelho "Particle swarm optimization with fast local search for the blind traveling salesman problem," *Proceedings of the International Conference on Hybrid Intelligent Systems*, Brazil, pp. 245-250, 2005.

[151] S.A Zonouz, J. Habibi, and M. Saniee, "A hybrid PS-based optimization algorithm for solving traveling salesman problem," *Proceedings of the IEEE International Symposium on Frontiers in Networking with Applications*, Austria, pp. 245-250, 2006.

[152] R.L. Becerra, C.A. Coello Coello, A.G. Hernández-Diaz, R. Caballero, and J. Molina, "Alternative technique to solve hard multi-objective optimization problems," *Proceedings of Genetic and Evolutionary Computation Conference*, London, UK, pp. 757-764, 2007.

[153] P.R. Ehrlich, D.S. Dobkin, and D. Wheye. Mixed-species flocking. *Birds of Stanford*. Available: http://www.stanford.edu/group/stanfordbirds/text/essays/MixedSpecies_Flocking.html

[154] F. Backhouse, *"Chapter 7: Relationships with other species,"* Woodpeckers of North America*. Firefly Books, Ontario, Canada, pp. 101-114, 2005.*

[155] T.P. Runarsson and X. Yao, "Search biases in constrained evolutionary optimization," *IEEE Transactions On Evolutionry Computation*, Vol. 35, No. 2, pp. 233-243, 2005.

[156] T. Takahama and S. Sakai, "Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elites," *Proceedings of Congress on Evolutionary Computation*, Vancouver, Canada, pp.1-8, 2006.

[157] Z. Cai and Y. Wang, "A multiobjective optimization-based evolutionary algorithm for constrained optimization," *IEEE Transactions on Evolutionary Computation*, Vol. 10, No. 6, pp. 658-674, 2006.

[158] C.M. Fonseca and P.J. Fleming, "Multiobjective optimization and multiple constraint handling with evolutionary algorithms, I: a unified formulation," *IEEE Transactions on Systems, Man, and Cybernetics*, *Part A*, Vol. 28, No.1, pp. 26-37, 1998.

[159] C.A. Coello Coello and A.D. Christiansen, "MOSES: A multiobjective optimization tool for engineering design," *Engineering Optimization,* Vol. 31, pp. 337-368, 1999.

[160] T. Binh and U. Korn, "MOBES: A multiobjective evolution strategy for constrained optimization problems," *Proceedings of the 3rd International Conference on Genetic Algorithms*, Brno, Czech Republic, pp. 176-182, 1997.

[161] F. Jimenéz, A.F. Gomez-Skarmeta, G. Sanchez and K. Deb, "An evolutionary algorithm for constrained multiobjective optimization," *Proceedings of Congress on Evolutionary Computation*, Honolulu, HI, pp. 1133-1138, 2002.

[162] T. Ray and K.S. Won, "An evolutionary algorithm for constrained bi-objective optimization using radial slots," *Proceedings of the 9th International Conference of Knowledge-Based Intelligent Information and Engineering Systems*, Melbourne, Australia, pp. 49-56, 2005.

[163] K. Harada, J. Sakuma, I. Ono and S. Kobayashi, "Constraint-handling method for multi-objective function optimization: Pareto descent repair operator," *Proceedings of the 4th International Conference of Evolutionary Multi-Criterion Optimization*, Matsushima/Sendai, Japan, pp.156-170, 2007.

[164] H. Geng, M. Zhang, L. Huang and X. Wang, "Infeasible elitists and stochastic ranking selection in constrained evolutionary multi-objective optimization," *Proceedings of the 6th International Conference of Simulated Evolution and Learning*, Hefei, China, pp. 336-344, 2006.

[165] D. Chafekar, J. Xuan and K. Rasheed, "Constrained multi-objective optimization using steady state genetic algorithms," *Proceedings of Genetic and Evolutionary Computation Conference*, Chicago, Illinois, pp. 813-824, 2003.

[166] Y.G. Woldesenbet, B.G. Tessema and G.G. Yen, "Constraint handling in multi-objective evolutionary optimization," *Proceedings of Congress on Evolutionary Computation*. Singapore, pp. 3077-3084, 2007.

[167] K.E. Parsopoulus and M.N. Vrahatis, "Particle swarm optimization method for constrained optimization problems," *Technologies - Theory and Applications: New Trends in Intelligent Technologies*, pp. 214-220, 2002.

[168] K. Zielinski and R. Laur, "Constrained single-objective optimization using particle swarm optimization," *Proceedings of Congress on Evolutionary Computation*, B.C. Canada, pp. 443-450, 2006.

[169] Q. He and L. Wang, "A hybrid particle swarm optimization with a feasibility-based rule for constrained optimization," *Applied Mathematics and Computation*, Vol. 186, pp. 1407-1422, 2007.

[170] G.T. Pulido, C.A. Coello Coello, "A constraint-handling mechanism for particle swarm optimization," *Proceedings of Congress on Evolutionary Computation*, Vol. 2, California, USA, pp. 1396 - 1403, 2004

[171] Z. Liu, C. Wang, and J. Li, "Solving Constrained Optimization via a Modified Genetic Particle Swarm Optimization," *International Workshop on Knowledge Discovery and Data Mining*, Adelaide, Austrilia, pp. 217-220, 2008.

[172] H. Lu and W. Chen, "Dynamic-objective particle swarm optimization for constrained optimization problems," *Journal of combinatorial optimization*, Vol 2, No. 4, pp. 409-419, 2006.

[173] L.D. Li, X. Li, and X. Yu, "A multi-objective constraint-handling method with PSO algorithm for constrained engineering optimization problems," *Proceedings of IEEE Congress on Evolutionary Computation*, Hong Kong, China, pp. 1528-1535, 2008.

[174] J.J. Liang and P.N. Suganthan, "Dynamic multi-swarm particle swarm optimizer with a novel constraint-handling mechanism," *Proceedings of Congress on Evolutionary Computation*, B.C. Canada, pp. 9-16, 2006.

[175] D.L. Cushman, *A particle swarm approach to constrained optimization informed by 'Global Worst'*, Pennsylvania State University, Pennsylvania, 2007.

[176] E. Mezura-Montes and C.A. Coello Coello, "A survey of constraint-handling techniques based on evolutionary multiobjective optimization," Technical Report EVOCINV-04-2006, *CINVESTAV-IPN*, Mexico City, México, 2006.

[177] C.A. Coello Coello and G.T. Pulido, "Multiobjective optimization using a micro-genetic algorithm," *Proceedings of Genetic and Evolutionary Computation Conference*, San Francisco, California, pp. 274-282, 2001.

[178] S. Venkatraman and G.G. Yen, "A generic framework for constrained optimization using genetic algorithms," *IEEE Transactions on Evolutionary Computation*, Vol. 9, No. 4, pp. 424-435,2005.

[179] B. Yang, Y. Chen, Z. Zhao, and Q. Han, "A master-slave particle swarm optimization algorithm for solving constrained optimization problems,"

Proceedings of the 6th World Congress on Intelligent Control and Automation, Dalian, China, pp. 3208-3212, 2006.

[180] V.L. Huang, P.N. Suganthan, A.K. Qin and S. Baskar, "Multiobjective differential evolution with external archive and harmonic distance-based diversity measure," Technical Report MODE-2005, *School of Electrical and Electronic Engineering, Nanyang Technological University,* Singapore*,* 2005.

[181] J.J. Liang, T.P. Runarsson, E. Mezura-Montes, M. Clere, P.N. Suganthan, C.A. Coello Coello, and K. Deb, "Problem definitions and evaluation criteria for CEC2006 special session on constrained real-parameter optimization," 2006.

[182] A.M. Zavala, A.H. Aguirre, and E.V. Diharce, "Robust PSO-based constrained optimization by perturbing the particle's memory," *Swarm Intelligence: Focus on ant and particle swarm optimization*, Felix T. S. Chan and Manoj Kumar Tiwari, Ed. I-Tech Education and Publishing, 2007, pp. 57-76.

[183] M. Tanaka, "GA-based decision support system for multi-criteria optimization," *Proceeding of International Conference on Evolutionary Multi-Criteria Optimization*, Guanajuato, Mexico, pp. 1556-1561, 1995.

[184] A. Osyezka and S. Kundu, "A new method to solve generalized multi-criteria optimization problems using the simple genetic algorithm," *Structural Optimization*, Vol. 10, No. 2, pp. 94-99, 1995.

[185] K. Deb, A. Pratap and T. Meyarivan, "Constrained test problems for multi-objective evolutionary optimization," *Proceeding of the First International Conference of Evolutionary Multi-Criterion Optimization*, Zurich, Switzerland, pp.284-298, 2001.

[186] J. Horn, N. Nafpliotis, and D.E. Goldberg "A niched Pareto genetic algorithm for multiobjective optimization," *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, Piscatawaym NJ, pp.82-87,1994.

[187] N. Srinivas and K. Deb, "Multiobjective optimizatiom using nondominated sorting in genetic algorithm," *Evolutionary Computation*, Vol. 2, No. 3, pp/ 221-248, 1994.

[188] S.N. Omkar, Dheevatsa Mudigere, G. Narayana Naik, and S. Gopalakrishnan, "Vector evaluated particle swarm optimization (VEPSO) for multi-objective design optimization of composite structures," *Computers and Structures*, Vol. 86, No. 1-2, pp. 1-14, 2008.

[189] T.H. Labella, M. Dorigo, and J.-L. Deneubourg, "Division of labour in a group of robots inspired by ants' foraging behaviour," *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 1, No. 1, pp. 4-25, 2006.

[190] G. Di Caro, F. Ducatelle, and L.M. Gambardella, "AntHocNet: An Adaptive Nature-Inspired Algorithm for Routing in Mobile Ad Hoc Networks," *European Transactions on Telecommunications, Special Issue on Self Organization in Mobile Networking*, Vol. 16, No. 5, pp. 443-455, 2005.

[191] M.P. Wachowiak, R. Smolikova, Yufeng Zheng, J.M. Zurada, and A.S. Elmaghraby, "An approach to multimodal biomedical image registration utilizing particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, Vol. 8, No. 3, pp. 289-301, 2004.

[192] J. Robinson and Y. Rahmat-Samii, "Particle swarm optimization in electromagnetics," *IEEE Transactions on Antennas and Propagation*, Vol. 52, No. 2, pp. 397-407, 2004.

[193] X. Shi, K.S. Yeo, J.-G. Ma, M.A. Do, and E. Li, "Scalable model of on-wafer interconnects for high-speed CMOS ICs," *IEEE Transactions on Advanced Packaging*, Vol. 29, No. 4, pp. 770-776, 2006.

[194] N. Jin and Y. Rahmat-Samii, "Advances in particle swarm optimization for antenna designs: real-number, binary, single-objective and multiobjective implementations," *IEEE Transactions on Antennas and Propagation*, Vol. 55, No. 3 I, pp. 556-567, 2007.

[195] C.-M. Huang, C.-J. Huang, and M.-L. Wang, "A particle swarm optimization to identifying the ARMAX model for short-term load forecasting," *IEEE Transactions on Power Systems*, Vol. 20, No. 2, pp. 1126-1133, 2005.

[196] L. Messerschmidt and A. Engelbrecht, "Learning to play games using a PSO-based competitive learning approach," *IEEE Transactions on Evolutionary Computation*, Vol. 8, No. 3, pp. 280–288, 2004.

VITA

Wen Fung Leong

Candidate for the Degree of

Doctor of Philosophy

Dissertation: MULTIOBJECTIVE PARTICLE SWARM OPTIMIZATION: INTEGRATION OF DYNAMIC POPULATION AND MULTIPLE-SWARM CONCEPTS AND CONSTRAINT HANDLING

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Seremban, Malaysia, the daughter of W.H. Leong and K.M. Yim.

Education: Received Bachelors of Science degree in Electrical Engineering from Oklahoma State University, Stillwater, Oklahoma in July, 2000.
Received Master of Science degree in Electrical Engineering from Oklahoma State University, Stillwater, Oklahoma in May, 2002.
Completed the requirements for the Doctor of Philosophy degree with major at Oklahoma State University, Stillwater, Oklahoma in December, 2008.

Experience: Research Assistance, Intelligent Systems and Control Laboratory (ISCL), Oklahoma State University.
Webmaster for WCCI 2006 conference webpage.
Teaching Assistant, Electrical and Computer Engineer Department, Oklahoma State University.

Professional Memberships: IEEE Computational Intelligence Society

Name: Wen Fung Leong Date of Degree: December, 2008

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: MULTIOBJECTIVE PARTICLE SWARM OPTIMIZATION: INTEGRATION OF DYNAMIC POPULATION AND MULTIPLE-SWARM CONCEPTS AND CONSTRAINT HANDLING

Pages in Study: 245 Candidate for the Degree of Doctor of Philosophy

Major Field: Electrical Engineering

Scope and Method of Study: Over the years, most multiobjective particle swarm optimization (MOPSO) algorithms are developed to effectively and efficiently solve unconstrained multiobjective optimization problems (MOPs). However, in the real world application, many optimization problems involve a set of constraints (functions). In this study, the first research goal is to develop state-of-the-art MOPSOs that incorporated the dynamic population size and multiple-swarm concepts to exploit possible improvement in efficiency and performance of existing MOPSOs in solving the unconstrained MOPs. The proposed MOPSOs are designed in two different perspectives: 1) dynamic population size of multiple-swarm MOPSO (DMOPSO) integrates the dynamic swarm population size with a fixed number of swarms and other strategies to support the concepts; and 2) dynamic multiple swarms in multiobjective particle swarm optimization (DSMOPSO), dynamic swarm strategy is incorporated wherein the number of swarms with a fixed swarm size is dynamically adjusted during the search process. The second research goal is to develop a MOPSO with design elements that utilize the PSO's key mechanisms to effectively solve for constrained multiobjective optimization problems (CMOPs).

Findings and Conclusions: DMOPSO shows competitive to selected MOPSOs in producing well approximated Pareto front with improved diversity and convergence, as well as able to contribute reduced computational cost while DSMOPSO shows competitive results in producing well extended, uniformly distributed, and near optimum Pareto fronts, with reduced computational cost for some selected benchmark functions. Sensitivity analysis is conducted to study the impact of the tuning parameters on the performance of DSMOPSO and to provide recommendation on parameter settings. For the proposed constrained MOPSO, simulation results indicate that it is highly competitive in solving the constrained benchmark problems.

ADVISER'S APPROVAL: Dr. Gary G. Yen