UNCERTAINTY AND CONSTRAINT HANDLING IN EVOLUTIONARY ALGORITHMS

By

YONAS GEBRE WOLDESENBET

Bachelor of Science in Electrical Engineering

Bahir Dar University

Bahir Dar, Ethiopia

2004

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2007

UNCERTAINTY AND CONSTRAINT HANDLING IN

EVOLUTIONARY ALGORITHMS

Thesis Approved:

Dr. Gary Yen

Advisor

Dr. Martin Hagan

Dr. Louis Johnson

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

I am very grateful to my advisor, Dr. Gary Yen, for his constant guide and support both in my academic and social endeavors. He has allowed me to further exploit my potentials and has motivated me with his deep knowledge and experience in the subject matter. This work wouldn't have been possible without his exceptional mentorship and dedication.

I am very blessed to have a loving and supporting family –Mom, Dawit, Meaza, Mintewab, Girum, Elisabeth, Helen, Eden, and Bethlehem. It is their constant love and prayers that gives me hope and strength. I will not be where I am right now without their love and support. And dad, it is because of your kindness and good deeds that God has overwhelmingly blessed our family. May God rest your soul.

I am also very grateful to Beka Hailu and his family; Biruk Tessema and his family; my uncle Tadesse Woldesenbet and all other people who have been there for me both in good and bad times. I am also thankful to my fellow graduate students, Wen-Fung Leong and Moayed Daneshyari; my roommates, Kumlachew Woldemariam and Amanuel Assefa; and the Ethiopian community in Stillwater for their encouragement and support.

I would also like to thank my professors and committee members, Dr. Martin Hagan and Dr. Louis G. Johnson, for attending my defense.

Above all, I will like to thank God for blessing my life and that of my beloved ones.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

Table Page

Table Page

LIST OF FIGURES

Figures Page

CHAPTER I

INTRODUCTION

1.1 Overview

Evolutionary algorithms (EAs) have been successfully applied to solve optimization problems in the fields of science and engineering. These algorithms are zero-order stochastic search approaches that mimic the process of natural selection to arrive at the optimal solutions. EAs have been an active research topic during the past few years and are gaining more attention especially in solving high dimensional, multimodal, discontinuous and/or NP-complete optimization problems.

Generally, evolutionary algorithms are implemented as computer programs in which a population of candidate solutions (called *individuals*) evolves from generation to generation toward finding better solutions for a given optimization problem [1]. The first step in a typical evolutionary algorithm is random initialization of individuals in the population. This population is evaluated using the objective function(s) and a corresponding *fitness* value is assigned to each individual. The fitness of an individual measures how well the individual satisfies the optimality condition. Based on their fitness values, a number of individuals, so-called parent individuals, are selected from the current population. The selection can be done in several ways. One of the popular designs, called tournament selection, is achieved by first choosing several individuals randomly from the current population and then by picking the best individual out of the selection. The chosen parent individuals are then modified by applying genetic operators to form a new population called the *offspring* population. There are two kinds of genetic operators that are commonly used in evolutionary algorithms. These are *crossover* and *mutation* operators. In *crossover* operation, two *parent individuals* recombine to produce an offspring individual whose genetic information is obtained partly from the first parent and the remaining from the second parent. On the other hand, a *mutation* operator alters a

single individual to form an offspring individual. The new population created by applying genetic operations will then *replace* some of the poorly fit solutions in the original population. This process is repeated until the termination condition, which is usually the *maximum generation* number, is reached. Finally, the best found individual will be reported as the optimal solution. The pseudo-code for a typical evolutionary algorithm is shown below in Figure 1.1.

```
Procedure for Evolutionary Algorithm
Begin
    nGen ← 0
    Initialize population
    Evaluate Population
    While (not termination_condition) do
    Begin
        Selection
        Recombination and/or Mutation
        Evaluation
        Replacement
        nGen ← nGen + 1
    End
End
```

Figure 1.1 Pseudo-code for a typical evolutionary algorithm

Evolutionary algorithms are preferred than traditional search techniques as global optimization techniques for several reasons. The main reasons include [2]:

A. EAs don't require prior knowledge of the problem in order to carry out the search. They make random changes to their candidate solutions and then use the fitness function to determine whether those changes produce an improvement or not.

B. EAs use stochastic instead of deterministic operators and appear to be robust in problems where the fitness function is complex, discontinuous, noisy, time varying, or has many local optima.
C. EAs operate on multiple solutions simultaneously, gathering information from a population of search points to direct subsequent search effort. This will make EAs less susceptible to the problems of local optima and noise. Most algorithms can only explore the solution space to a problem in only one direction at a time; and if the solution they discover is suboptimal the search should be repeated again.
D. EAs are suitable for parallel evaluation of many solutions at once. As a result, they are particularly well-suited to solving nonlinear problems where the search space is very large.

However the price to pay when using EAs is twofold. First because of their stochastic nature, EAs can not guarantee finding the optimal solution in every run. Secondly, the computational cost associated with EAs is generally very high, and a large number of function evaluations must be performed for a satisfying result to be found. Therefore it is usually advised not to use EAs whenever a deterministic optimization method can provide quality solutions.

Evolutionary algorithms have been successfully applied to solve many real-world optimization problems. However, a significant number of these optimization problems have to be performed under the presence of various uncertainties and constraints and it has become a recent trend to devise techniques to handle these features in the evolutionary algorithm.

Uncertainties are one of the two most common features in real-world optimization. In general, four major categories of uncertainties have been dealt with using evolutionary approaches. These are noise in the fitness function, perturbations in the design variables, approximation in the fitness function, and dynamism in the optimal solutions [3]. Noise and approximation introduce uncertainty in the fitness function, while perturbation brings uncertainty in the decision space. On the other hand, dynamic optima result in uncertainty in the location, height, and width of the optimal solutions through time. A significant number of uncertainty problems have dynamism in their optimal solutions and one focus of this thesis is on these types of problems.

Constraints are the second most common features in real-world optimization problems. Constraints restrict the usable regions of the decision space and impose greater difficulty on the optimization process. Constrained evolutionary algorithms not only have to find optimal solutions but also have to make sure that these solutions satisfy the constraints. Several researches have been done to handle constraints in evolutionary algorithms for single objective optimization problems. However, few researches have been devoted to handle constraints in multi-objective evolutionary algorithms.

In this paper, we discuss about two major types of optimization problems - dynamic optimization problems (DOPs), which are common uncertainty problems in which the fitness function changes through time, and constrained multi-objective optimization problems (CMOPs), which are constrained optimization problems with a set of competing objectives. For each set of problems, we propose state-of-the-art evolutionary algorithm that will be shown to possess very promising performance.

1.2 Problem Definitions

1.2.1 *Dynamic optimization problem*

A dynamic optimization problem (DOP) can be formulated as:

Minimize / Maximize

$$f(X,e) = f(x_1, x_2, ..., x_n, e) \tag{1.1}$$

where each dimension of the search space is defined between $x_j^{\min} \le x_j \le x_j^{\max}$ for $j = 1,2,\cdots,n$. f is the objective function to be optimized; $X = (x_1, x_2, ..., x_n)$ is the *n*-dimensional decision vector.

e represents the environmental state whose variation can have either periodic or sporadic nature. This variable can be modeled in different ways. The common method is to use a counter that counts the environmental state. We can also use the time variable as the counter for the environment. The dynamics of change can have deterministic or stochastic nature. So, it is common to assume stochastic nature at first and then possibly pick up any recurrent pattern along the evolutionary process. The use of memory also increases the probability of locating the optimal solution in cyclic changes.

1.2.2 *Constrained multi-objective optimization problem*

A constrained multi-objective optimization problem (CMOP) can be mathematically formulated as:

Minimize / Maximize

$$f_i(x) = f_i(x_1, x_2, ..., x_n), \quad i = 1,2,..., p \tag{1.2}$$

Subject to $\quad g_i(x) = g_i(x_1, x_2, ..., x_n) < 0, \quad i = 1,..., q$

$$h_i(x) = h_i(x_1, x_2, ..., x_n) = 0, \quad i = q+1,..., m$$

$$x_j^{\min} \le x_j \le x_j^{\max}, \quad j = 1,2,..., n.$$

There are p objective functions that are required to be optimized simultaneously. Each objective function $f_i(x)$ is defined on the search space $S \subseteq \Re^n$. Usually the search space is an n-dimensional hyperbox in $\Re^n$. Each dimension of the search space is bounded by its upper ($x_j^{\max}$) and lower ($x_j^{\min}$) limits.

$g_i(x)$ is the i^{th}-inequality constraint, and $h_i(x)$ is the i^{th}-equality constraint. There are a total of m constraints, q inequality and m - q equality, which are required to be satisfied by the optimum solution. The presence of equality and inequality constraints will restrict our search space to a feasible region $F \subseteq S$, where a usable solution can be found.

1.3 Research Goal and Approach

The goal of this study is to propose two state-of-the-art evolutionary algorithms-one for dynamic optimization problems and another for constrained multi-objective optimization problems. These algorithms exploit as much information as possible from the previous generations to facilitate the evolutionary process. The dynamic evolutionary algorithm is further designed to have higher reusability, quicker adaptation, faster convergence, easier implementation, better accuracy, and ability to work under drastic changes and higher frequencies of change. On the other hand, the proposed constraint handling technique for multi-objective evolutionary algorithms is designed to provide a reliable algorithm that obtains accurate and diverse solutions with comparatively less

computational cost. The algorithm not only exploits the evolutionary information contained in feasible individuals, but also from infeasible individuals.

The proposed algorithms are implemented as computer programs and are then evaluated on several benchmark test problems. The simulation results are studied and compared with the results reported for some state-of-the-art designs.

1.4 Document Organization

The remaining part of this thesis is organized as follows. In Chapter II, dynamic optimization problems are discussed in further detail. The chapter presents the different approaches used so far, the various performance indexes used to measure performance, and the different types of benchmark problems used to test dynamic evolutionary algorithms. Finally in this chapter, a dynamic evolutionary algorithm is proposed and analyzed at length. Chapter III provides a review of constrained multi-objective optimization problems, different kinds of approaches proposed so far to solve such problems, and various performance indexes used to measure performance of these algorithms. Finally, a constraint handling technique for multi-objective evolutionary algorithms is proposed and discussed in detail. In Chapter IV, the two proposed algorithms are tested on different standard benchmark suites. The empirical data are numerically analyzed and the results are presented and discussed in detail. Chapter V concludes this thesis with relevant observations and remarks and provides recommendations for future work.

CHAPTER II

DYNAMIC EVOLUTIONARY OPTIMIZATION

2.1 Introduction

Many real world optimization problems sporadically change over time. These problems are collectively termed as dynamic optimization problems. The location of the optimal solution in these problems changes over time. These changes can result from changes in the environmental parameters, changes in the constraints, changes in the objectives or changes in the problem representations and settings [4]. These changes may then be reflected on the landscapes as changes in the optimal peak heights, peak shapes or peak locations or combination of these three [5].

A dynamic optimization problem (DOP) can be formulated as:

Minimize / Maximize

$$f(X, e) = f(x_1, x_2, ..., x_n, e) \tag{2.1}$$

where each dimension of the search space is defined between $x_j^{\min} \le x_j \le x_j^{\max}$ for $j = 1,2,\cdots,n$. f is the objective function to be optimized; $X = (x_1, x_2, ..., x_n)$ is the *n*-dimensional decision vector and e represents the environmental state whose variation can have either periodic or sporadic nature.

There are several practical applications as dynamic optimization problems. A very good example is dynamic portfolio optimization which is a common optimization problem in modern finance [6]. This problem aims to obtain an optimal set of assets that maximize profit while minimizing risk of investment. In engineering, dynamic portfolio optimization problems are common in deregulated electricity markets in which the operations of different power stations are controlled and coordinated to maximize profit while minimizing risk. There are various uncertainties in a deregulated electricity market,

such as spot market prices, load obligations, and strip/option prices [7]. The values for some of these factors change over time and it is not unusual to optimize for the market price at each hour.

Another good example of real-world dynamic optimization problem is the dynamic job shop scheduling problem [8]. This problem is a variation of the job shop scheduling problem in which new jobs arrive over time after the scheduling process has started. The dynamism in the problem can also arise due to cases where some machines break down or wear out slowly, cases where the quality of the raw material changes; or cases where the production tolerances are required to be taken into account. Therefore, the job schedules should be dynamically modified to accommodate the changes over time.

In general, a good dynamic evolutionary algorithm (DEA) must be able to track the changing optimal solution irrespective of the severity and frequency of change. It must be able to reuse as much information as possible from previous generations to speedup the optimization search. Furthermore, the extra computational cost incurred should be reasonably comparable to its performance improvement.

The challenges in solving DEAs arise from the occurrence of changes in the location, number, and properties of the optimal solutions. When a standard evolutionary algorithm converges for a certain problem setting, the diversity and exploration capability of the population are greatly diminished. As a result, continuing the evolutionary process from the converged population without any adaptation scheme or facilitation of exploration creates a higher probability of being unable to find the new optimal solutions or of being stuck with local optima. Therefore, it is necessary to implement certain scheme in the evolutionary algorithm to account for the dynamism of the optimization problem.

There are several important aspects of dynamic optimization problems (DOPs) and these include severity, frequency, observability, detectability, and dynamics of change [8]. While higher severity of change necessitates the DEA to increase diversity and exploration, higher frequency of change requires a faster convergence after a change has occurred. If the severity of change is too high for the DEA, the algorithm may not locate the new optima or might get stuck in local optima. Similarly, if the frequency of change is faster than the adaptation speed of the DEA, the algorithm will not reach the optimal solutions before another change occurs.

On the other hand, detection of change can be made in several ways assuming that the change is observable. Some of the common practices include checking if the time averaged best performance of DEA deteriorates and checking if the fitness of at least one of the reevaluated individuals has changed appreciably [8]. It is also a common practice to assume that a change is explicitly known to the system and that the system will observe and detect a change instantly. For the discussion of this study, the authors assume that the change is explicitly known to the system.

Furthermore, the DOP may involve different dynamics changes and these include constant, linear, circular/revolving, reshaping, and random modes [8], [9], [10]. In most cases, the type of dynamics is not explicitly known to the system and the algorithm is expected to work without knowing it. In this chapter, we present a dynamic evolutionary algorithm that adapts all individuals of the previous population when a change occurs based on evolutionary history. The adaptation is carried out by relocation vectors that introduce shifts in the individuals' decision variables to enhance the population's diversity.

Some sought qualities in dynamic evolutionary algorithms include reusability, faster convergence, higher accuracy, faster adaptation, easier implementation, and better performance. Reusability refers to the ability to reuse as much information as possible from the previous evolutionary process. Reusability normally provides faster convergence and allows the algorithm to adapt to the new environmental condition quickly. Higher severity of change reduces reusability of previous evolutionary data and demands greater exploration capability. Higher frequency of change demands faster convergence and adaptation to the new environment. Accuracy, on the other hand, refers to how close the best individual found is to the actual optimal solution. Improvement in the accuracy of the optimal solution may compromise the speed of convergence and hence, the algorithm should be equipped to balance between the additional computational cost and the observed performance improvement. Adaptation, alternatively, refers to adjusting the current population to the new environmental condition. Adaptation can be done by modifying the level of evolutionary operators, like mutation, to encourage exploration or by modifying the evolutionary process based on the previous evolutionary history.

In this chapter, the above qualities are achieved by using variable relocation vectors that adapt already converged or currently evolving individuals to the new environmental condition. The proposed algorithm relocates the individuals based on their change in functional value due to the change in the environment and the average sensitivities of their decision variables to the corresponding change in the objective space. The relocation vectors introduce a certain radius of uncertainty to be applied to each individual and restore diversity and accelerating exploration. Furthermore, because the population is adapted from the previous population, there is a higher reuse of previous evolutionary material, which often provides faster convergence. The relocation vectors are specific to each individual, and this gives the algorithm better adaptation than those approaches that use a single adaptation value for the whole population. As a technique to be used at transient periods, the proposed algorithm provides the next evolutionary cycle with better initial population than any other randomly generated population. As a result, there will be a considerable progress jump for the upcoming evolutionary process, and this gives the proposed algorithm faster adaptation and convergence.

This chapter is structured as follows. Section 2.2 provides a brief summary of the various types of fitness landscapes. Section 2.3 discusses various aspects of dynamic optimization problems. Then Section 2.4 presents a review of the various evolutionary approaches proposed so far for dynamic optimization problems. Next in Section 2.5, two performance metrics used for measuring the performance of dynamic evolutionary algorithms are presented. In Section 2.6, various dynamic benchmark test problems are discussed. In Section 2.7, the proposed relocation vector based dynamic evolutionary algorithm (RVDEA) is elaborated and analyzed. Finally, we conclude with a summary in Section 2.8.

2.2 Types of Fitness Landscapes

Depending on the changes over the whole landscape through time, Weicker *et al* [10] classified fitness landscapes into various groups. The first category is that of *stationary or static landscapes* where there is no change or movement in the landscape. These types of landscapes are the ones that are commonly used in most EA studies. The second type of

fitness landscape has changes over the *landscape that is constant every period of time*. A good dynamic evolutionary algorithm should be able to pick up the recurrent similarity in the amount of change and use it in the upcoming change period. These types of problems are comparatively easy and impose very small difficulty to the dynamic evolutionary algorithm. The third type of fitness landscapes has *periodic changes* in which the landscape returns to its original state at certain intervals. The challenge in this type of landscapes is predicting the length of the period and most of the time this is difficult to do. Most algorithms implement a small-sized memory to hold the latest optimum solutions from few of the past changes. This allows possible re-usage of the optimum solutions found so far in future fitness landscapes that resemble any of the past landscapes. The fourth type of fitness landscape is called *homogenous landscape* where the whole landscape moves coherently, as opposed to various parts behaving heterogeneously. The last type of landscapes is *alternating landscape* where the optimum point jumps from one component or peak of the landscape to another. The changes in the overall landscape are stochastic and heterogeneous. These types of problems impose the greatest difficulty to the dynamic evolutionary algorithm because the nature of the change can not be modeled accurately. Hence, it is important to implement a certain kind of dynamic adaptation scheme in the evolutionary algorithm to cope with the sporadic changes in the landscape.

2.3 Aspects of Dynamic Optimization Problems

There are several important aspects of dynamic optimization problems (DOP). Some of these are severity, frequency, observability, detectability and dynamics of change [8].

2.3.1 *Severity of change* - Severity of change indicates the strength of change that occurred in the landscape. Higher severity of change means that the current fitness landscape has lesser correlation with the previous one and this necessitates the dynamic evolutionary algorithm to increase diversity and exploration. If the severity of change is too high for the DEA, the algorithm may not locate the new optima or might get stuck in

local optima. Hence, this aspect of change demands a dynamic evolutionary algorithm that is capable of performing well under drastic changes in landscape.

2.3.2 *Frequency of change* - The frequency of change indicates how fast the landscape changes. A higher value of frequency means that the landscape changes quite frequently and the corresponding evolutionary algorithm is expected to act upon these changes very quickly. If the frequency of change is too high for the DEA, the algorithm will not reach the new optimum solution before another change occurs. Hence, a good dynamic evolutionary algorithm should have a high adaptation frequency and faster convergence so as adapt to changes even at high frequencies.

2.3.3 *Observability and detectability* - The next two closely related aspects of changes are observability and detectability. Most of the time the change is assumed to be observable and detectable. On the other hand, several methods have been proposed to detect changes. Some of these techniques detect changes by checking whether the time averaged best performance of the algorithm deteriorates or by checking whether the fitness of at least one of the re-evaluated individuals has changed or by constantly monitoring whether an explicit model of the environment is still consistent with the current environment. It is also a common practice to assume that the change is explicitly known to the system and that the system will detect the changes instantly. This assumption is useful for certain researchers to focus only on the development of a dynamic evolutionary algorithm, while other researchers study the issue of observability and detectabilty of change.

2.3.4 *Dynamics of change* - The dynamics of change represent the way the landscape moves when a change occurs. A drifting motion dynamics has a drifting motion in the landscape. An oscillatory motion dynamics has an optimum that oscillates periodically. A reshaping dynamics has a fitness landscape that changes its morphology when a change occurs. A revolving landscape dynamics has a landscape that rotates at each occurrence of change [9]. A random jump or stochastic landscape dynamics has an optimum that changes randomly. In most cases, the type of dynamics mode is not

explicitly known to the system and the DEA is expected to work without explicitly knowing it.

2.4 Literature Review

The following subsections provide a brief review of evolutionary approaches developed for dynamic optimization problems. For more detailed reviews, the readers are referred to references cited in [3], [4], and [8].

2.4.1 *Re-initialization* - The most naïve approach ever conceived for solving DOPs is to reinitialize the evolutionary process when a change occurs. A similar approach restarts the population based on evolutionary algorithm convergence [11]. The deficiency of these approaches is that almost none of the past evolutionary materials are ever used and this unavoidably hinders any possible speedup in convergence.

2.4.2 *Maintaining diversity* - The basic idea behind maintaining diversity in DEAs is to prevent the algorithm from premature convergence. Grefenstette [12] proposed the idea of introducing randomly generated individuals when a change occurs. The introduction of random immigrants allows the algorithm to keep a certain level of diversity for exploration. The algorithm is easy to understand and implement, but provides little means of adapting the current individuals to the new environment. Furthermore, when the changes are severe, the algorithm requires a larger number of random immigrants which will compromise the algorithm's performance. In this study, we used a similar implementation as in [8] in which 25 random immigrants are migrated into current population when a change occurs. This implementation of random immigrants will be referred to as *RI25* and when a memory is used, it will be referred to as *RI25mem*.

Meanwhile, Andersen [13] approached the issue of diversity maintenance by using fitness sharing as a means to favor less populated area. When a region is highly populated, fitness is shared by a large number of individuals and in effect reducing or penalizing their fitness. On the other hand, if a region is less populated, the fitness is shared between few individuals and hence their fitness is less affected and un-penalized.

As a result, less populated areas will be more favored than highly populated areas and in effect preserving diversity.

On the other hand, Ghosh and colleagues [14] used the age of an individual to favor the fitness of middle aged individuals which in effect will maintain the diversity of the overall population. On the other hand, Jin and colleagues [15] suggested imposing a lower threshold on step-size to maintain diversity in evolutionary strategies.

Although most researchers agree that having a diversified population is a good idea for DOPs, the fact remains that maintaining diversity may impose greater computation and may also slow down the evolutionary process.

2.4.3 *Memory based approaches* - Memory based approaches are commonly divided into two groups: explicit and implicit memory approaches. Explicit memory approaches are memory based approaches that uses external memory to store previous evolutionary information that may be helpful in future stages of the evolutionary process. The common approaches under this category use a small-sized memory to store the best solutions and add them back to the population if they are better fit than the current individuals [16], [17]. In [18], Acan and Tekol used a "gene library" to store promising genetic materials for reuse later in the evolutionary process. In [19], Trojanowski and Michalewicz used a short term memory to remember some of the solutions of an individual's ancestors so as to increase the diversity by reintroducing individuals that have been considered good in recent generations. Lastly, Bendtsen and Krink [20] used dynamic memory model that is updated during the evolutionary process. They used the memory to store best individuals for each change period, but at the same time allowed the stored individuals to be evolved by small amounts of Gaussian mutation in the direction of the current best individual.

On the other hand, implicit memory approaches do not use an explicitly defined external memory but some implicit form of memory exists in the system representation. One form of implicit memory is redundant representation which is commonly used to slow down convergence and favor diversity. Diploidy is a common approach in redundant representations. In [21, 22], Smith and Goldberg used tri-allelic scheme where an allele can take one of the three values "0", "1 recessive", and "1 dominant". In [23],

Ng and Wong proposed using a diploid scheme with four possible alleles ("0 recessive", "0 dominant", "1 recessive", and "1 dominant"). In [24], a multi-level structured gene-representation was used so that each level can activate or deactivate genes at the next lower level. In [25], the diploidy scheme proposed in [24] was extended by a dominance change mechanism. In [26], an additive diploidy scheme was used where the genes determining one trait are added in order to determine the phenotypic trait. The phenotypic trait becomes 1 when a certain threshold is exceeded, and is 0 otherwise.

The basic assumption in memory approaches is that out of the stored information in the memory, there might be some individuals that fall in the vicinity of the new optimal solution. This kind of assumption becomes inappropriate when there is non-cyclic stochastic dynamism in the optimal solution. In general, however, enhancing any of the other uncertainty handling techniques by memory is a good practice.

2.4.4 *Multiple population approaches* - Multiple population approaches use several subpopulations to track multiple peaks in the landscape. The method proposed in [27], called shifting balance genetic algorithm, uses one core population to exploit the best optimum found so far and several colonies to explore the search space. A diversity measure, distance to the core population, was included in fitness evaluations of the colonies. Another method proposed in [28], called self-organizing scouts (SOS), uses a small fraction of the population called "child population" to watch over the peaks while the rest of the population searches for other peaks. The size of parent and child population is adaptively adjusted depending on the performance of the population. Another method proposed in [29] is called multi-nationals genetic algorithm and uses a "hill valley detection procedure" that defines the borders of the subpopulations. A valley is detected if the fitness in a sample point is lower than the fitness of both end points. This method requires a large number of fitness evaluations.

The challenge in this type of approaches is that the algorithm should coordinate the operation of each sub-population. As a result, these approaches tend to incur large computational cost compared to single population approaches.

The multi-population implementations used in this study follow that of [8]. The first implementation is called *P3* which is a standard evolutionary algorithm with three

independent sub-populations. Similarly, *P3mem* is the variation of this implementation with memory. Another multi-population implementation divides the population into memory and search subpopulations and is denoted as *Mem/search*.

2.4.5 *Mutation and self-adaptation* - When a change occurs, the population undergoes a transient state where the values of the evolution operators are changed so as to enhance diversity and performance. Cobb and Grefenstette [30] introduced the idea of hyper-mutation in which mutation probability is increased immediately after a change has occurred. In this method, the individuals undergo a drastic increase in the mutation level when a change occurs, which in effect improves the diversity of the population. Vavak and colleagues [31] introduced the idea of variable local search (VLS) that uses a step-by-step increase in the mutation level based on the performance of the population. In [10], different self adaptation schemes were compared and these include uniform self adaptation, different mutation level for each dimension, mutation with covariance matrix adaptation and sphere mutation which learns the upper and lower limits of the required mutation level. In [32], multiplicative update rule is compared against self-adaptation mutation; while in [33] lognormal adaptation is compared against self-adaptation mutation. Other suggested techniques include life-time learning [34] and adaptive chaotic mutation [35].

The assumption in this type of approaches is that the changes are in the reach of the algorithm's adaptation capability. If this is not the case, the adapted population might not locate the new optimal solutions.

In general, memory based approaches are suitable for periodical optima; multi-population approaches are suitable for competing peaks; mutation and self-adaptation techniques are appropriate for landscapes with very fast but less drastic changes; and maintaining diversity is suitable for continuously moving optima [4].

The proposed algorithm is inspired from the comparatively small computational cost of self-adaptation schemes and also from the idea to have a better technique that can be used alongside other approaches for solving dynamic optimization problems. In this spirit, it can be categorized under self-adaptation schemes. The algorithm also utilizes a small memory to further improve its performance in cyclic changes. The unique attribute

about this algorithm is that the self-adaptation scheme and the relocation amount are specific to each individual, and this allows the algorithm to provide better adapted initial population to the new environment. There are two variations of the proposed algorithm – the first is *RVDEAmem* which is a relocation vector dynamic evolutionary algorithm enhanced with memory, and the second is *RVDEAclusters* which is a relocation vector based dynamic evolutionary algorithm enhanced with memory and several clusters to provide superior performance over a relative increase in the computational cost.

2.5 Performance Indexes for Dynamic Evolutionary Algorithms

There are several performance indexes that have been used to measure the performance of dynamic evolutionary algorithms. In this chapter, we used offline error performance [36] and adaptation performance [16] as a means to measure the performance of the proposed dynamic evolutionary algorithm.

2.5.1 *Offline error performance* -

Off-line error performance index [36] is the most common performance index and it is obtained as the average of the error between the true optimal point and the best fitness at each evaluation. It is mathematically expressed as:

$$e_{offline}^{av} = \frac{1}{T}\sum_{i=1}^{T}(f_{true} - f_{best}^{i}) \tag{2.2}$$

where i is the evaluation counter; T is the total number of evaluations considered; f_{true} is the true optimum solution which is updated whenever a change occurs; and finally f_{best}^{i} is the best individual out of the evaluations starting from the most recent occurrence of change until the current evaluation. This form of error formulation may not provide a good insight on how well the algorithm is performing when the optimal function values are very large.

2.5.2 *Adaptation performance* -

Adaptation performance [16] is the average ratio between the best fitness and the true optimum at each evaluation. It is mathematically expressed as:

$$I = \frac{1}{T}\sum_{i=1}^{T}\frac{f_{best}^{i}}{f_{true}} \tag{2.3}$$

where i is the evaluation counter; T is the total number of evaluations considered; f_{true} is the true optimum solution which is updated whenever a change occurs; and finally f_{best}^{i} is the best individual out of the evaluations starting from the most recent occurrence of change until the current evaluation. This way of error formulation is not a good indicator of performance when the optimal function values are very small.

2.6 Dynamic Optimization Problems

There are various dynamic optimization test functions proposed by researchers. In general, these functions are able to simulate real-world optimization problems and provide a simple mechanism to control the type of landscape dynamics. One of the earliest forms of dynamic optimization test problems use a number of standard static optimization problems and switch back and forth between these landscapes through the run of the evolutionary process [30].

Other forms of dynamic optimization problems use a number of competing peaks that are independently specified by their width, height, and location. Branke [17] suggested a general platform on which such type of test problems can be implemented. This platform is called *moving peaks problem* and is mathematically expressed as:

$$F(\vec{x},t) = \max(B(\vec{x}), \max_{i=1,\ldots,M} P(\vec{x}, h_i(t), w_i(t), \vec{p}_i(t))) \tag{2.4}$$

where $B(\vec{x})$ is a time-invariant "basis" landscape and P is the function defining the peak shape, where each of the m peaks has its own time-varying parameters: height (h), width (w), and location ($\vec{p}_i(t)$). When the peaks have a 'cone' shape, then the moving peaks problem will become *competing cones problem* (DF1 [5]). On the other hand, if the peaks have a 'Gaussian' shape, then the moving peaks problem becomes *time-varying Gaussian peaks problem* (DF2 [37]).

The most frequently used moving peaks problem was proposed by Morrison and De Jong [5]. They proposed using a number of competing cones each explicitly defined by its height, center, and width. These cone peaks are not differentiable at their peaks and

mimic real-world optimization problems that are justifiable for using evolutionary algorithms. They called their function DF1 (Dynamic Function 1) and it is expressed as:

$$f(\vec{x}) = \max_{i=1,m}\left[H_i - R_i \times \sqrt{\sum_{j=1}^{n}(x_j - X_{ij})^2} \right] \tag{2.5}$$

where $\vec{x} = (x_1,...,x_n)$ is a point in the landscape, *m* specifies the number of cones in the environment, and each cone *i* is independently specified by its height H_i, its slope R_i, and its center $\vec{X}_i = (X_{i1},...,X_{in})$.

For the remainder of this paper, we assigned similar nomenclature (DF2, DF3, etc) for the remaining test functions that we used to test the proposed dynamic evolutionary algorithm.

Another common moving peaks problem is the time-varying *n*-dimensional Gaussian peaks problem proposed by Grefenstette [37]. This problem is similar to that of the competing cones problem, but has 'Gaussian' peaks and is differentiable at the apex of the peaks. However, this problem becomes very challenging when the number of peaks is increased and provides a good benchmark evaluation for dynamic evolutionary algorithms. For the context of this chapter, this problem will be referred as DF2 and is mathematically expressed as:

$$f(X,t) = \max_{i=1,N}\left[A_i(t)\exp\left(\frac{-d(X,C_i(t))^2}{2\sigma_i^{\,2}(t)} \right) \right] \tag{2.6}$$

where $A_i(t)$ is the amplitude, $C_i(t)$ denotes the center and $\sigma_i(t)$ represents the width of the *n*-dimensional Gaussian peak.

Other forms of DOP test problems shift stationary optimization test problems using various dynamics of change. The common form of such types of problems is the moving parabola problem ([32, 33]). This problem has an objective function which has a general form as given below.

$$f(x,t) = Min\sum_{i=1}^{n}(x_i + \delta_i(t))^2 \tag{2.7}$$

The amount of shift in the landscape, $\delta_i(t)$, can have different dynamics of change and the common types are linear, random, and circular dynamics of change. In the

remainder of this chapter, moving parabola problems with these dynamics of change are referred to as DF3, DF4, and DF5 test problems respectively and are mathematically expressed as:

Linear translation [DF3]

$$\begin{aligned} &\delta_i(0) = 0 \qquad \forall i \in \{1,...,n\} \\ &\delta_i(t) = \delta_i(t-1) + s \end{aligned} \tag{2.8a}$$

Random dynamics [DF4]

$$\begin{aligned} &\delta_i(0) = 0 \qquad \forall i \in \{1,...,n\} \\ &\delta_i(t) = \delta_i(t-1) + s \times N_i(0,1) \end{aligned} \tag{2.8b}$$

Circular dynamics [DF5]

$$\begin{aligned} &\delta_i(0) = \begin{cases} 0 & : i \quad odd \\ s & : i \quad even \end{cases} \\ &\delta_i(t) = \delta_i(t-1) + s \times c(i,t) \end{aligned} \tag{2.8c}$$

where

$$c(i,t) = \begin{cases} \sin\left(\dfrac{2.\pi.t}{\gamma}\right) & : i \quad odd \\ \cos\left(\dfrac{2.\pi.t}{\gamma}\right) & : i \quad even \end{cases} \tag{2.8d}$$

γ determines the cycle length of the problem in moving parabola benchmark problems with circular dynamics. Since sinusoidal functions repeat themselves exactly at every period, DF5 will have a cycle accuracy of *100%. t* is used as index for the environmental state. Whenever the environment state changes, *t* is incremented by 1.

Another function that was used in Branke [8] is oscillating peaks function. This function has an oscillating weight function that causes oscillation in the base objective function. Due to this oscillation in the fitness landscape, the location of the optima also oscillates between various points. This test function has two landscapes with 10 peaks each. The parameters of each peak can be varied independently. In the remainder of this paper, this function will be referred as DF6 and is mathematically expressed as:

$$f_i(t) = Max\ w(t) f_i(0)$$
$$w(t) = \frac{1}{3}\cos(\frac{2t\pi}{steps} + 2\pi\frac{i-1}{l}) + \frac{2}{3}, \quad i = 1,...,l \tag{2.9}$$

where l is number of different landscapes that will be obtained from the base landscape $f_i(0)$ by multiplying with an oscillating weight function *w(t)*.

2.7 Proposed Dynamic Evolutionary Algorithm

The proposed dynamic evolutionary algorithm uses variable relocation vectors to adapt already converged or currently evolving individuals to the new environmental condition. The proposed algorithm relocates those individuals based on their change in function value due to the change in the environment and the average sensitivities of their decision variables to the corresponding change in the objective space. The relocation occurs during the transient stage of the evolutionary process and the algorithm reuses as much information as possible from the previous evolutionary history. As a result, the algorithm will have faster adaptation and convergence. In addition, the design is easier to implement and can be incorporated into standard evolutionary algorithms. A complete description of the proposed algorithm is presented in detail below.

Let $f = f(X, e)$ represent the dynamic optimization problem to be optimized; X represents the *n*-dimensional decision space vector and x^d represents the d^{th}-dimension decision variable. For the discussion of this section, a minimization dynamic optimization problem is assumed. Note that a maximization problem can be converted into a minimization problem by multiplying with -1.

Δx^d_{child} denotes the child's evolutionary progress in the d^{th}-dimension of the decision variable with respect to its parents. It is measured as the difference between the d^{th}-dimension decision variable of a child (x^d_{child}) and that of the centroid of its parents. This can be mathematically expressed as:

$$\Delta x^d_{child} = x^d_{child} - \frac{\left(x^d_{parent1} + x^d_{parent2}\right)}{2} \tag{2.10}$$

Δf_{child} is the evolutionary fitness progress of a child with respect to its parents and is measured as the difference between a child's fitness (f_{child}) and the interpolated fitness of

its parents. The interpolation is based on the distance between a child and its parents (ΔX_1 and ΔX_2). The farther a parent is away from its child, the lesser is its contribution to the interpolated fitness.

$$\Delta X_1 = \left\| X_{child} - X_{parent1} \right\| = \sqrt{\sum_{d=1}^{n} \left(x_{child}^{d} - x_{parent1}^{d} \right)^2} \tag{2.11a}$$

$$\Delta X_2 = \left\| X_{child} - X_{parent2} \right\| = \sqrt{\sum_{d=1}^{n} \left(x_{child}^{d} - x_{parent2}^{d} \right)^2} \tag{2.11b}$$

$$\Delta f_{child} = f_{child} - \left(\frac{\Delta X_2 \cdot f_{parent1} + \Delta X_1 \cdot f_{parent2}}{\Delta X_1 + \Delta X_2} \right) \tag{2.12}$$

The child's average evolutionary progress in the d^{th}-dimension decision variable (Δx_{av}^{d}) can be obtained as the weighted sum of the child's Δx_{child}^{d} and its parents' average decision progress ($_d\Delta x_{parent}^{av}$). The same is true for the child's average evolutionary fitness progress (Δf_{av}) except that we use the interpolated value of its parents' average fitness progress.

$$\Delta x_{av}^{d} = \frac{\Delta x_{child}^{d} + w \cdot nGen \cdot \left(\frac{{}_d\Delta x_{parent1}^{av} + {}_d\Delta x_{parent2}^{av}}{2} \right)}{w \cdot nGen + 1} \tag{2.13}$$

$$\Delta f_{av} = \frac{\Delta f_{child} + w \cdot nGen \cdot \left(\frac{\Delta X_2 \cdot \Delta f_{parent1}^{av} + \Delta X_1 \cdot \Delta f_{parent2}^{av}}{\Delta X_1 + \Delta X_2} \right)}{w \cdot nGen + 1} \tag{2.14}$$

where *nGen* denotes the total number of generations either from the start of an evolutionary process or the last occurrence of change, whichever is recent, up to the current generation. On the other hand, *w* represents the inertia given to previous evolutionary progresses relative to the current one. If all evolutionary progresses have equal weight, *w* = 1. Otherwise, *w* is set between 0 and 1. $_d\Delta x_{parent1}^{av}$ is Δx_{av}^{d} of the first parent which was calculated in the previous generation. Similarly, $\Delta f_{parent1}^{av}$ is Δf_{av} of the first parent which was calculated in the previous generation. The same is true for $_d\Delta x_{parent2}^{av}$ and $\Delta f_{parent2}^{av}$ of the second parent.

The total average evolutionary progress in the decision space of an individual can then be obtained as

$$\Delta X_{av} = \sqrt{\sum_{d=1}^{n} \left(\Delta x_{av}^{d}\right)^2} \tag{2.15}$$

The average sensitivity of the decision space to change in the objective space is defined as the ratio of the average evolutionary fitness progress and the average evolutionary progress in decision space. Mathematically:

$$S_X^{av} = \frac{\Delta f_{av}}{\Delta X_{av}} \tag{2.16}$$

The average sensitivity of the d^{th}-dimension of the decision space to change in the objective space can then be obtained as:

$${}_d S_x^{av} = S_X^{av} \cdot \frac{\Delta x_{av}^{d}}{\Delta X_{av}} = \Delta x_{av}^{d} \cdot \frac{\Delta f_{av}}{\sum_{q=1}^{n} \left(\Delta x_{av}^{d}\right)^2} \tag{2.17}$$

In dynamic optimization problems, the evolutionary fitness progress, Δf_i, can arise from changes in the decision space of an individual or changes in the environmental parameter. This can be approximately formulated as

$$\Delta f_i = \sum_{d=1}^{n} \left({}_d S_x^{av} \cdot \Delta x^{d} \right) + S_e \cdot \Delta e \tag{2.18}$$

where ${}_d S_x^{av}$ is the average sensitivity of the fitness to change in the d^{th}-dimension of the decision space; S_e is the average sensitivity of the individual's fitness to change in the environment; Δx^{d} and Δe are the corresponding changes in the d^{th}-dimension decision variable and the environmental parameter, respectively.

Under normal evolutionary process, the environmental parameter is constant, i.e., $\Delta e = 0$. Under such cases, the above equation reduces to

$$\Delta f_i = \sum_{d=1}^{n} \left({}_d S_x^{av} \cdot \Delta x_i^{d} \right) = S_X^{av} \cdot \Delta X_i \tag{2.19}$$

where ΔX_i is the average evolutionary progress in the decision space (ΔX_{av}) for individual *i*.

The equality of the last and middle terms in equation (2.19) can be proved by substituting equation (2.17) in the middle term and performing the summation.

Δe is different from zero during the transition period. But if we re-evaluate all the previous individuals, then all the changes in the decision variables become zero ($\Delta x^d = 0$). In this case, equation (2.18) can be written as:

$$\Delta f_i = f_i^{e2} - f_i^{e1} = S_e \cdot \Delta e \tag{2.20}$$

where $f_i^{e2} - f_i^{e1}$ represents the difference between the functional values of an individual in the new (with superscript, $e2$) and old (with superscript $e1$) environment, respectively.

The proposed algorithm estimates the required offsets in the decision variables that will match the fitness changes caused by the environment. This is done through the relocation vector, which is the anticipated uncertainty in the decision space of an individual. The relocation vector will relocate all individuals so that their fitness is restored or is further enhanced. It can be expressed as

$$\Delta X_i = \begin{cases} \dfrac{f_i^{e2} - f_i^{e1}}{S_X^{av}}, & f_i^{e2} \geq f_i^{e1} \\ \min\left\{ -\dfrac{f_{best}^{e2} - f_i^{e1}}{S_X^{av}}, -\dfrac{f_i^{e2} - f_i^{e1}}{S_X^{av}} \right\}, & f_i^{e2} < f_i^{e1} \end{cases} \tag{2.21}$$

where f_{best}^{e2} is the best fitness in the new environment.

The relocation offsets in each dimensions of the decision space can then be obtained as:

$$\Delta x_i^d = \frac{\Delta X_i \cdot_d S_x^{av}}{S_X^{av}} = \Delta X_i \cdot \frac{\Delta x_{av}^d}{\Delta X_{av}} \tag{2.22}$$

If $\left|\Delta x_i^d\right| > x_{\max}^d - x_{\min}^d$, then Δx_i^d is trimmed down to:

$$\Delta x_i^d = sign(\Delta x_i^d) \cdot (x_{\max}^d - x_{\min}^d) \tag{2.23}$$

where $x_{\max}^d$ and $x_{\min}^d$ are the maximum and minimum limits of the d^{th}-dimension decision variable respectively; and sign (z) is a function that returns the sign of z.

On the other hand, if Δx_i^d is less than $\Delta x_{\min}^d$ (minimum allowable relocation offset in the d^{th}-dimension decision variable), then

$$\Delta x_i^d = \Delta x_{\min}^d \tag{2.24}$$

The variable $\Delta x_{\min}^d$ is varied based on the diversity of the population just before change. After validation of Δx_i^d, the relocation algorithm will generate a number of offsprings as:

$$x_{i,new}^d = x_{i,old}^d + r \cdot \Delta x_i^d \tag{2.25}$$

where r is a random number between 0 and 1.

If the value of $x_{i,new}^d$ lies outside the interval [$x_{\min}^d$, $x_{\max}^d$], then the algorithm re-assigns $x_{i,new}^d$ as:

If $x_{i,new}^d > x_{\max}^d$,

$$x_{i,new}^d = x_{\max}^d + x_{i,old}^d - x_{i,new}^d = x_{\max}^d + r \cdot \Delta x_i^d$$

Else if $x_{i,new}^d < x_{\min}^d$,

$$x_{i,new}^d = x_{\min}^d + x_{i,old}^d - x_{i,new}^d = x_{\min}^d + r \cdot \Delta x_i^d$$

End

Figure 2.1 Re-assignment of $x_{i,new}^d$ when it lies outside the interval [$x_{\min}^d$, $x_{\max}^d$]

The best-fit individual out of a parent and its offsprings will then be passed on to the initial population of the new environment. This new initial population will be better adapted to the change and is claimed to converge quickly. After this initial population is obtained, the evolutionary process will proceed with its normal operation. In addition to the relocation vectors, the algorithm uses a small archive to store the best individuals obtained so far. When the archive is full, the oldest best individual will be replaced by the most recent one. The pseudo-code for the proposed dynamic evolutionary algorithm using variable relocation vector is given in Figures 2.2 and Figure 2.3.

```
Procedure  RVDEA
Begin
    k = 1
    nGen = 1
    Initialization
    Clear archive
    Evaluation
   While (not exit_condition) Do
    Begin
        Detection of change
       If  change  is detected
        Begin
            Transient_EA
            nGen = 1
            k = k + 1
        End
       Else
        Begin
            Selection
            Recombination
            Mutation
            Evaluation
            Replacement
            k = k + 1
            nGen = nGen + 1
        End
    End
End
```

Figure 2.2. Pseudo-code for the proposed dynamic evolutionary algorithm using variable relocation vectors

The *exit_condition* in the pseudo-code can be defined in several ways. Some of these are reaching maximum number of fitness evaluations, maximum number of generations and maximum number of changes. On the other hand, the counter k in the pseudo-code is used to keep track of the total number of generations. Similarly, *nGen* counts the number of generations but it is restarted from 1 every time a change occurs.

```
Procedure  Transient_EA
 Begin
     obtain average sensitivities of the decision variables to change in the
       landscape
     update archive (archive ← best individual)
     re-evaluate all individuals and obtain their functional changes due to
       the environment
     obtain relocation vectors for all individuals
     relocate all individuals a number of times
     select the best individual from a parent and its relocated offsprings
       and put it in the initial population of the new environment
     reset Δf_av and Δx_av^d values of all individuals
     reset change flag
 End
```

Figure 2.3. Pseudo-code for the transient evolutionary algorithm using variable relocation vectors

In general, a relocation vector lies between the minimum and maximum allowable shifts in the decision variables. Minimum relocation refers to cases where the fitness landscape is almost insensitive to shifts in the decision space. Maximum relocation, on the other hand, refers to cases where the fitness landscape is extremely sensitive to shifts in the decision space and corresponds to random initialization since the designated relocation can pick up any of the allowable values in the decision space. Other intermediate values of relocation vectors will try to introduce certain radius of uncertainty over the decision space in which the proposed algorithm will look for better individuals. This way of relocation formulation will allow the algorithm to treat dynamic optimization problems without regard to their dynamics of change. If the dynamics of change is homogenous, then the relocation values of all individuals will have the same value. If the dynamics of change is heterogeneous and deterministic, the relocation values will vary from individual to individual and the relocation amount will have a deterministic nature. Lastly, if the dynamics of change have a random nature, the relocation vectors will account for the changes in the fitness landscape by taking average sensitivity values over the evolutionary run and this will allow treating the changes stochastically.

Furthermore, since the population is adapted from the previous population, there is a higher reuse of previous evolutionary data which provides faster convergence. The

relocation vectors are specific to each individual and this gives the algorithm better adaptation than those approaches that use a single adaptation value for the whole population, such as [30]. As a technique to be used at transient periods, the proposed algorithm provides the next evolutionary cycle with better initial population than any other randomly generated population. As a result, there will be a considerable progress jump for the upcoming evolutionary process and this gives the proposed algorithm faster adaptation and convergence.

2.8 Summary

Dynamic optimization problems are common types of uncertainty problems in real-world optimization problems. In recent years, it has become a critical need to account for the dynamism of the evolutionary optimization problems in the evolutionary algorithm. Various methodologies have been suggested to adapt stationary evolutionary algorithms into dynamic evolutionary algorithms. In earliest researches, a complete restart of the evolutionary process is invoked when a change occurs. Other techniques try to maintain the diversity of the population in the entire run and thus allow the population to explore and adapt to the new environment whenever a change occurs. Allocating some memory to store the best individuals found so far in the recent periods of change is also a commonly used approach in dynamic evolutionary algorithms. Multiple population approaches are also used to solve DOPs. They use several sub-populations to adapt to the new environment and there are different variations under this type of approaches. The last commonly used technique adapts the population to the new environment by using either increased levels of mutation or self-adaptation mechanisms.

The proposed algorithm in this chapter is inspired from the comparatively small computational cost of self-adaptation schemes and also from the idea to have a better technique that can be used alongside other approaches for solving dynamic optimization problems. In this spirit, it can be categorized under self-adaptation schemes. The algorithm also utilizes a small memory to further improve its performance in cyclic changes. The unique attribute about this algorithm is that the self-adaptation scheme and the relocation amount are specific to each individual, and this allows the algorithm to

provide better adapted initial population to the new environment. There are two variations of the proposed algorithm – the first is *RVDEAmem* which is a relocation vector dynamic evolutionary algorithm enhanced with memory, and the second is *RVDEAclusters* which is a relocation vector based dynamic evolutionary algorithm which is enhanced with memory and several clusters to provide superior performance over a relative increase in the computational cost.

The relocation vectors are specific to each individual and this gives the algorithm better adaptation than those approaches that use a single adaptation value for the whole population. As a technique to be used at transient periods, the proposed algorithm provides the next evolutionary cycle with better initial population which results in faster convergence. Furthermore, the extra computational cost of the proposed algorithm is comparable to its performance improvement as the additional calculations are basic arithmetic operations.

There are several performance indexes suggested to measure the performance of dynamic evolutionary algorithms and in this study, we use offline error performance index and adaptation error performance index.

There are a number of dynamic evolutionary test problems that have been used for testing dynamic evolutionary algorithms. These problems try to introduce dynamism in the location, width, and height of the optimal solutions. Some of the common types of problems include competing cones problem [5], time-varying Gaussian peaks problem [37], moving parabola problems [32, 33] and oscillating peaks problem [8].

CHAPTER III

CONSTRAINED MULTI-OBJECTIVE OPTIMIZATION

3.1 Introduction

Evolutionary algorithms were originally designed for solving unconstrained optimization problems, but in recent years, researchers have been able to tailor constraint handling techniques into these algorithms. The great challenges in constrained optimization problems arise from the various limits on the decision variables, the constraints involved, the interference among constraints, and the interrelationship between the constraints and the objective functions. In the mean time, researchers were also developing evolutionary approaches for solving multi-objective optimization problems (MOPs). These multi-objective evolutionary algorithms (MOEAs) are capable of simultaneously optimizing a set of competing objectives. Nevertheless, little research was conducted in the area of constrained multi-objective optimization (CMOP). Such problems involve multiple competing objectives that are subject to various equality and inequality constraints.

A constrained multi-objective optimization problem (CMOP) can be mathematically formulated as:

Minimize / Maximize

$$f_i(x) = f_i(x_1, x_2, ..., x_n), \quad i = 1,2,..., p \tag{3.1a}$$

Subject to $$g_i(x) = g_i(x_1, x_2, ..., x_n) < 0, \quad i = 1,..., q \tag{3.1b}$$

$$h_i(x) = h_i(x_1, x_2, ..., x_n) = 0, \quad i = q+1,..., m \tag{3.1c}$$

$$x_j^{\min} \le x_j \le x_j^{\max}, \quad j = 1,2,..., n \tag{3.1d}$$

There are p objective functions that are required to be optimized simultaneously. Each objective function $f_i(x)$ is defined on the search space $S \subseteq \Re^n$. Usually the search

space is an *n*-dimensional hyperbox in $\Re^n$. Each dimension of the search space is bounded by its upper ($x_j^{\max}$) and lower ($x_j^{\min}$) limits.

$g_i(x)$ is the i^{th}-inequality constraint, and $h_i(x)$ is the i^{th}-equality constraint. There are a total of *m* constraints: *q* inequality and *m* - *q* equality, which are required to be satisfied by the optimum solution. The presence of equality and inequality constraints will restrict our search space to a feasible space $F \subseteq S$, where a usable solution can be found.

This chapter extends the single-objective constrained evolutionary algorithm proposed by Tessema and Yen [38] to CMOPs. The proposed algorithm basically modifies the objective function of an individual using its distance measure and penalty value. These modified objective function values are ranked through the non-dominance sorting of the multi-objective optimization. Distance measures are found for each dimension of the objective space by incorporating the effect of an individual's constraint violation into its objective function. The penalty function, on the other hand, introduces additional penalty for infeasible individuals based on their objective values and constraint violations. The balance between the two components, one based on objective function and the other on constraint violation, is controlled by the number of feasible individuals currently present in the population. If few feasible individuals are present, then those infeasible individuals with higher constraint violations are more penalized than those with lower constraint violations. On the other hand, if sufficient number of feasible individuals exist, then those infeasible individuals with worse objective values are more penalized than those with better objective values. However, if the number of feasible individuals is in the middle of the two extremes, then the individual with lower constraint violation and better objective function is less penalized. The two components of the penalty function allow the algorithm to switch between feasibility and optimality at anytime during the evolutionary process. Furthermore, since priority is initially given to finding feasible individuals before searching for optimal solutions, the algorithm is capable of finding feasible solutions in cases where the feasible space is very small compared to the search space.

This chapter is structured as follows: Section 3.2 provides a brief overview of the various evolutionary approaches developed so far for constrained multi-objective

optimization problems. Next, in Section 3.3, the various performance indexes used to measure the performance of multi-objective evolutionary algorithms are presented. In Section 3.4, the various CMOP test problems are discussed and are used to evaluate the proposed algorithm. In Section 3.5, the proposed CMOP evolutionary algorithm is presented and analyzed in detail. Finally, we conclude with a summary of this chapter.

3.2 Literature Review

Over the last decade several MOEAs have been developed to solve multi-objective optimization problems. The earlier MOEAs are non-elitism based methods that assign fitness to population members based on non-dominated sorting. In addition, they exploit different techniques to preserve diversity among solutions of the same non-dominated front. Of these types, the Multi-Objective Genetic Algorithm (MOGA) [39] by Fonseca and Fleming's and the Non-dominated Sorting Genetic Algorithm (NSGA) [40] by Srinivas and Deb are very popular. MOGA uses the niche-formation technique to preserve diversity over the Pareto optimal region and sharing is performed on the objective function values. On the other hand, sharing is performed on the decision variable space for NSGA.

More recently, elitism based algorithms have been suggested to enhance the convergence properties of MOEAs. The Pareto Archived Evolution Strategy (PAES) [41] by Knowles and Corne uses a (1+1) evolution strategy together with a historical archive that records all the non-dominated solutions found until the current generation. It also designs a novel approach to maintain diversity which consists of a crowding procedure that divides objective space in a recursive manner into several grids. This procedure is adaptive and has lower computational complexity than the traditional niching based approaches. Zitzler and Thiele introduce the Strength Pareto Evolutionary Algorithm (SPEA) [42] that uses an external archive to preserve non-dominated solutions. In each generation, the non-dominated solutions in the external set will be given a strength value which is proportional to the number of individuals they dominate. Fitness of individuals in the main population will be computed according to the strengths of all external non-dominated solutions that dominated it. In addition, a clustering technique is used to

preserve diversity. Today, many advanced version of MOEAs have been constantly made available in literature in continually pursuing the performance frontier.

On the other hand, constraint handling for single objective optimization problems has also been actively researched over the past two decades. Penalty functions are the simplest and the most commonly used methods for handling constraints using EAs. In death penalty function methods such as [43], individuals that violate any one of the constraints are completely rejected and no information is extracted from infeasible individuals. If the penalties added do not depend on the current generation number and remain constant during the entire evolutionary process, then the penalty function is called static penalty function. In static penalty function methods, the penalties are the weighted sum of the constraint violations. If, alternatively, the current generation number is considered in determining the penalties, then the method is called dynamic penalty function method [44]. In adaptive penalty function methods [45-47], information gathered from the search process will be used to control the amount of penalty added to infeasible individuals.

In [44, 48], methods based on preference of feasible solutions over infeasible solutions are employed. In these types of techniques, feasible solutions are always considered better than infeasible ones. Therefore, when population fitness ranking is performed, feasible individuals will come first followed by infeasible individuals with low constraint violation. In [49], Runarsson and Yao introduce the stochastic ranking method to achieve a balance between objective and penalty functions stochastically. A probability factor is used to determine whether the objective function value or the constraint violation value determines the rank of each individual. In [50-51], similar algorithms are proposed where constraint violation and objective function are optimized separately.

More recently, multi-objective optimization techniques have been used to solve constrained optimization problems. In [52], a multi-objective optimization technique that uses population-based algorithm generator and infeasible solutions archiving and replacement mechanism is introduced. In [53], a two-phase algorithm that is based on multi-objective optimization technique is proposed. In the first phase of the algorithm, the objective function is completely disregarded and the constraint optimization problem is

treated as a constraint satisfaction problem. In the second phase, both constraint satisfaction and objective optimization are treated as a bi-objective optimization problem. An algorithm that combines penalty function approach and multi-objective optimization technique is also proposed in [54]. The algorithm has a similar structure as the penalty-based approach but borrows the ranking scheme from multi-objective optimization techniques.

Although multi-objective optimization and constraint handling have received a lot of attention individually, very little research has been done in solving constrained multi-objective optimization problems. Coello and Christiansen [55] propose a naïve approach to solve CMOPs by ignoring any solution that violates any of the assigned constraints. This method is very easy to implement but it often experiences difficulty in searching for even a single feasible solution.

In [56], Binh and Korn propose the Multi-objective Evolution Strategy (MOBES), which takes into account the objective function vector as well as the degree of constraint violation of infeasible solutions in order to evaluate their fitness. Infeasible individuals will be divided into different classes according to their "nearness" to the feasible region and ranking will be performed based on the class. In addition, a mechanism to maintain a feasible Pareto optimal set is employed.

In [57], Deb, *et al*. propose a constrained multi-objective algorithm based on constrained dominance of individuals. According to their algorithm, a solution i is said to constrained-dominate a solution j if (1) i is feasible while j is infeasible; (2) both are infeasible and i has less constraint violation; or 3) both are feasible and i dominates j. Feasible solutions constrained-dominate all infeasible solutions. However, when two feasible individuals are compared, the usual dominance relationship is used. The level of constraint violation is used to compare two infeasible individuals.

In [58], Jimenez, *et al*. propose the Evolutionary algorithm of Non-dominated Sorting with Radial Slots (ENORA), which employs the min-max formulation for constraint handling. Feasible individuals evolve towards optimality, while infeasible individuals evolve towards feasibility. In addition, a diversity technique based on partitioning the search space in a set of radial slots along which the successive populations generated by the algorithm are positioned is introduced.

In [59], Ray, *et al*. suggest using three different non-dominated rankings of the population. The first ranking is performed using the objective function values, the second is performed using the different constraints, and the last ranking is based on the combination of all objective functions and constraints. Depending on these rankings, the algorithm performs according to the predefined rules.

In [60], Chafekar, *et al*. propose two novel approaches for solving constrained multi-objective optimization problems. One method, called Objective Exchange Genetic Algorithm of Design Optimization (OEGADO), runs several GAs concurrently with each GA optimizing one objective and exchanging information about its objective with others. The other method, called Objective Switching Genetic Algorithm for Design Optimization (OSGADO), runs each objective sequentially with a common population for all objectives.

In light of superior performance achieved in [38] for the single objective constraint optimization, a similar idea is extended in this chapter into the uses of multi-objective constraint optimization. In the next section, we introduce the proposed constrained multi-objective evolutionary algorithm.

3.3 Performance Indexes for Multi-Objective Evolutionary Algorithms

The performance of the algorithm is measured using the convergence and diversity metrics. These metrics can be obtained as follows [57].

3.3.1 *Convergence metric* -

The convergence metric can be obtained by calculating the smallest normalized Euclidean distance between the non-dominated set and the true Pareto-front [57]. In actual implementation, a set of uniformly distributed sample points are taken from the true Pareto-front and then for each point k in the non-dominated set, the smallest distance from the true Pareto-front to point k is calculated as follows.

$$L_k = \min_{j \in T} \sqrt{\sum_{i=1}^{p} \left(\frac{f_i(k) - f_i(j)}{f_i^{\max} - f_i^{\min}} \right)^2} \tag{3.2}$$

Here, T is the set of points in the true Pareto-front and $f_i^{\max}$ and $f_i^{\min}$ are the maximum and minimum function values of the i^{th} objective function in the true Pareto-front. p is the

total number of objective functions in the problem. The convergence metric will then equals to the normalized average of minimum distances to the true Pareto-front from all individuals in the non-dominated set. Mathematically,

$$\gamma = \frac{\sum_{k \in P^*} L_k}{|P^*|} \tag{3.3}$$

where P^* is the non-dominated set, $|P^*|$ denotes the total number of individuals in the set P^* and γ is the convergence metric for the set P^*.

3.3.2 *Diversity Metric* - The diversity metric measures the extent of spread achieved among the obtained solutions. This metric is calculated as [57]:

$$\Delta = \frac{L_f + L_l + \sum_{k=1}^{N-1} \left| L_i - \overline{L} \right|}{L_f + L_l + (N-1)\overline{L}} \tag{3.4}$$

where L_f and L_l are the extreme solutions in the non-dominated set, and $\overline{L}$ is the average of all distances $L_i, i = 1,2,...,(N-1)$ assuming that there are (*N-1*) consecutive distances.

3.4 Constrained Multi-Objective Optimization Test Problems

Several constrained multi-objective test problems that have been proposed by researchers. Those test problems used in this thesis are presented below. Note that all the test problems given below are minimization problems.

Test Problem OSY [61]

Minimize

$$f_1(X) = -[25(x_1 - 2)^2 + (x_2 - 2)^2 + (x_3 - 1)^2 + (x_4 - 4)^2 + (x_5 - 1)^2],$$
$$f_2(X) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2, \tag{3.5}$$

subject to

$$
\begin{aligned}
C_1(X) &\equiv x_1 + x_2 - 2 \geq 0, \\
C_2(X) &\equiv 6 - x_1 - x_2 \geq 0, \\
C_3(X) &\equiv 2 - x_2 + x_1 \geq 0, \\
C_4(X) &\equiv 2 - x_1 + 3x_2 \geq 0, \\
C_5(X) &\equiv 4 - (x_3 - 3)^2 - x_4 \geq 0, \\
C_6(X) &\equiv (x_5 - 3)^2 + x_6 - 4 \geq 0,
\end{aligned}
\tag{3.6a}
$$

and

$$
\begin{aligned}
&0 \leq x_1, x_2, x_6 \leq 10 \\
&1 \leq x_3, x_5 \leq 5, \\
&0 \leq x_4 \leq 6.
\end{aligned}
\tag{3.6b}
$$

Test Problem BNH [56]

Minimize

$$
\begin{aligned}
f_1(X) &= 4x_1^2 + 4x_2^2, \\
f_2(X) &= (x_1 - 5)^2 + (x_2 - 5)^2,
\end{aligned}
\tag{3.7}
$$

subject to

$$
\begin{aligned}
C_1(X) &\equiv (x_1 - 5)^2 + x_2^2 \leq 25, \\
C_2(X) &\equiv (x_1 - 8)^2 + (x_2 + 3)^2 \geq 7.7,
\end{aligned}
\tag{3.8a}
$$

and

$$
\begin{aligned}
&0 \leq x_1 \leq 5, \\
&0 \leq x_2 \leq 3.
\end{aligned}
\tag{3.8b}
$$

Test Problem SRN [40] [62]

Minimize

$$
\begin{aligned}
f_1(X) &= 2 + (x_1 - 2)^2 + (x_2 - 1)^2, \\
f_2(X) &= 9x_1 - (x_2 - 1)^2,
\end{aligned}
\tag{3.9}
$$

subject to

$$
\begin{aligned}
C_1(X) &\equiv x_1^2 + x_2^2 \leq 225, \\
C_2(X) &\equiv x_1 - 3x_2 + 10 \leq 0,
\end{aligned}
\tag{3.10a}
$$

and

$$
\begin{aligned}
&-20 \leq x_1 \leq 20, \\
&-20 \leq x_2 \leq 20.
\end{aligned}
\tag{3.10b}
$$

Test Problem TNK [63]

Minimize

$$f_1(X) = x_1,$$
$$f_2(X) = x_2, \tag{3.11}$$

subject to

$$C_1(X) \equiv x_1^2 + x_2^2 - 1 - 0.1\cos\left(16\arctan\frac{x_1}{x_2}\right) \geq 0$$
$$C_2(X) \equiv (x_1 - 0.5)^2 + (x_2 - 0.5)^2 \leq 0.5, \tag{3.12a}$$

and

$$0 \leq x_1 \leq \pi,$$
$$0 \leq x_2 \leq \pi. \tag{3.12b}$$

Test Problem CTP1 [64]

Minimize

$$f_1(X) = x_1,$$
$$f_2(X) = (1 + x_2) \cdot e^{-\frac{f_1(X)}{1+x_2}}, \tag{3.13}$$

subject to

$$C_1(X) \equiv f_2(X) - 0.858 \cdot e^{-0.541 f_1(X)} \geq 0,$$
$$C_2(X) \equiv f_2(X) - 0.728 \cdot e^{-0.295 f_1(X)} \geq 0, \tag{3.14a}$$

and

$$0 \leq x_1 \leq 1,$$
$$0 \leq x_2 \leq 1. \tag{3.14b}$$

Test Problem CTP2-CTP8 [64]

Minimize

$$f_1(X) = x_1,$$
$$f_2(X) = (1 + x_2) \cdot \left(1 - \frac{f_1(X)}{1 + x_2}\right) \tag{3.15}$$

subject to

$$C_1(X) \equiv \cos(\theta)[f_2(X) - e] - \sin(\theta) f_1(X)$$
$$\geq a\left|\sin\left\{b\pi[\sin(\theta).(f_2(X) - e) + \cos(\theta).f_1(X)]^c\right\}\right|^d \tag{3.16a}$$

and

$$0 \leq x_1 \leq 1,$$
$$0 \leq x_2 \leq 1. \tag{3.16b}$$

CTP2

$$C_1: \quad \theta = -0.2\pi, \quad a = 0.2, \quad b = 10, \quad c = 1, \quad d = 6, \quad e = 1 \tag{3.17a}$$

CTP3

$$C_1: \quad \theta = -0.2\pi, \quad a = 0.1, \quad b = 10, \quad c = 1, \quad d = 0.5, \quad e = 1 \tag{3.17b}$$

CTP4

$$C_1: \quad \theta = -0.2\pi, \quad a = 0.75, \quad b = 10, \quad c = 1, \quad d = 6, \quad e = 1 \tag{3.17c}$$

CTP5

$$C_1: \quad \theta = -0.2\pi, \quad a = 0.1, \quad b = 10, \quad c = 2, \quad d = 0.5, \quad e = 1 \tag{3.17d}$$

CTP6

$$C_1: \quad \theta = 0.1\pi, \quad a = 40, \quad b = 0.5, \quad c = 1, \quad d = 2, \quad e = -2 \tag{3.17e}$$

CTP7

$$C_1: \quad \theta = -0.05\pi, \quad a = 40, \quad b = 5, \quad c = 1, \quad d = 6, \quad e = 0 \tag{3.17f}$$

CTP8

$$\begin{aligned} &C_1: \quad \theta = 0.1\pi, \quad a = 40, \quad b = 0.5, \quad c = 1, \quad d = 2, \quad e = -2 \\ &C_2: \quad \theta = -0.05\pi, \quad a = 40, \quad b = 2, \quad c = 1, \quad d = 6, \quad e = 0 \end{aligned} \tag{3.17g}$$

Test Problem CONSTR [64]

Minimize

$$\begin{aligned} f_1(X) &= x_1, \\ f_2(X) &= \frac{1 + x_2}{x_1}, \end{aligned} \tag{3.18}$$

subject to

$$\begin{aligned} &C_1(X) \equiv x_2 + 9x_1 \geq 6, \\ &C_2(X) \equiv -x_2 + 9x_1 \geq 1, \\ &0.1 \leq x_1 \leq 1, \\ &0 \leq x_2 \leq 5. \end{aligned} \tag{3.19}$$

Test Problem Welded Beam [60]

Minimize

$$\begin{aligned} f_1(X) &= 1.10471 x_1^2 x_3 + 0.04811 x_4 x_2 (x_3 + 14), \\ f_2(X) &= \frac{2.1952}{x_4^3 x_2}, \end{aligned} \tag{3.20}$$

subject to

$$
\begin{aligned}
&C_1(X) \equiv 13600 - \tau \geq 0, \\
&C_2(X) \equiv 30000 - \sigma \geq 0,, \\
&C_3(X) \equiv x_2 - x_1 \geq 0 \\
&C_4(X) \equiv P_c - 6000 \geq 0, \\
&0.125 \leq x_1, x_2 \leq 5, \\
&0.1 \leq x_3, x_4 \leq 10,
\end{aligned}
\tag{3.21a}
$$

where

$$
\begin{aligned}
&\tau = \sqrt{(\tau')^2 + (\tau'')^2 + \frac{x_3 \tau' \tau''}{\sqrt{0.25\left(x_3^2 + (x_1 + x_4)^2\right)}}}, \\
&\tau' = \frac{6000}{\sqrt{2} x_1 x_3}, \\
&\tau'' = \frac{6000(14 + 0.5 x_3)\sqrt{0.25\left(x_3^2 + (x_1 + x_4)^2\right)}}{2\sqrt{2} x_1 x_3 \dfrac{x_3^2}{12 + 0.25(x_1 + x_4)^2}}, \\
&\sigma = \frac{504000}{x_4^2 x_2}, \\
&P_c = 64746.022(1 - 0.0282346 x_4) x_4 x_2^3.
\end{aligned}
\tag{3.21b}
$$

3.5 Proposed Constrained Multi-Objective Evolutionary Algorithm

The proposed algorithm extends the single-objective constrained evolutionary algorithm proposed by Tessema and Yen [38] into the multi-objective case. The major difference in various constraint handling techniques used in multi-objective optimization arises from the variations in the involvement of infeasible individuals in the evolutionary process. The main purpose of involving infeasible individuals in the search process is to exploit the information they carry. Since EAs are stochastic search techniques, discarding infeasible individuals might lead to the EA being stuck in local optima, especially in problems with discontinuous search space. In addition, in some highly constrained optimization problems, finding a single feasible individual by itself might be a daunting challenge when the algorithm has to be able to extract information from the previous infeasible individuals.

The proposed algorithm uses modified objective function values for checking dominance in the population. The modification is based on the constraint violation of the

individual and its objective performance. The modified objective value has two components: distance measure and adaptive penalty function. The two components are discussed below in detail. Without loss of generality, the discussion of this chapter assumes minimization problems. A maximization problem can be easily converted into minimization problem by multiplying with -1.

3.5.1 *Distance Values* - Distance measures are found for each dimension of the objective space by including the effect of an individual's constraint violation into its objective function. The major steps in calculating the distance measure are discussed below. First, obtain the minimum and maximum values of each objective function in the population.

$$f_{\min}^{i} = \min_{x} f_i(x) \tag{3.22a}$$

and

$$f_{\max}^{i} = \max_{x} f_i(x) \tag{3.22b}$$

Then using these values, normalize each objective function *i* for every individual *k*.

$$\widetilde{f}_k^{\,i}(x) = \frac{f_k^i(x) - f_{\min}^i}{f_{\max}^i - f_{\min}^i} \tag{3.23}$$

where $\widetilde{f}_k^{\,i}(x)$ is the normalized i^{th}-objective value of individual *k* with decision variable *x*.

Constraint violation, $v_k(x)$, of each individual *k* is then calculated as the summation of the normalized violations of each constraint divided by the total number of constraints,

$$v_k(x) = \frac{1}{m}\sum_{j=1}^{m} \frac{c_k^j(x)}{c_{\max}^j}, \tag{3.24}$$

where

$$c_k^j(x) = \begin{cases} \max(0, g_j(x_k)) & j = 1, \cdots, q \\ \max(0, |h_j(x_k)| - \delta) & j = q+1, \cdots, m \end{cases} \tag{3.25a}$$

$$c_{\max}^j = \max_{x} c_k^j(x). \tag{3.25b}$$

δ is a tolerance value for equality constraints (usually 0.001 or 0.0001). q is the number of inequality constraints, and $m - q$ is the number of equality constraints. If the constraint violation $c_k^j(x)$ is greater than zero, then individual k violates the j^{th}-constraint. On the other hand, if the constraint violation $c_k^j(x)$ is equal to zero, then the individual k satisfies the j^{th}-constraint and the constraint violation $c_k^j(x)$ is set to zero.

Then the "distance" value of individual k in each objective function dimension i is formulated as follows:

$$d_k^i(x) = \begin{cases} v_k(x), & \text{if } r_f = 0 \\ \sqrt{\widetilde{f}_k^i(x)^2 + v_k(x)^2}, & \text{otherwise} \end{cases}, \tag{3.26}$$

where

$$r_f = \frac{\text{number of feasible individuals in current population}}{\text{population size}}. \tag{3.27}$$

The pseudo-code for calculating the distance value is given in Figure 3.1.From Equation (3.26), we can observe that if there is no feasible individual in the current population, then the distance values are equal to the constraint violation of the individual. In this case, according to the distance values, an infeasible individual with smaller constraint violation will dominate another infeasible individual with higher constraint violation irrespective of their objective function values. This is an intuitively reasonable way to compare infeasible individuals in the absence of feasible individuals and it will help us approach the feasible space very quickly.

On the other hand, if there is more than one feasible solution in the population, then the distance values will have the properties summarized below:

A. For a feasible individual k, the distance value in a given objective function dimension i is equal to $\widetilde{f}_k^i(x)$. Hence, those feasible individuals with smaller objective function value will have smaller distance value in that given dimension.
B. For infeasible individuals, the distance value has two components: the objective function value and the constraint violation. Hence, individuals closer to the origin in the $\widetilde{f}^i(x) - v(x)$ space would have lower distance value in that objective function dimension than those farther away from the origin.

Input: $f_k^i(x)$, $v_k(x)$, $f_{\min}^i$, $f_{\max}^i$, r_f

$\forall k, k = 1, \ldots,$ *Population Size*

$\forall i, i = 1, \ldots,$ *number of objectives*

Output: $d_k^i(x)$ $\forall k, k = 1, \ldots,$ *Population Size*

$\forall i, i = 1, \ldots,$ *number of objectives*

Begin

If $r_f = 0$ **then**

For $i = 1$ to *number of objectives* **Do**

For $k = 1$ to *Population Size* **Do**

$d_k^i(x) \leftarrow v_k(x)$

End For

End For

Else

For $i = 1$ to *number of objectives* **Do**

For $k = 1$ to *Population Size* **Do**

$$\tilde{f}_k^i(x) \leftarrow \frac{f_k^i(x) - f_{\min}^i}{f_{\max}^i - f_{\min}^i}$$

$$d_k^i(x) \leftarrow \sqrt{\tilde{f}_k^i(x)^2 + v_k(x)^2}$$

End For

End For

End If

End

Figure 3.1. Pseudo code for finding distance value

C. If we compare the distance values of infeasible and feasible individuals, then either one may have a smaller value. But if the two individuals have similar objective function value, then the feasible individual will have smaller distance value in the corresponding objective dimension.

3.5.2 *Two Penalties* - In addition to the penalty imposed upon infeasible individuals by the distance measure, two other penalty functions are also added. These functions introduce additional penalty for infeasible individuals based on their corresponding objective value and constraint violation. The first penalty function is based on the objective functions and the second one is based on the constraint violation. The balance

between the two components is controlled by the number of feasible individuals currently present in the population.

These penalties have two major purposes:

A. To further reduce the fitness of infeasible individuals as the penalty imposed by the distance formulation alone is small.
B. To identify the best infeasible individuals in the population by adding different amount of penalty to each infeasible individual's fitness.

The two penalties are formulated for individual k in the i^{th}-objective function dimension as follows:

$$p_k^i(x) = (1 - r_f) X_k(x) + r_f Y_k^i(x), \tag{3.28}$$

where

$$X_k(x) = \begin{cases} 0, & \text{if } r_f = 0 \\ v_k(x), & \text{otherwise} \end{cases}, \tag{3.29a}$$

and

$$Y_k^i(x) = \begin{cases} 0, & \text{if } x_k \text{ is a feasible individual} \\ \tilde{f}_k^i(x), & \text{if } x_k \text{ is an infeasible individual} \end{cases}. \tag{3.29b}$$

From the penalty function definition in Equations (3.28-3.29), we can observe that if the feasibility ratio of the population is small (but not zero), then the first penalty ($X_k(x)$) will have more impact than the second penalty ($Y_k^i(x)$). The first penalty is formulated to have large value for individuals with large amount of constraint violation. Hence in the case when there are few feasible individuals present in the population (r_f is small), infeasible individuals with higher constraint violation will be more penalized than those with lower constraint violation. On the other hand if there are many feasible solutions in the population (r_f is large), the second penalty will have more effect than the first one. In this case, infeasible individuals with larger objective function value will be more penalized than infeasible individuals with smaller objective function value. If there are no feasible individuals in the population ($r_f = 0$), both penalties will be zero.

The two components of the penalty function allow the algorithm to switch between finding more feasible solutions and finding better solutions at anytime during the evolutionary process. Furthermore, since priority is initially given to searching for feasible individuals, the algorithm is capable of finding feasible solutions in cases where the feasible space is small or discontinuous compared to the search space.

The pseudo-code for calculating the penalty value is given below in Figure 3.2.

Input: $f_k^i(x)\ , v_k(x), r_f$
$\forall k, k = 1, \ldots,$ *Population Size*
$\forall i, i = 1, \ldots,$ *number of objectives*
Output: $p_k^i(x)\ \ \forall k, k = 1, \ldots,$ *Population Size*
$\forall i, i = 1, \ldots,$ *number of objectives*

Begin
For $i = 1$ to *number of objectives* **Do**
For $k = 1$ to *Population Size* **Do**
If $r_f = 0$ **then**
$X_k(x) \leftarrow 0$
Else
$X_k(x) \leftarrow v_k(x)$
End If

If $v_k(x) = 0$ **then**
$Y_k^i(x) \leftarrow 0$
Else
$Y_k^i(x) \leftarrow \tilde{f}_k^i(x)$
End If
$p_k^i(x) \leftarrow (1 - r_f) X_k(x) + r_f Y_k^i(x)$
End For
End For
End

Figure 3.2. Pseudo code for finding penalty value

3.5.3 *Final modified objective value formulation* - The final modified objective value of individual *k*, using which non-dominance sorting is performed, is formulated as the sum of the distance measure and penalty function in the i^{th}-objective dimension.

$$F_k^i(x) = d_k^i(x) + p_k^i(x) \tag{3.30}$$

This modified objective value formulation is very flexible and will allow us to utilize infeasible individuals efficiently. Most constraint optimization algorithms in literature are "rigid" in a sense that they always prefer certain types of infeasible individuals. For example, they might always give priority to those individuals with small constraint violation only or those individuals with low objective value only. But according to our new fitness formulation, the infeasible individuals that are considered valuable are not always similar. Here are some of the interesting properties of this modified objective value formulation:

A. If there is no feasible individual in the current population, each $d_k^i(x)$ will be equal to the constraint violation ($v_k(x)$) and each $p_k^i(x)$ term will be zero. In this case, the objective values of the individuals will be totally disregarded, and all individuals will be compared based only on their constraint violation. This will help us find feasible individuals before looking for optimal solutions.
B. If there are feasible individuals in the population, then individuals with both low objective function values and low constraint violation value will be preferred than those individuals with high objective function values or high constraint violation or both.
C. If two individuals have equal or very close distance values, then the penalty value ($p^i(x)$) determines the dominant individual. According to our penalty function, if the feasibility ratio (r_f) in the population is small, then the individual closer to the feasible space will be dominant. In the other case, the individual with smaller objective function values will be dominant. Otherwise, the two individuals will be non-dominant solutions.
D. If there is no infeasible individual in the population ($r_f = 1$), then individuals will be compared based on their objective function values alone.

The general pseudo-code for the proposed algorithm is given in Figure 3.3.

/ Proposed Constrained Multi-Objective Evolutionary Algorithm */*

Begin

Initialize *N* solutions

Evaluate all individuals

/ Constraint satisfaction */*

Do While ("***no feasible solution is found***" or "***maximum generation is reached***")

1) Give fitness to individuals based on their sum of constraint violations

2) Rank individuals based on fitness in 1

3) Selection, Recombination, Mutation and Replacement

4) Archive if any feasible solutions is found

End Do */* Feasible solutions have been found */*

/ Constraint satisfaction and objective optimization */*

Do While ("***maximum generation is reached***")

5) Calculate modified objective function values using distance measures and penalty functions for all individuals

6) Pareto sort individuals according to their modified objective function values

7) Give fitness to individuals according to Pareto ranking and crowding distance

8) Use tournament selection to select *N* parents

9) Generate *N* offspring solutions

10) Calculate fitness of offspring solutions

11) Update archive. If a feasible offspring dominates a solution in the archive, then it will replace that solution.

12) Trim the main population to *N* individuals based on the fitness of the individuals

End Do

End

Output archive

Figure 3.3. Pseudo-code of proposed algorithm

After the computation of the modified objective values, the standard features of NSGA-II, such as non-dominant ranking and diversity through crowding distances, will be used based on these modified values. During the archiving process, the best-feasible individuals are given priority than any infeasible individuals as the goal of constrained multi-objective optimization is eventually to find feasible optimal solutions.

3.6 Summary

This chapter focuses on the issue of constrained multi-objective optimization. First, the problem definition is presented in Section 3.1. Following that in Section 3.2, a review of the various evolutionary algorithms suggested to solve constrained multi-objective optimization problems is presented. The literature review is followed by the discussion about the performance indexes often used to measure the performance of multi-objective problems. There are two commonly used performance indexes: diversity and convergence metrics. The diversity metric measures how well the resulting solution is distributed over the Pareto front. The convergence metric, on the other hand, measures how close the resulting solutions are to the true Pareto-optimal solutions. Then in Section 3.4, the different constrained multi-objective optimization problems are presented. The test problems used in this paper include CONSTR [64], SRN [40, 62], BNH [56], OSY [61], TNK [63], CTP1-CTP8 [64], and Welded-beam problems [60].Finally in this chapter, we propose a constraint handling technique for solving constrained multi-objective optimization problems. The proposed algorithm is based on adaptive penalty functions and distance measures. These two functions are dependent upon the objective function values and the sum of constraint violations of an individual. Through this design, the objective space is modified to account for the performance and constraint violation of each individual. The modified objective functions are used in the non-dominance sorting to facilitate in evolution of optimal solutions not only in the feasible space but also in the infeasible space. The search in the infeasible space is designed to exploit those individuals with better objective values and lower constraint violations. The number of feasible individuals in the population is used to guide the search process either toward finding more feasible solutions or favor in search for optimal solutions. The proposed method is simple to implement and does not need any parameter tuning.

CHAPTER IV

EXPERIMENTAL SIMULATIONS AND RESULTS

4.1 Proposed Dynamic Evolutionary Algorithm

4.1.1 *Experimental setup* - The first set of experiments were conducted on dynamic benchmark problems, DF1 – DF6. Each test is run 50 times. We use a population size of 100, a crossover rate of 0.6, a mutation rate of 0.2, and a maximum fitness evaluation of 500,000 for all implementations. Each decision dimension is bounded between 0 and 100. In addition, we use SBX crossover and mutation. Tournament selection is adopted in recombination and replacement scheme. Elitism is also used to improve performance.

The proposed dynamic evolutionary algorithm is compared at least once against the following approaches for solving dynamic optimization problems. The first is standard evolutionary algorithm which is abbreviated as *SEA*. A variation of this algorithm with memory is denoted as *SEAmem*. The other algorithms that are used include introducing 25 random immigrants when a change occurs (*RI25* [12], *RI25mem*), self-organizing scouts (*SOS* [28]), standard evolutionary algorithm with three independent sub-populations (*P3* [8], *P3mem*), and finally dividing the population into memory and search subpopulations (*Mem/Search* [8]). In addition, a combination of *P3* and *RI25* approaches enhanced with memory, denoted as *P3RI25mem*, is also compared against the proposed algorithm. We used two variations of the proposed algorithm, RVDEA-one enhanced with memory (*RVDEAmem*) and another using several clusters to preserve diversity (*RVDEAcluster*).

Due to the lack of reported data for some of the algorithms under certain benchmark test functions, different test functions are compared against different sets of algorithms. DF1 test problem was tested on all of above listed algorithms. On the other hand, DF6 was tested for *SEAmem*, *RI25mem*, *P3mem*, *P3RI25mem*, *Mem/search*, and the two variations of the proposed algorithm. All the other test functions were tested on *SEAmem*,

RVDEAmem and *RVDEAcluster*. The first test was conducted on DF1 test problem with default parameter settings given in Table 4.1. Some of these values that apply were also used for the other test problems unless stated otherwise.

4.1.2 *Results and discussions* - Table 4.2 compares the performance of the proposed RVDEA with other state-of-the-art algorithms based on the number of peaks. As can be seen from Table 4.2, the proposed algorithm performs much better than the other dynamic algorithms except SOS. In the case of SOS, the proposed algorithm performed better only for a single peak DF1 problem. For the other cases, RVDEA provided comparable or lesser result. As RVDEA is a an adaptation scheme performed at the transient stage, other evolutionary techniques that enhance performance of algorithm, like diversity preservation, can be applied to further improve the steady state performance of RVDEA keeping the computational cost the same as SOS. Since the number of fitness evaluation is fixed when comparing algorithms, by computational cost we refer to the additional calculations required by the algorithm for its proper operation. For example, in the case of SOS, additional computation is required for forming and organizing the scouts. On the contrary, the additional evaluations required by RVDEA are simple averaging operations that are linearly dependent on the number of individuals in the population. This allows RVDEA to be used in conjunction with other techniques like steady state diversity preservation to further enhance the algorithm's performance. In this notion, we tested a modified RVDEA with clustering technique that has a comparable or less computational cost as that of SOS. In Table 4.2, it can be seen how well the modified algorithm's performance exceeded that of SOS in all different numbers of peak.

In Table 4.3, RVDEA was compared with the other algorithms based on frequency of change. RVDEA provides very good results at higher frequency of change and a comparable result as SOS when frequency is decreased. Due to the structure of the algorithm, the maximum allowable change frequency is two generations – one for re-evaluation and one for relocation. Lower frequencies less than two generations can be analyzed by reducing the size of the original population so that RVDEA will have a

minimum change frequency of at least two while maintaining the number of evaluations constant.

Figures 4.1 and 4.2 provide a graphical presentation of the offline errors of the different dynamic EAs at varying peak number and varying frequency of change, respectively.

In Table 4.4, we present the outputs of RVDEA as the memory size is changed. Even though having memory is generally recommended in dynamic EAs, the algorithm still performs well without the support of memory.

We also have tested the algorithm using time-varying Gaussian peaks [DF2]. We used the same setting as in Table 4.1 for this problem. The results were compared with standard evolutionary algorithm with memory in Table 4.5. As can be clearly seen from the table, the proposed algorithm has better adaptation and performance even in higher number of peaks. RVDEA with memory provides very good results, but further improvements in performance can be obtained by using RVDEA with clusters. We also tested DF2 by varying the number of generations between changes. As can be seen from Table 4.6, the proposed algorithm provides better results at higher frequencies of change. As in the previous case, RVDEA with memory provides very good performance, but RVDEA with clusters provides much better results even though it involves more computation.

The algorithm is also tested for moving parabola test problem with linear [DF3], random [DF4], and circular [DF5] dynamics. We run the problems with different cycle length. The results show that the proposed algorithm performs very well both in lower and higher frequencies of change. The results are summarized in Table 4.7. Furthermore, the algorithm performs well in higher number of peaks as shown in Table 4.8.

The proposed algorithm was also tested on oscillating peaks function [DF6]. We used two landscapes with 10 peaks each. The minimum and maximum peak widths parameters are set to 0.001 and 0.08, respectively. The rest of the parameters are kept the same as in Table 4.1. In Table 4.9, the algorithm's performance under different cycle lengths is presented. As can be seen from the results, the algorithm has better performance both in lower and higher frequencies of change and the algorithm's performance was intact even with large variations in cycle lengths.

In Tables 4.10 and 4.11, we presented a summary of the adaptation performance for RVDEA with memory in the test problems DF1-DF6. As can be seen from the tables, the proposed algorithm adapts to the new environment effectively and quickly. This combination of qualities makes the proposed algorithm attractive to be used in environment with severe changes and higher frequencies of changes.

Similarly, in Tables 4.12 and 4.13, we presented the adaptation performance for RVDEA with clusters. The results obtained were better than those of RVDEA with memory, but as previously pointed out RVDEA with clusters involves more computation than RVDEA with memory alone. Generally speaking, the adaptation performance of RVDEA with memory and RVDEA with clusters are better and indicate the effectiveness of the adaptation scheme used by the proposed algorithm.

TABLE 4.1

DEFAULT EXPERIMENTAL PARAMETER SETTINGS FOR DYNAMIC BENCHMARK PROBLEMS

Default number of peaks	10
Default change frequency	Every 50 generation
Peak shape	Cone [DF1], Gaussian [DF2], Parabola [DF3-DF5], Bell Curve [DF6]
Dimension	5
Min and Max limit of each decision dimension	[0,100]
Height severity	7.0
Width severity	1.0
Min and Max peak height	[30,70]
Min and Max peak width	[1,12]
Peak shift length	1.0

TABLE 4.2

OFFLINE ERROR VARIATION AFTER 500,000 EVALUATIONS AS FUNCTION OF PEAK NUMBER ON MOVING CONE PEAKS BENCHMARK PROBLEM [DF1]

Peak no.	SEA	RI25 [12]	P3 [8]	SOS [28]	RVDEA mem	RVDEA clusters
1	3.69	9.29	3.45	2.06	1.23	1.02
10	17.98	14.67	14.47	4.01	4.88	3.54
20	20.06	13.93	15.62	4.43	5.68	3.87
30	20.27	12.93	14.39	4.20	5.86	3.92
40	19.50	12.45	14.57	4.06	5.65	3.49
50	19.70	12.74	13.78	4.12	5.21	3.78
100	17.91	11.21	11.49	3.75	4.98	3.37
200	18.13	10.85	10.66	3.62	4.92	3.54

Results for SEA, RI25, P3, SOS as reported in [8]

TABLE 4.3

OFFLINE ERROR VARIATION AFTER 500,000 EVALUATIONS AS FUNCTION OF GENERATION BETWEEN CHANGES ON MOVING CONE PEAKS BENCHMARK PROBLEM WITH 10 PEAKS [DF1]

Gen. no.	SEA	SEA mem	Mem/ search	SOS [28]	RVDEA mem	RVDEA clusters
2	24.59	25.22	18.74	15.62	15.82	12.91
5	22.44	22.16	14.54	8.59	8.89	7.67
10	21.07	20.81	11.95	6.51	7.21	6.048
25	19.12	19.79	9.41	4.93	5.35	4.28
50	17.93	18.23	7.74	4.01	4.88	3.54
100	17.06	17.53	6.58	3.62	4.12	3.14

Results for SEA, SEAmem, Mem/search and SOS as reported in [8]

TABLE 4.4

OFFLINE ERROR VARIATION AFTER 500,000 EVALUATIONS AS FUNCTION OF MEMORY SIZE ON MOVING CONE PEAKS BENCHMARK PROBLEM WITH 10 PEAKS [DF1]

Mem size	SEA mem	RI25 mem	P3 mem [8]	P3RI25 mem [8]	Mem/ search	RVDEA mem
0	n/a	n/a	n/a	n/a	n/a	4.90
4	17.94	13.27	14.53	17.88	7.76	5.15
10	18.23	13.60	14.45	18.73	7.34	4.88
16	17.91	13.64	14.49	21.21	7.46	5.06

Results for SEAmem, RI25mem, P3mem, P3RI25mem and Mem/search as reported in [8]

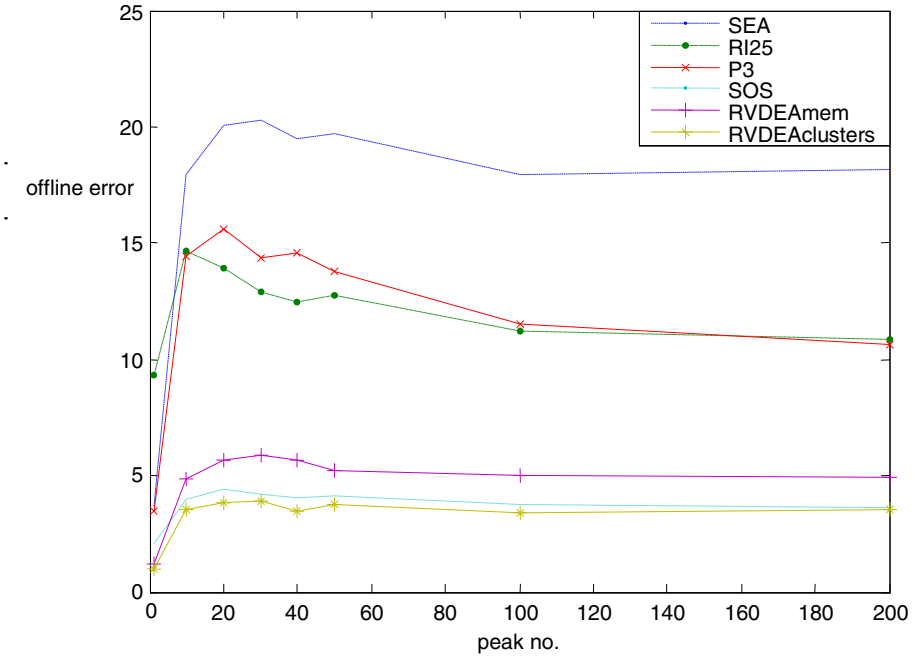


Figure 4.1. Offline error vs. peak number for some DEAs on a moving cone peaks benchmark problem [DF1]

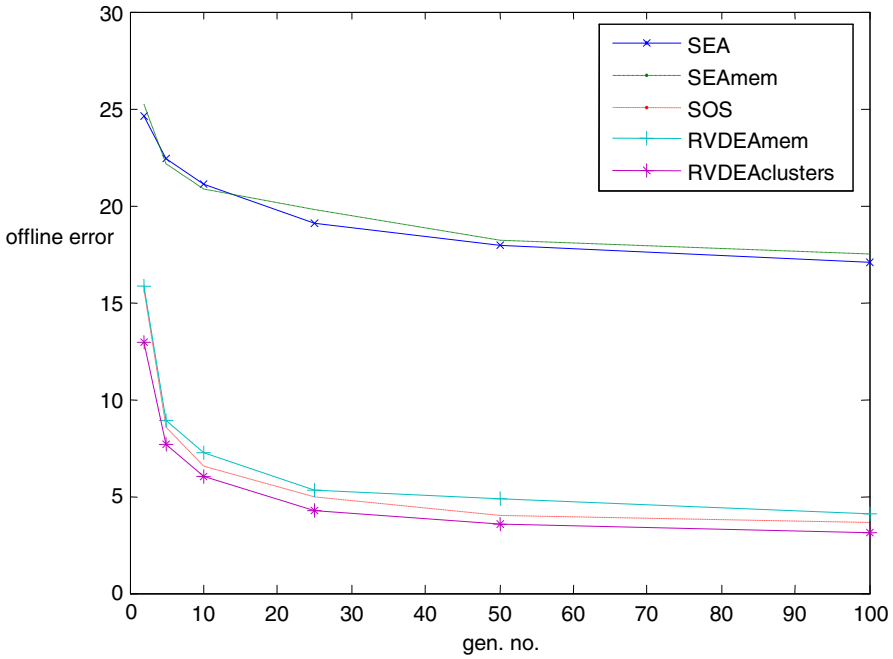


Figure 4.2. Offline error vs. change frequency for some DEAs on a moving cone peaks benchmark problem [DF1]

TABLE 4.5

OFFLINE ERROR VARIATION AFTER 500,000 EVALUATIONS AS FUNCTION OF PEAK NUMBER ON TIME VARYING GAUSSIAN PEAKS BENCHMARK PROBLEM [DF2]

Peak no.	SEA mem	RVDEA mem	RVDEA clusters
1	16.15	1.79	0.302
5	24.49	4.20	2.653
10	28.67	6.36	3.871
50	29.19	7.54	3.322
100	29.75	8.06	3.713
200	31.32	11.59	3.755

TABLE 4.6

OFFLINE ERROR VARIATION AFTER 500,000 EVALUATIONS AS FUNCTION OF GENERATION BETWEEN CHANGES ON TIME VARYING GAUSSIAN PEAKS BENCHMARK PROBLEM WITH 10 PEAKS [DF2]

Gen. no.	RVDEA mem	RVDEA clusters
2	14.41	10.11
5	8.83	7.55
10	6.98	4.41
25	6.44	4.12
50	6.36	3.87
100	5.95	3.34

TABLE 4.7

OFFLINE ERROR VARIATION AFTER 500,000 EVALUATIONS AS FUNCTION OF CYCLE LENGTH ON MOVING PARABOLA BENCHMARK PROBLEMS WITH 10 PEAKS [DF3-DF5]

Moving parabola type	Cycle length (eval)	SEA mem	RVDEA mem	RVDEA clusters
Linear [DF3]	1,000	10.893	2.334	0.881
	2,500	10.821	2.125	0.755
	5,000	10.865	2.082	0.609
	10,000	10.944	1.781	0.483
	20,000	11.291	1.622	0.299
	100,000	11.033	1.413	0.177
Random [DF4]	1,000	11.126	2.752	1.026
	2,500	11.065	2.611	0.982
	5,000	10.877	2.303	0.891
	10,000	10.531	2.142	0.769
	20,000	10.218	1.897	0.536
	100,000	9.893	1.662	0.247
Circular [DF5]	1,000	12.844	2.989	1.583
	2,500	12.815	2.788	1.457
	5,000	12.663	2.445	1.162
	10,000	12.587	2.121	0.783
	20,000	12.499	1.965	0.607
	100,000	12.431	1.792	0.340

TABLE 4.8

OFFLINE ERROR VARIATION AFTER 500,000 EVALUATIONS AS FUNCTION OF PEAKS NUMBER ON MOVING PARABOLA BENCHMARK PROBLEMS WITH CYCLE LENGTH 5,000 EVALUATIONS [DF3-DF5]

Moving parabola type	Peaks no.	RVDEA mem	RVDEA clusters
Linear [DF3]	1	1.517	0.081
	5	1.892	1.122
	10	2.082	1.609
	50	2.367	1.756
	100	2.688	2.186
	200	2.850	2.377
Random [DF4]	1	1.268	0.106
	5	1.764	1.446
	10	2.303	1.791
	50	2.675	1.862
	100	2.904	1.985
	200	3.023	2.149
Circular [DF5]	1	1.687	0.158
	5	2.022	0.967
	10	2.445	1.162
	50	2.664	1.368
	100	2.864	1.743
	200	3.191	2.040

TABLE 4.9

OFFLINE ERROR VARIATION AFTER 500,000 EVALUATIONS AS FUNCTION OF CYCLE LENGTH ON OSCILLATING PEAKS BENCHMARK PROBLEM [DF6]

Cycle length (eval)	SEA mem	RI25 mem	P3 mem	P3RI25 mem [8]	Mem/ search	RVDEA mem	RVDEA clusters
1,000	11.93	9.01	10.07	9.87	7.19	4.252	2.648
5,000	n/a	n/a	n/a	n/a	n/a	4.132	2.411
10,000	n/a	n/a	n/a	n/a	n/a	3.978	2.342
100,000	17.66	7.26	12.00	9.27	4.71	3.821	2.025

Results for SEAmem, RI25mem, P3mem, P3RI25mem and Mem/search as reported in [8]

TABLE 4.10

ADAPTATION PERFORMANCE FOR RVDEAMEM AFTER 500,000 EVALUATIONS AS FUNCTION OF PEAK NUMBER

Peak no.	DF1	DF2	DF3	DF4	DF5	DF6
1	0.9822	0.9741	0.9780	0.9816	0.9756	0.9382
5	0.9329	0.9391	0.9726	0.9744	0.9707	0.9394
10	0.9293	0.9078	0.9699	0.9666	0.9646	0.9411
25	0.9309	0.8907	0.9679	0.9644	0.9634	0.9415
50	0.9245	0.8832	0.9657	0.9612	0.9614	0.9431
100	0.9278	0.8320	0.9610	0.9579	0.9585	0.9438

TABLE 4.11

ADAPTATION PERFORMANCE FOR RVDEAMEM AFTER 500,000 EVALUATIONS AS FUNCTION OF GENERATION BETWEEN CHANGES

Gen. no.	DF1	DF2	DF3	DF4	DF5	DF6
2	0.7707	0.7912	0.9538	0.9486	0.9455	0.9293
5	0.8712	0.8720	0.9573	0.9535	0.9521	0.9343
10	0.8955	0.8988	0.9662	0.9601	0.9567	0.9384
25	0.9225	0.9067	0.9692	0.9622	0.9596	0.9391
50	0.9293	0.9078	0.9698	0.9666	0.9646	0.9401
100	0.9403	0.9138	0.9742	0.9690	0.9693	0.9423

TABLE 4.12

ADAPTATION PERFORMANCE FOR RVDEACLUSTER AFTER 500,000 EVALUATIONS AS FUNCTION OF PEAK NUMBER

Peak no.	DF1	DF2	DF3	DF4	DF5	DF6
1	0.9852	0.9956	0.9988	0.9985	0.9977	0.9570
5	0.9612	0.9616	0.9837	0.9790	0.9860	0.9846
10	0.9487	0.9439	0.9767	0.9740	0.9832	0.9625
25	0.9436	0.9480	0.9756	0.9734	0.9818	0.9653
50	0.9452	0.9519	0.9746	0.9730	0.9802	0.9659
100	0.9516	0.9462	0.9683	0.9712	0.9747	9.9677

TABLE 4.13

ADAPTATION PERFORMANCE FOR RVDEACLUSTER AFTER 500,000 EVALUATIONS AS FUNCTION OF GENERATION BETWEEN CHANGES

Gen. no.	DF1	DF2	DF3	DF4	DF5	DF6
2	0.8129	0.8535	0.9819	0.9796	0.9709	0.9558
5	0.8888	0.8906	0.9851	0.9824	0.9737	0.9573
10	0.9123	0.9361	0.9872	0.9851	0.9771	0.9616
25	0.9380	0.9403	0.9891	0.9858	0.9789	0.9630
50	0.9487	0.9439	0.9912	0.9871	0.9832	0.9651
100	0.9545	0.9516	0.9930	0.9889	0.9887	0.9661

The effect of increasing number of peaks on the output of the algorithm is more or less equivalent to an increase in problem difficulty and decrease in performance. But as can be seen from the results given in some of the tables, at some values of peak number there are discrepancies and this is mainly due to the fact that the operation of the algorithm is stochastic and the output may not always follow a given pattern. A similar argument also applies to increase in the number of generations between changes. In such cases, the output of the algorithm generally decreases with increase in change frequency with the exception of some discrepancies. However, the output of the algorithm is still better than most chosen algorithms.

TABLE 4.14

OFFLINE ERROR PERFORMANCE FOR RVDEA WITOUT MEMORY AFTER 500,000 EVALUATIONS ON TEST PROBLEMS DF1-DF6 WITH 10 PEAKS

	DF1	DF2	DF3	DF4	DF5	DF6
RVDEA	4.90	6.42	2.11	2.35	2.51	4.35
RVDEAmem	4.88	6.36	2.082	2.303	2.445	4.132
RVDEAcluster	3.54	3.877	1.609	1.791	1.162	2.411

Lastly, the proposed algorithm, RVDEA, is analyzed without memory to see how much of the observed performance can be attributed to the proposed algorithm in contrast to the use of memory or clusters. The results clearly suggest that RVDEA contributes to most part of the observed performances. The results are presented in Table 4.14.

In general, the results show that RVDEA is capable of solving dynamic optimization problems. The proposed algorithm is very easy to implement and the additional computational cost is very cheap. The proposed algorithm uses the evolutionary progress of each individual to arrive at the optimal relocation needed specifically for that individual to adapt to the new environment. Hence, it is a fast and effective adaptation scheme that occurs at the transient stages of a change. Furthermore, since the algorithm is fast and simple, additional enhancements in population diversity can be included at each steady state operation without bypassing the acceptable computational cost limit. In this paper, we implemented multiple clusters as a means to maintain steady state diversity. By using this simple clustering scheme, the algorithm performed better than all other dynamic evolutionary algorithms tested in this study. This performance improvement shows that RVDEA has a great potential to be used alongside other dynamic evolutionary techniques. Furthermore, the ease of implementing the algorithm makes the proposed algorithm very attractive. It should be noted that the algorithm still performs much better than most of the other algorithms cited in this chapter without any performance enhancement techniques.

4.2 Constrained Multi-Objective Evolutionary Algorithm

4.2.1 *Experimental setup* - The second set of experiments were conducted on the proposed constrained multi-objective evolutionary algorithm. The algorithm is tested on several constrained multi-objective benchmark problems available from literature. The results are summarized below. We use a population size of 100, crossover rate of 0.8, mutation rate of 0.2, and maximum generation number of 100 for all implementations. These values are chosen in consistent with the other algorithms to be compared with. In addition, we use SBX crossover and mutation. Tournament selection is adopted in recombination and replacement scheme. The test problems are denoted as BNH [56], SRN [40, 62], OSY [61], TNK [63], CTP1 [64], CTP2 [64], CTP3 [64], CTP4 [64], CTP5 [64], CTP6 [64], CTP7 [64], CTP8 [64], CONSTR [64], and Welded Beam Problem [60]. Each test is run 50 times and the performance metrics are measured statistically.

4.2.2 *Results and discussions* - The algorithm was able to find very accurate Pareto optimal solutions for all the benchmark test problems. The convergence metric with respect to the true Pareto front is given in TABLE 4.15. As can be noticed from the table, the proposed algorithm has an average accuracy in the order of 0.0165. Furthermore, we have presented the diversity metric for the Pareto front obtained in TABLE 4.16. As can be observed from the table, the proposed algorithm consistently provides diversified non-dominant individuals in the resulted Pareto front. Moreover, the lower and upper limits of the objective functions of the Pareto front attained are covered to the possible extent. TABLE 4.17 provides the upper and lower extreme values of the objective functions in the non-dominated set. These values represent the extent to which the Pareto front extends and this indicates how far in the Pareto-front the proposed algorithm is able to find optimal feasible solutions in each test problem.

The test problem, OSY [61], has two second-order nonlinear objective functions, four linear inequality constraints, and two nonlinear second-order inequality constraints. The resulting Pareto front obtained by the proposed algorithm is shown in Figure 4.3a. The Pareto front is known to be piece-wise continuous and as can be seen from the plot, the proposed algorithm is able to provide better-fit feasible individuals well distributed over the Pareto front (with convergence metric (0.0056) and diversity metric (0.6879) over 50 runs). The Pareto-front extends well between -273 and -49 in the first objective function, and between 4.05 and 75.2 in the second objective function. The convergence and diversity metric plots over 100 generations for OSY test problem are given in Figure 4.6a.

The next test problem, BNH [56], has two second-order nonlinear objective functions and two nonlinear second-order inequality constraints. The resulting Pareto front generated by the proposed algorithm is given in Figure 4.3b. As can be noted from the figure and also from the convergence (0.0038) and diversity (0.6162) metrics, the proposed algorithm provides feasible optimal solutions that are diversely distributed on the true Pareto front. Furthermore, the resulting allowable feasible limits in the Pareto front extends to as low as 1.6e-3 and as high as 136 in the first objective function, and as low as 4.0 and as high as 49.8 in the second objective function. The convergence and

diversity metric plots over 100 generations for BNH test problem are given in Figure 4.6b.

The following test is conducted using CTP1 test problem [64]. This problem has one linear objective function, one exponentially nonlinear objective function, and two exponentially nonlinear inequality constraints. The generated Pareto front is given in Figure 4.3c. As can be noted from the plot and the convergence metric (0.0014) and the diversity metric (0.5827) obtained over 50 runs, the proposed algorithm provides better-fit feasible individuals well distributed over the Pareto front. The first objective function extends between 7.5e-13 and 0.995, while the second objective function extends between 0.543 and 1.05. The convergence and diversity metric plots over 100 generations for CTP1 test problem are given in Figure 4.6c.

The next consecutive experiments are conducted on six test problems starting from CTP2 up to CTP7 [64]. These problems have one linear objective function, one nonlinear objective function, and a single highly nonlinear inequality constraint. The problems have different parameter values which results in varying degrees of difficulties and varying feasible optimal solutions. The resulting Pareto fronts are shown in Figures 4.3d-4.3i. As can be seen from the plots and the resulted convergence metrics and the diversity metrics from Tables 4.14 and 4.15, the proposed algorithm provides better-fit feasible individuals over the discontinuous Pareto front. For the test problem CTP2, the algorithm is able to provide a convergence metric value of 9.3e-4 and a diversity metric value of 0.6389. The Pareto front obtained extends between 6.2e-4 and 0.981 for the first objective function and between 0.288 and 1.0 for the second objective function. For the case of CTP3 test problem, a convergence metric value of 0.0074 and a diversity metric value of 0.9032 are obtained. For this problem, the Pareto front extends between 0 and 0.976 for the first objective function and between 0.301 and 1.11 for the second objective function. Similarly for CTP4, a convergence metric value of 0.0259 and a diversity metric value of 0.8993 are obtained and the resulting Pareto front extends between 0 and 0.824 for the first objective function and between 0.433 and 1.17 for the second objective function. In test problem CTP5, the convergence metric is found to be 0.0017 and the diversity metric is found to be 0.9182. In this problem, the resulted Pareto front extends between 5.3e-17 and 0.974 in the first objective function and between 0.329 and 1.0 in

the second objective function. Similarly for CTP6, a convergence metric value of 0.0039 and a diversity metric value of 0.5632 are obtained and the resulting Pareto front is found to extend between 1.7e-16 and 1.0 in the first objective function and 0.884 and 3.81 in the second objective function. Finally for test problem CTP7, a convergence metric value of 9.6e-4 and a diversity metric value of 0.6602 are obtained. The Pareto front for this problem extends between 1.5e-16 and 1.0 for the first objective function and between 3.1e-9 and 1.09 for the second objective function. The convergence and diversity metric plots over 100 generations for test problems CTP2-CTP7 are given in Figures 4.6d – 4.6i.

The next test is conducted on CTP8 test problem [64] which has one linear objective function, one nonlinear objective function, and two highly nonlinear inequality constraints. The resulted Pareto front is shown in Figure 4.3j. As can be observed from the plot and the convergence metric (7.6e-4) and the diversity metric (0.6368), the proposed algorithm provides better-fit feasible individuals over the discontinuous Pareto front. The resulting Pareto front extends between 6.1e-17 and 0.822 in the first objective function and between 1.38 and 3.71 in the second objective function. The convergence and diversity metric plots over 100 generations for CTP8 test problem are given in Figure 4.6j.

The next test is conducted on CONSTR [64] which has one linear objective function, one nonlinear objective function, and two linear inequality constraints. The resulted Pareto front by the proposed algorithm is shown in Figure 4.4c. This problem is relatively easy compared to the other test problems. As can be noted from the plot and the convergence (0.0049) and diversity (0.6841) metrics, the proposed algorithm produces a very good Pareto front. This Pareto front extends between 0.398 and 0.98 for the first objective function and between 1.00 and 8.79 for the second objective function. The convergence and diversity metric plots over 100 generations for CONSTR test problem are given in Figure 4.6k.

The next experiment applies to SRN test problem [40, 62]. This problem has two second-order nonlinear objective functions, one linear inequality constraint, and one nonlinear second-order inequality constraint. The resulted Pareto front by the proposed algorithm is displayed in Figure 4.4f. As can be seen from the plot and the convergence metric (0.0016) and the diversity metric (0.5792), the proposed algorithm provides very

good and diversified feasible individuals over the Pareto front. This Pareto front extends between 10.3 and 219 for the first objective function and between -216 and 3.97 for the second objective function. The convergence and diversity metric plots over 100 generations for SRN test problem are given in Figure 4.6l.

The next experiment is conducted on TNK test problem [63]. This problem has two linear objective functions, one second-order nonlinear inequality constraint, and one highly nonlinear trigonometric inequality constraint. The converged Pareto front by the proposed algorithm is shown in Figure 4.4i. The Pareto front is known to be discontinuous and as can be seen from the plot, the proposed algorithm still provides better-fit feasible individuals that are well distributed over the Pareto front (with convergence metric 0.0053 and diversity metric 0.7461); first objective function extends between 2.9e-6 and 1.05; second objective function extends between 1.3e-7 and 1.05). The convergence and diversity metric plots over 100 generations for TNK test problem are given in Figure 4.6m.

TABLE 4.15
CONVERGENCE METRIC
MEAN (FIRST ROW) AND VARIANCE (SECOND ROW) OF THE CONVERGENCE METRIC γ AFTER 100 GENERATIONS

	OSY	BNH	CTP1	CTP2	CTP3	CTP4	CTP5
Avg	0.0056	0.0038	0.0014	9.3e-4	0.0074	0.0259	0.0017
Var	9.7e-6	2.5e-5	3.5e-5	4.3e-4	3.4e-4	4.1e-4	3.6e-4
	CTP6	CTP7	CTP8	CONSTR	SRN	TNK	
Avg	0.0039	9.6e-4	7.6e-4	0.0049	0.0016	0.0053	
Var	4.2e-4	4.6e-4	4.9e-4	3.3e-5	3.1e-5	2.9e-4	

The plots of the obtained Pareto fronts for the constrained multi-objective test problems meet our expectation. Not only are we able to find feasible individuals, but also we are able to find better-fit individuals that are on or very close to the true Pareto front. In addition, the infeasible individuals are fully exploited during the evolutionary process to allow the algorithm to have an evenly distributed and well extended Pareto front. For comparison with other algorithms, we have reproduced the results reported in [57] for

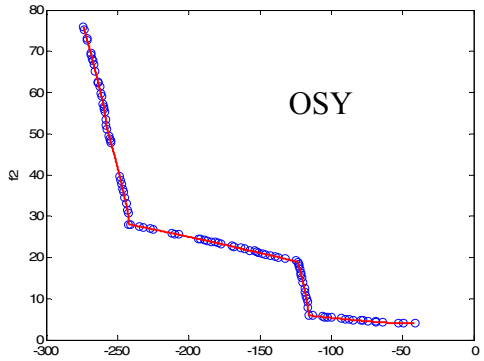
CONSTR, SRN, and TNK test problems in Figure 4.4. Clearly, the proposed algorithm performs better than Ray-Tai-Seow's algorithm in all of the three test problems. The uniformity and the extent of the Pareto fronts obtained from the proposed algorithm are also better than those of NSGA-II even though subtle. The reason is that the proposed algorithm not only searches in the feasible space but also exploits the evolutionary information contained in individuals with low objective value and low constraint violation.

TABLE 4.16
DIVERSITY METRIC
MEAN (FIRST ROW) AND VARIANCE (SECOND ROW) OF THE DIVERSITY METRIC Δ AFTER 100 GENERATIONS

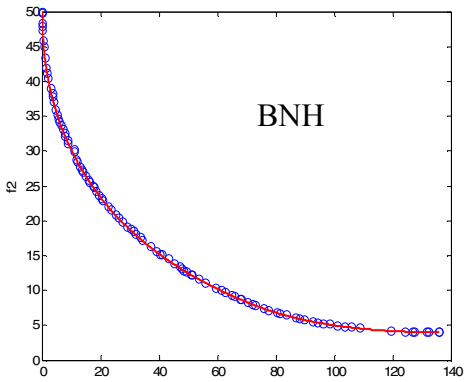
	OSY	BNH	CTP1	CTP2	CTP3	CTP4	CTP5
Avg	0.6879	0.6162	0.5827	0.6389	0.9032	0.8993	0.9182
Var	0.0422	0.0502	0.0459	0.0479	0.0497	0.0508	0.0517
	CTP6	CTP7	CTP8	CONSTR	SRN	TNK	
Avg	0.5632	0.6602	0.6368	0.6841	0.5792	0.7461	
Var	0.0513	0.0534	0.0595	0.0224	0.0381	0.0529	

TABLE 4.17
LOWER AND UPPER BOUNDS OF THE OBJECTIVE FUNCTIONS IN THE NON-DOMINANT SOLUTIONS AFTER 100 GENERATIONS

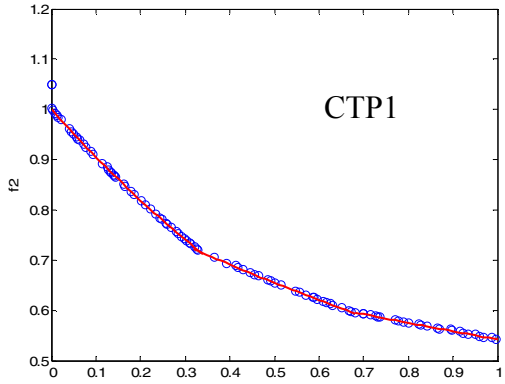
		OSY	BNH	CTP1	CTP2	CTP3	CTP4	CTP5
f_1	Lower	-273	1.6e-3	7.5e-13	6.2e-4	0.0	0.0	5.3e-17
	Upper	-49	136	0.995	0.981	0.976	0.824	0.974
f_2	Lower	4.05	4.0	0.543	0.288	0.301	0.433	0.329
	Upper	75.2	49.8	1.05	1.0	1.11	1.17	1.0
		CTP6	CTP7	CTP8	CONSTR	SRN	TNK	
f_1	Lower	1.7e-16	1.5e-16	6.1e-17	0.398	10.3	2.9e-6	
	Upper	1.0	1.0	0.822	0.98	219	1.05	
f_2	Lower	0.884	3.1e-9	1.38	1.0	-216	1.3e-7	
	Upper	3.81	1.09	3.71	8.79	3.97	1.05	



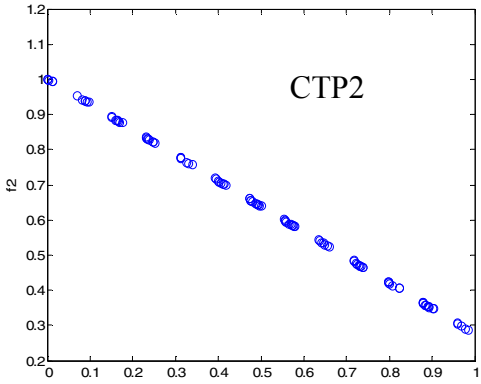
(a)



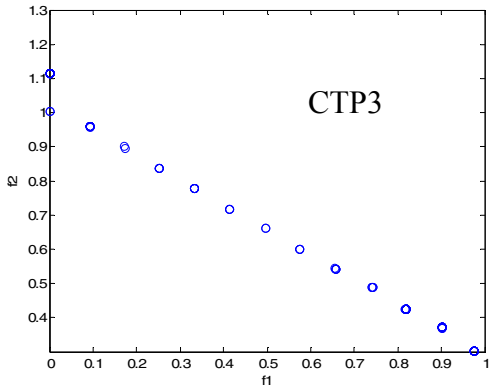
(b)



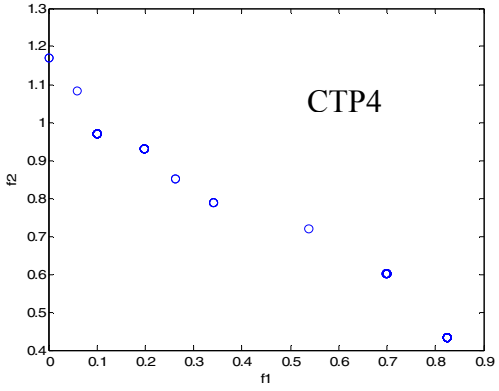
(c)



(d)



(e)



(f)

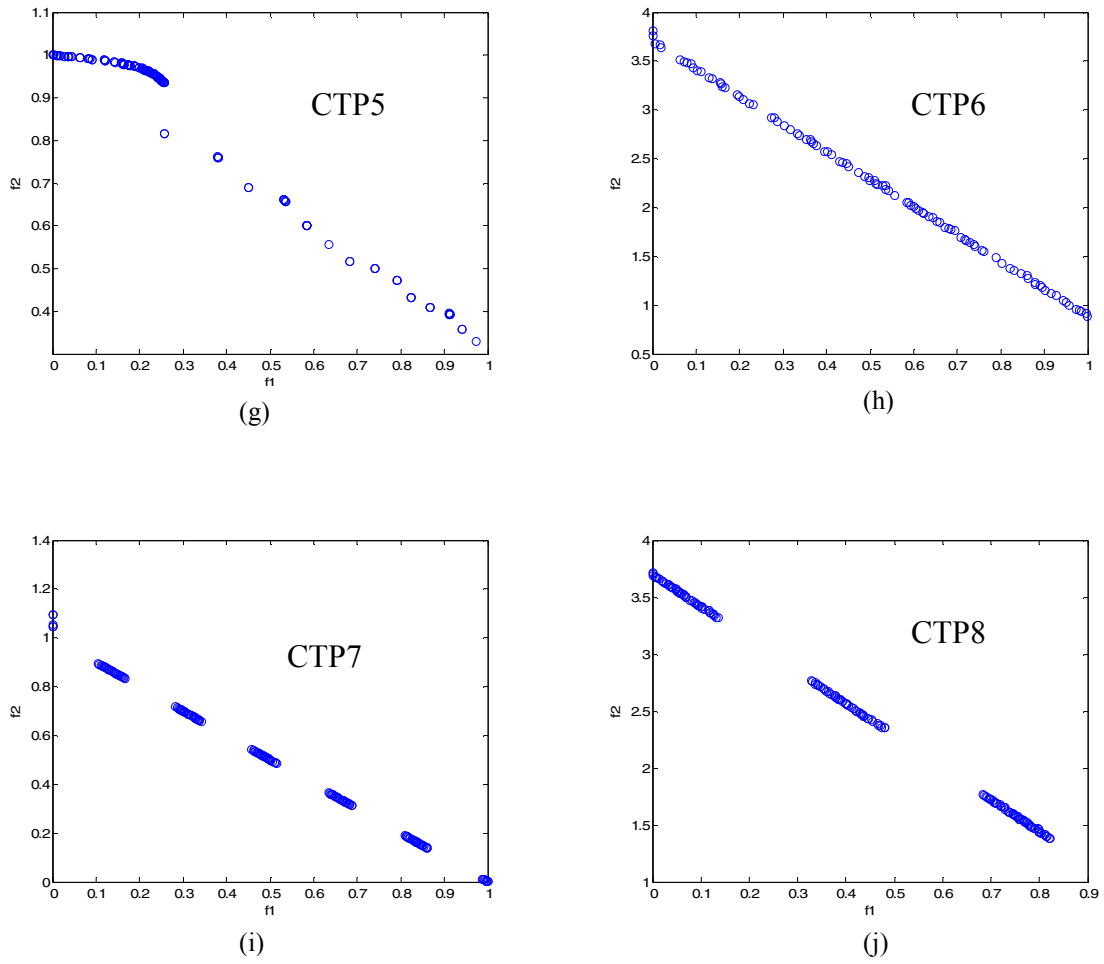


Figure 4.3. The Pareto fronts for different constraint multi-objective test problems obtained by the proposed algorithm

A more notable performance difference is obtained for welded beam problem presented in [60]. The proposed algorithm is able to provide a much better diversity in the Pareto front within the same (i.e., 8,000) objective function evaluations as shown in Figure 4.5. The convergence metric for this problem is found to be 0.0062 while the diversity metric is found to be 0.7581. The Pareto front extends between 2.2 and 35.7 in the first objective function and between 4.39e-4 and 8.56e-3 in the second objective function. The convergence and diversity metric plots over 100 generations for the welded beam test problem are given in Figure 4.6n.

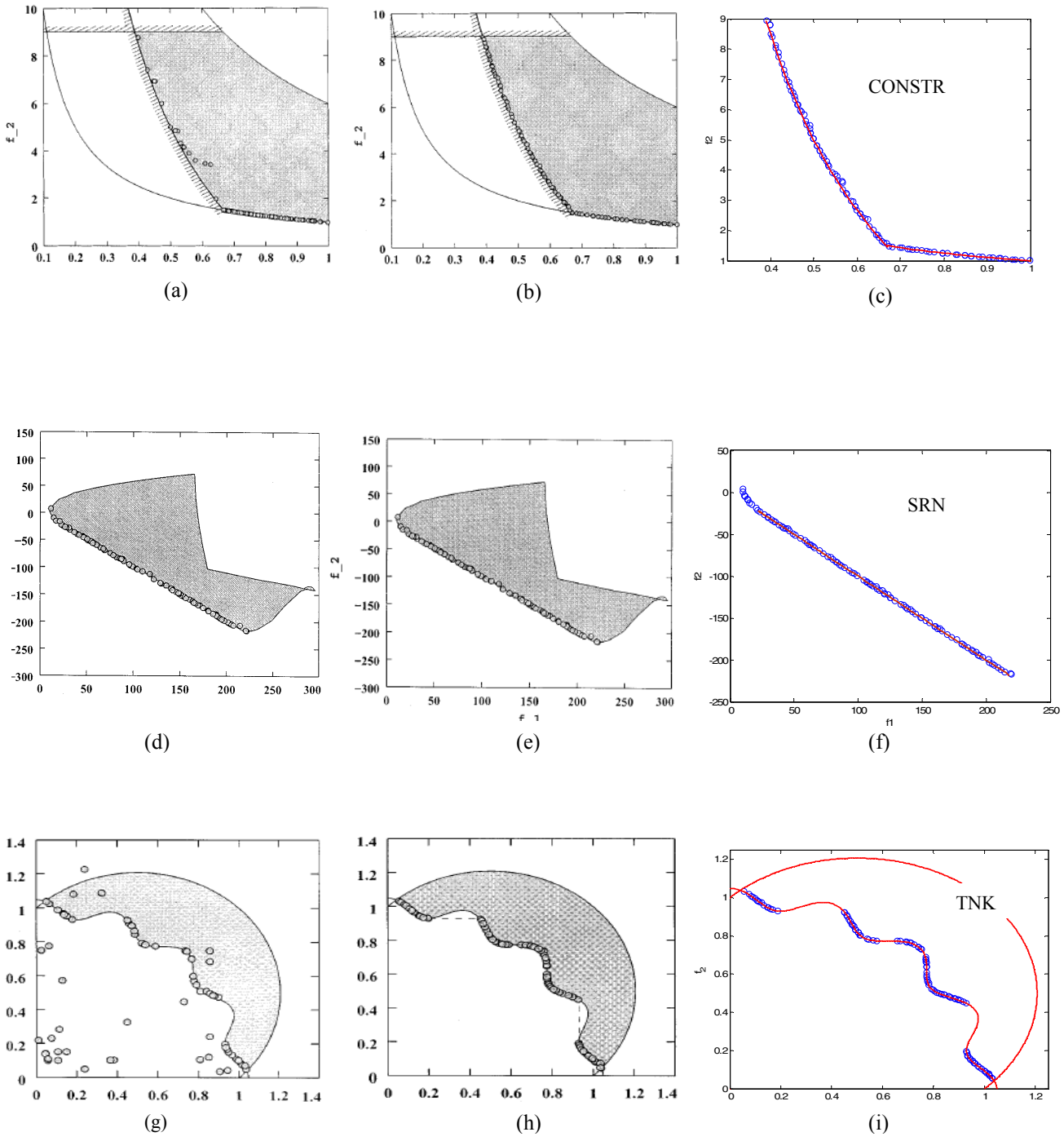
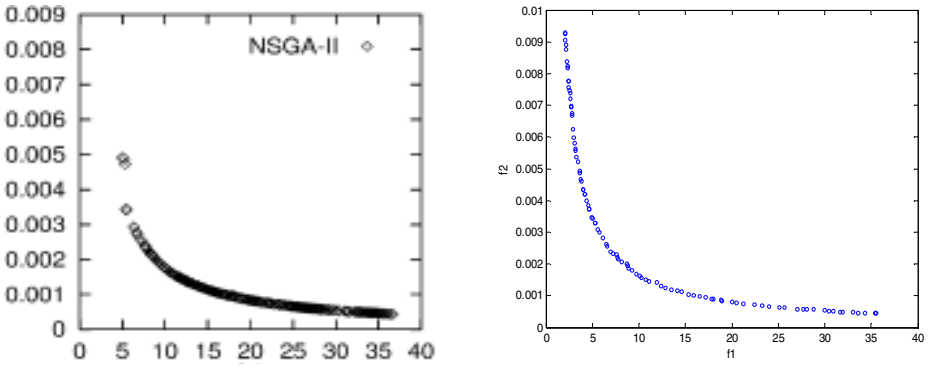
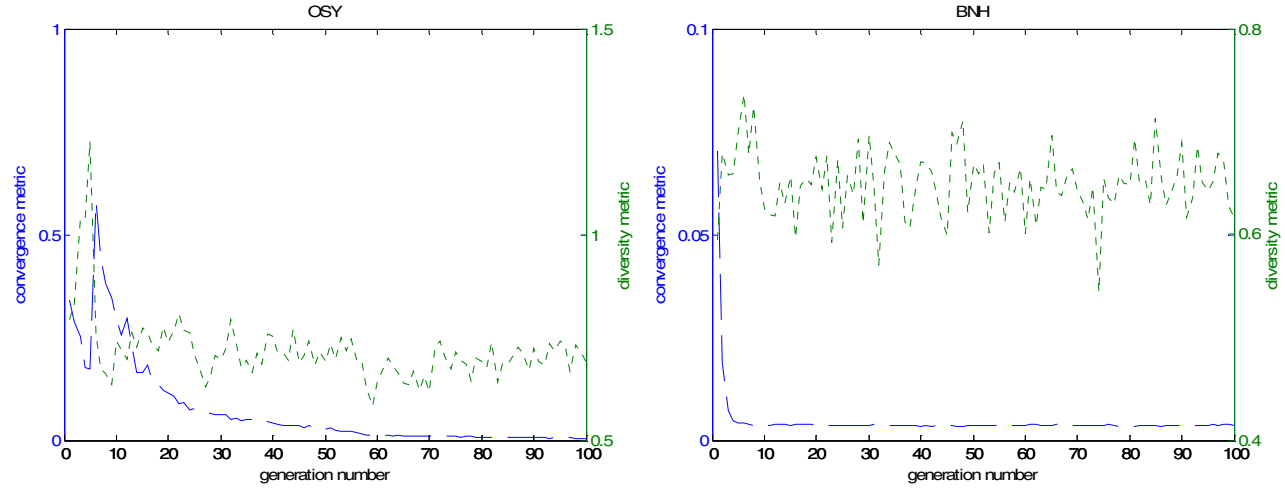


Figure 4.4. The Pareto fronts for three constraint multi-objective test problems, CONSTR, SRN and TNK (from top to bottom) obtained by Ray-Tai-Seow algorithm (left), NSGA-II (middle) [57] and the proposed algorithm (right).

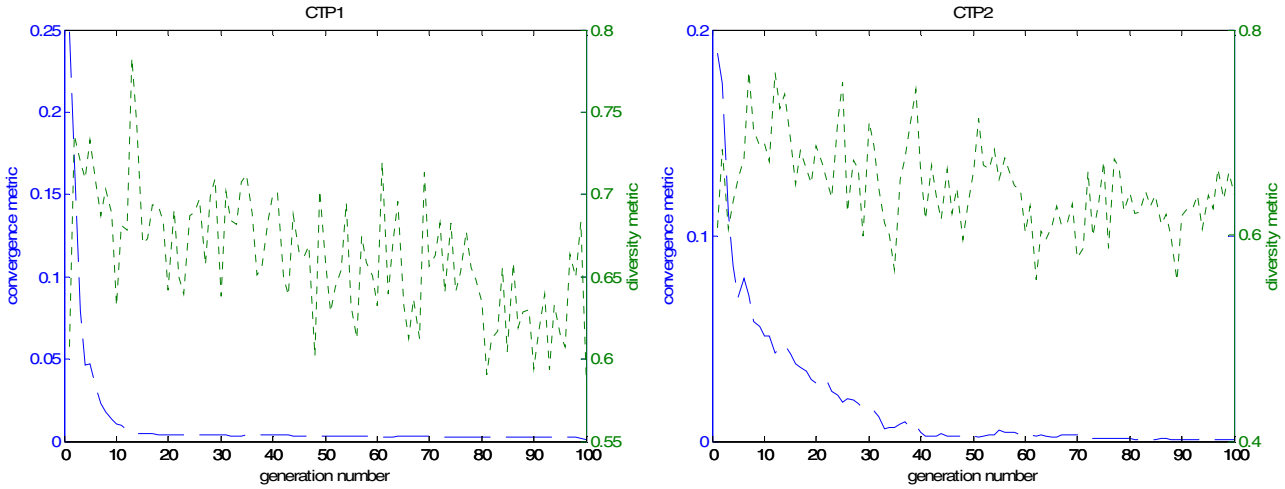


(a) (b)

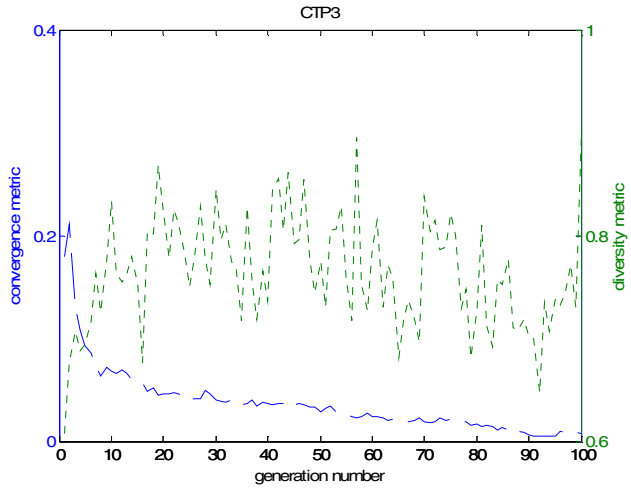
Figure 4.5. The obtained Pareto front of welded beam problem for (a) NSGA-II (as reported in [60]) and (b) the proposed algorithm.



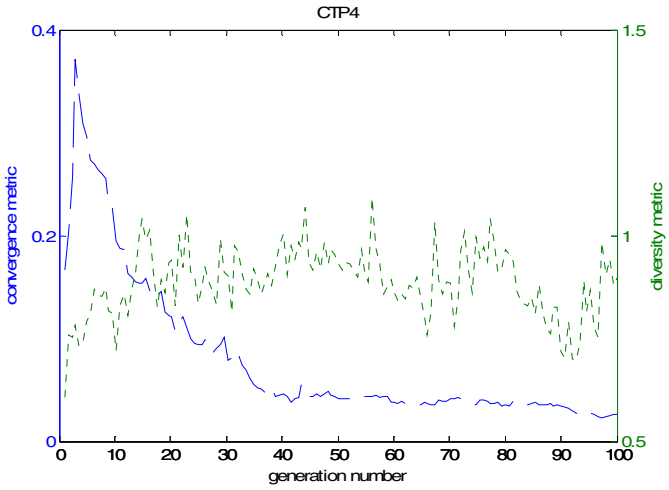
(a) (b)



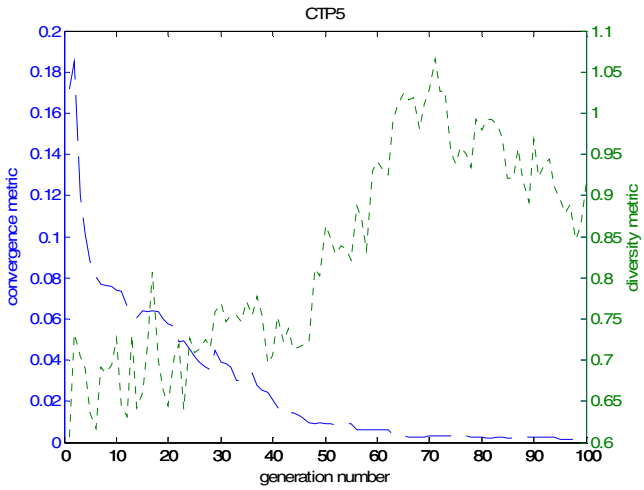
(c) (d)


CTP3
convergence metric
diversity metric
generation number
0.4
0.2
0
1
0.8
0.6
0 10 20 30 40 50 60 70 80 90 100

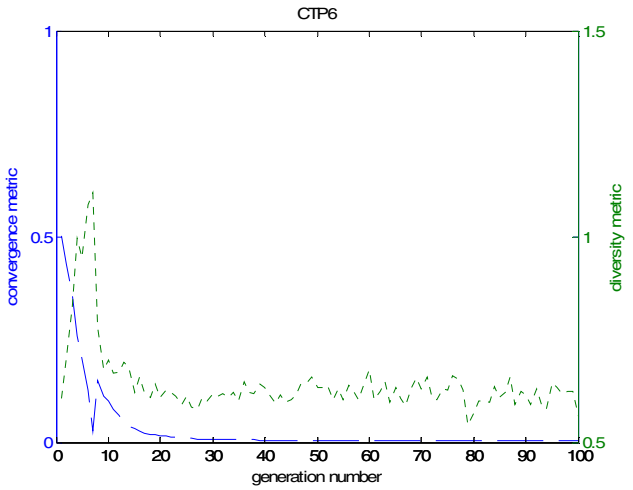
(e)


CTP4
convergence metric
diversity metric
generation number
0.4
0.2
0
1.5
1
0.5
0 10 20 30 40 50 60 70 80 90 100

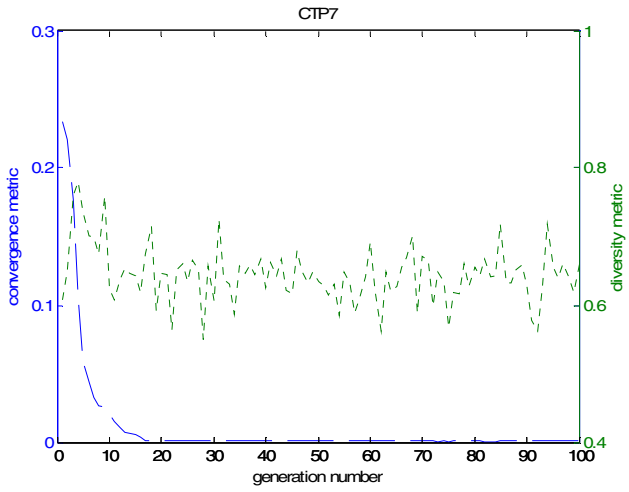
(f)


CTP5
convergence metric
diversity metric
generation number
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
1.1
1.05
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0 10 20 30 40 50 60 70 80 90 100

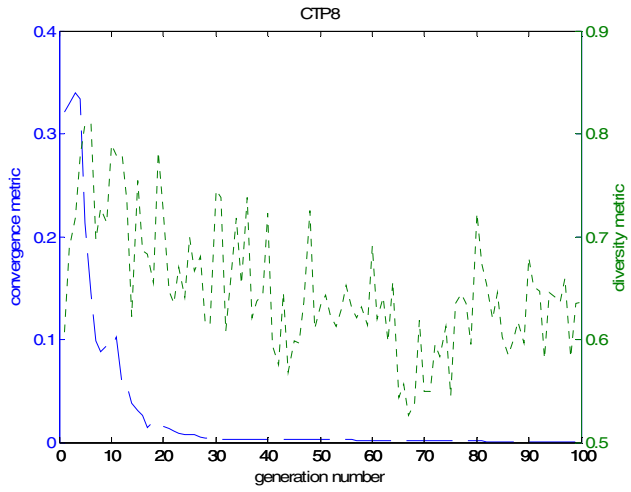
(g)


CTP6
convergence metric
diversity metric
generation number
1
0.5
0
1.5
1
0.5
0 10 20 30 40 50 60 70 80 90 100

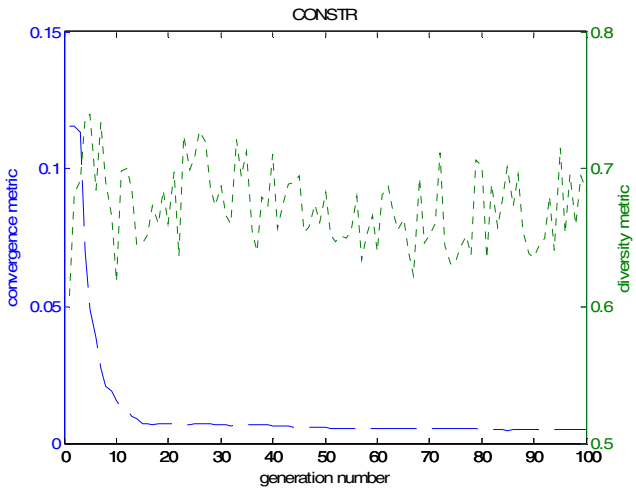
(h)


CTP7
convergence metric
diversity metric
generation number
0.3
0.2
0.1
0
1
0.8
0.6
0.4
0 10 20 30 40 50 60 70 80 90 100

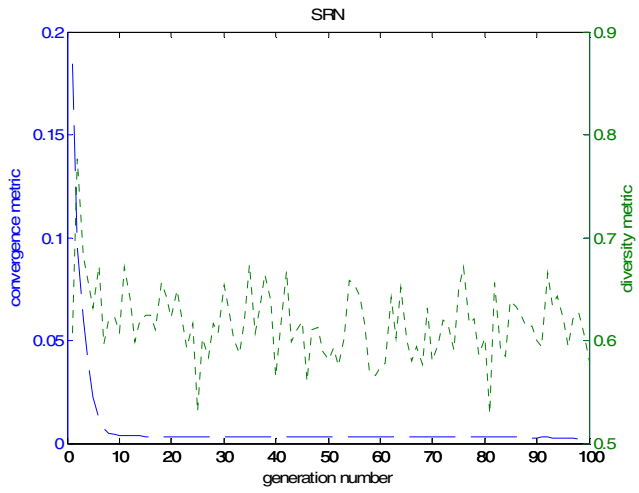
(i)


CTP8
convergence metric
diversity metric
generation number
0.4
0.3
0.2
0.1
0
0.9
0.8
0.7
0.6
0.5
0 10 20 30 40 50 60 70 80 90 100

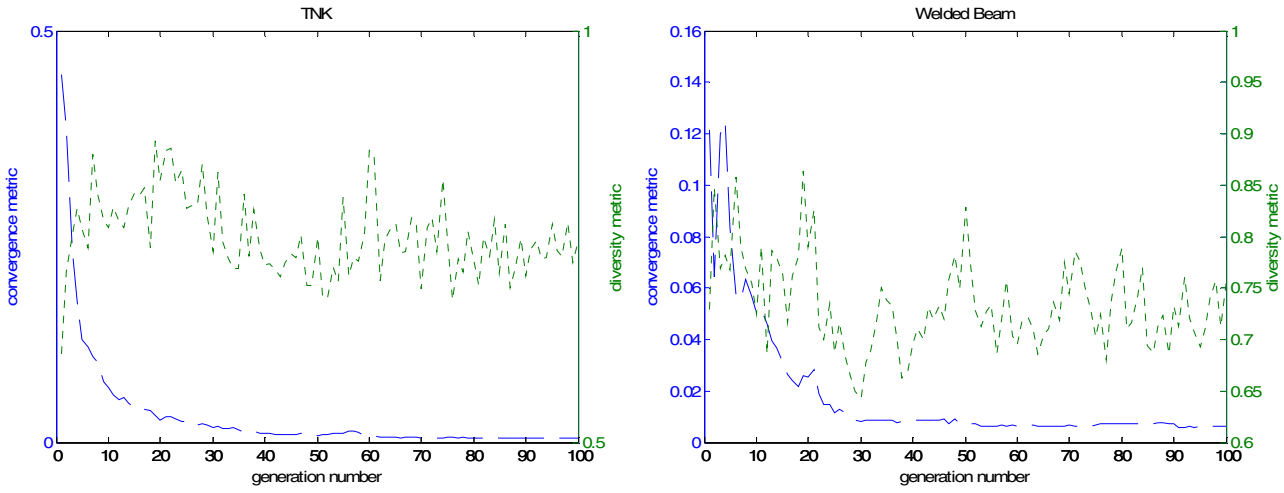
(j)


CONSTR
convergence metric
diversity metric
generation number
0.15
0.1
0.05
0
0.8
0.7
0.6
0.5
0 10 20 30 40 50 60 70 80 90 100

(k)


SRN
convergence metric
diversity metric
generation number
0.2
0.15
0.1
0.05
0
0.9
0.8
0.7
0.6
0.5
0 10 20 30 40 50 60 70 80 90 100

(l)



(m) (n)

Figure 4.6. The diversity and convergence metric plots for all test problems over 100 generations

The success of the proposed algorithm is mainly due to the exploitation of the evolutionary information contained in infeasible individuals in addition to that contained in feasible individuals. The constraint handling normally used in NSGA-II [57] compares infeasible individuals solely based on their constraint violation. This way of non-dominance ranking ignores how well each individual was performing in the objective space and may result in the inefficient use of some evolutionary materials. The proposed algorithm, on the other hand, uses a combined measure of constraint violation and objective performance to arrive at the fitness of individuals which will govern the evolutionary process. The number of feasible individuals available in the current population is used to control the relative emphasis given to either constraint violation or objective performance in the final fitness calculation. As can be observed from the test results of the proposed algorithm, this way of fitness formulation provides better solutions that extend well over the Pareto front. Furthermore, the convergence and diversity plots indicate that a continuous convergence and diversified population is achieved in the proposed algorithm

CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS

Many real-world optimization problems have to be performed under the presence of various uncertainties and constraints. In this thesis, two common optimization problems are studied using evolutionary algorithms. These are dynamic optimization problems and constrained multi-objective optimization problems.

Dynamic optimization problems are common optimization problems with a fitness landscape that undergoes various changes during the optimization. In this thesis, we proposed a dynamic evolutionary algorithm that uses variable relocation vectors to adapt already converged or currently evolving individuals to the changing landscape. The proposed algorithm relocates the individuals based on their change in function value due to the change in the environment and the average sensitivities of their decision variables to the corresponding change in the objective space. The relocation vectors introduce a certain radius of uncertainty to be applied specifically to each individual and in effect restoring diversity and accelerating exploration. Since the adaptation is conducted on the previous population, the proposed algorithm provides higher reusability of previous evolutionary information. Furthermore, the algorithm provides faster convergence and better adaptation and this makes it attractive for optimizing fitness landscapes with higher frequency of change. In addition, the algorithm is able to find optimal solutions in higher severities of change. Severe changes require higher diversity restoration and the proposed algorithm uses larger relocation vectors to do so. The relocated population is shown to be better fit to the new environment than the original or any other randomly generated population. The algorithm has been tested for several dynamic benchmark problems and has shown better results compared to some chosen state-of-the-art dynamic evolutionary approaches.

The relocation vectors are specific to each individual and this gives the algorithm better adaptation than those approaches that use a single adaptation value for the whole population. Furthermore, using specific sensitivities and relocation vectors allow the algorithm to provide a considerable progress jump for the next evolutionary process. As a technique to be used during transient periods, the proposed algorithm provides the next evolutionary cycle with better initial population than any other randomly generated population. Furthermore, the extra computational cost of the proposed algorithm is comparable to its performance improvement since the additional calculations are basic arithmetic operations. Hence, the performance improvement per extra computational cost is higher in the proposed algorithm than the other dynamic evolutionary algorithms used in this thesis. The algorithm can be easily integrated into standard evolutionary algorithms and other uncertainty handling techniques like multi-population and diversity preservation. This fast adaptation scheme when enhanced with diversity perseveration techniques provides much better overall performance. The authors believe that better results can also be obtained by implementing the relocation scheme on multi-population approaches. For future work, the authors recommend applying the variable relocation scheme to multi-population approaches for solving dynamic optimization problems.

This thesis also studies constrained multi-objective optimization problems which are common multi-objective optimization problems with constraints limiting their feasible space. In this thesis, we proposed an adaptive constraint handling technique for solving constrained multi-objective optimization problems. Beside the search for optimal solutions in the feasible region, the algorithm also exploits the information hidden in infeasible individuals with better objectives and lower constraint violation. This is achieved by using the modified objective values in the non-dominance ranking of the multi-objective evolutionary algorithm. The modified objective values are the modifications of the objective functions to incorporate the effects of the individuals' constraint violation. They are composed of distance measures and penalty functions. These values are associated with how well an individual performs and how much it violates the constraints. They are obtained for every objective function dimension. For feasible individuals, the distance values are just the normalized objective function values. For infeasible individuals, the distance values are obtained from the normalized objective

function values and their constraint violation. The penalty function, on the other hand, will be applied to infeasible individuals in order to further decrease their fitness compared to feasible individuals. The number of feasible individuals in the population adaptively controls the emphasis given to objective values or constraint violation in the modified objective function formulation. If there is no feasible individual in the population, the algorithm uses the constraint violations as the primary means to rank the individuals. This adaptive formulation allows further exploitation of the evolutionary information possessed by infeasible individuals with low objective values and low constraint violation. Involving the infeasible individuals in the evolutionary process helps the algorithm to find additional feasible individuals even in cases where the feasible space is very small or discontinuous. Furthermore, since there is no parameter tuning in the design of constraint handling, this makes the algorithm easy to implement. Moreover, the additional evaluations are simple arithmetic operations and do not impose any significant increase in the computational cost.

The performance of the algorithm is tested on fourteen constrained multi-objective test problems. From the simulation results, it is observed that the algorithm is capable of finding better-fit feasible solutions that are well spread over the Pareto front in all the runs of the test problems. In addition, the results of the algorithm are compared with some of the constrained multi-objective algorithms suggested so far. The comparison results indicate that the proposed algorithm performs better than the other algorithms in that it is able to provide a well distributed Pareto front that has optimal individuals. Moreover, the proposed algorithm provides solutions that extend very well to the maximum allowable limits over the Pareto front. For future work, the authors recommend applying the proposed constraint handling technique using modified objective function formulation ("distance" and "penalty") for other multi-objective evolutionary approaches other than NSGA-II.

Lastly, the authors recommend further work on combining the two proposed evolutionary algorithms into a single algorithm to provide an optimization approach for solving the more general dynamic, constrained multi-objective optimization problems.

REFERENCES

[1] B. Tessema, "A self adaptive genetic algorithm for constrained optimization," *Master's Thesis*, Oklahoma State University, Stillwater, OK, 2006.

[2] B. P. Buckles and F. E. Petry, "Genetic Algorithms," *Technology Series*, IEEE Computer Society Press, 1992.

[3] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments – a survey," *IEEE Transaction on Evolution Computation*, **9**(6), pp. 303-317, 2005.

[4] Y. Jin, "Evolutionary computation in dynamic and uncertain environments," in *IEEE Tutorial Evol. Comput. Conf.*, Honda Research Institute Europe, Germany, 2004.

[5] R. W. Morrison and K. A. De Jong, "A test problem generator for non-stationary environments," in *Proceedings of Congress on Evolutionary Computation*, Washington, DC, pp. 2047-2053, 1999.

[6] R. Merton, "*Continuous-Time Finance*," Basil Blackwell, Oxford, UK, 1990.

[7] J. Xu, P. B. Luh, F. B. White, E. Ni, and K. Kasiviswanathan, "Power portfolio optimization in deregulated electricity markets with risk management," *IEEE Transaction on Power Systems,* **21**(4), Nov. 2006.

[8] J. Branke, "*Evolutionary Optimization in Dynamic Environments*," Norwell, MA: Kluwer Publishing, 2001.

[9] S. Yang, "Constructing dynamic test environments for genetic algorithms based on problem difficulty", in *Proceedings of IEEE Congress on Evolutionary Computation*, pp. 1262-1269, 2004.

[10] K. Weicker and N. Weicker, "Dynamic rotation and partial visibility," in *Proceedings of IEEE Congress on Evolutionary Computation*, La Jolla, CA, pp. 1125-1131, 2000.

[11] K. Krishnakumar, "Micro-genetic algorithms for stationary and non-stationary function optimization," in *Proceedings of SPIE Conference on Intelligent Control and Adaptive Systems*, vol. 1196, pp. 289-296, 1989.

[12] J. J. Grefenstette, "Genetic algorithms for changing environments," in *Proceedings of International Conference on Parallel Problem Solving from Nature*, Amsterdam, The Netherlands: Elsevier, pp. 137-144, 1992.

[13] H. C. Andersen, "An investigation into genetic algorithms, and the relationship between speciation and the tracking of optima in dynamic functions," *Honors thesis*, Queensland University of Technology, Brisbane, Australia, 1991.

[14] A. Ghosh, S. Tsutsui, and H. Tanaka, "Function optimization in non-stationary environment using steady state genetic algorithms with aging of individuals," in *Proceedings of IEEE Congress on Evolutionary Computation*, pp. 666-671, 1998.

[15] Y. Jin, M. Husken, M. Olhofer, and B. Sendhoff, "Neural Networks for fitness approximation in evolutionary optimizations," in *Knowledge Incorporation in Evolutionary Optimization*, Y. Jin, Ed., Berlin, Germany: Springer-Verlag, pp. 281-306, 2004.

[16] N. Mori, H. Kita, and Y. Nishikawa, "Adaptation to a changing environment by means of the feedback thermodynamical genetic algorithm," in *Proceedings of International Conference on Parallel Problem Solving from Nature*, Berlin, Germany: Springer-Verlag, vol. 1498, pp. 149-158, 1998.

[17] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *Proceedings of IEEE Congress on Evolutionary Computation*, Washington, DC, pp. 1875-1882, 1999.

[18] A. Acan and Y. Tekol, "Ants can play prisoner's dilemma," in *Proceedings of IEEE Congress on Evolutionary Computation*, pp. 1348-1354, 2003.

[19] K. Trojanowski and Z. Michalewicz, "Searching for optima in non-stationary environments," in *Proceedings of IEEE Congress on Evolutionary Computation*, Washington, DC, pp. 1843-1850, 1999.

[20] C. N. Bendtsen and T. Krink, "Dynamic memory model for non-stationary optimization," in *Proceedings of IEEE Congress on Evolutionary Computation*, Honolulu, HI, pp. 145-150, 2002.

[21] R. E. Smith, "Diploid genetic algorithms for search in time varying environments," in *Annual Southeast Regional Conference of the ACM*, pp. 175-179, 1987.

[22] D. E. Goldberg and R. E. Smith, "Non-stationary function optimization using genetic algorithms with dominance and diploidy," in *Proceedings of 2nd International Conference on Genetic Algorithms*, pp. 59-68, Lawrence Erlbaum Associates, 1987.

[23] D. Dasgupta and D. R. McGregor, "Non-stationary function optimization using the structured genetic algorithm," in *Proceedings of International Conference on Parallel Problem Solving from Nature*, pp. 145-154, Elsevier Science Publisher, 1992.

[24] K. P. Ng and K. C. Wong, "A new diploid scheme and dominance change mechanism for non-stationary function optimization," in *Proceedings of 6th International Conference on Genetic Algorithms*, pp. 159-166, 1995.

[25] J. Lewis, E. Hart, and G. Ritchie, "A comparison of dominance mechanisms and simple mutation on non-stationary problems," in *Proceedings of International Conference on Parallel Problem Solving from Nature*, vol. 1498, Springer, pp. 139-148, 1998.

[26] C. Ryan, "Diploidy without dominance," in *Proceedings of 3rd Nordic Workshop on Genetic Algorithms*, pp. 63-70, 1997.

[27] F. Oppacher and M. Wineberg, "The shifting balance genetic algorithm: Improving the GA in a dynamic environment," in *Proceedings of Conference on Genetic and Evolutionary Computation*, vol. 1, pp. 504-510, 1999.

[28] J. Branke, T. Kaubler, C. Schmidt, and H. Schmeck, "A multi-population approach to dynamic optimization problems," in *Adaptive Computing in Design and Manufacturing 2000*, Berlin, Germany: Springer-Verlag, 2000.

[29] R. K. Ursem, "Multi-national GA optimization techniques in dynamic environments," in *Proceedings of Conference on Genetic and Evolutionary Computation.*, pp. 19-26, 2000.

[30] H. G. Cobb and J. J. Grefenstette, "Genetic algorithms for tracking changing environments," in *Proceedings of 5th International Conference Genetic Algorithms*, pp. 523-530, 1993.

[31] F. Vavak, T. C. Fogarty, and K. Jukes, "A genetic algorithm with variable range of local search for tracking changing environments," in *Proceedings of 4th International Conference on Parallel Problem Solving from Nature*, vol. 1141, Berlin, Germany, pp. 376-385, 1996.

[32] P. J. Angeline, "Tracking extrema in dynamic environments," in *Proceedings of 6th International Conference on Evolutionary Programming*, Springer, vol. 1213, pp. 335-345, 1997.

[33] T. Back, "On the behavior of evolutionary algorithms in dynamic environments," in *Proceedings of IEEE Congress on Evolutionary Computation*, Anchorage, AK, pp. 446-451, 1998.

[34] T. Sasaki and M. Tokoro, "Adaptation under changing environments with various rates of inheritance of acquired characters," in *Proceedings of Artificial Life and Robotics*, vol. 1585, pp. 34-41, 1998.

[35] T. Nanayakkara, K. Watanabe, and K. Izumi, "Evolving in dynamic environments through adaptive chaotic mutation," in *Proceedings of 3rd International Symposium on Artificial Life and Robotics*, pp. 520-523, 1999.

[36] K. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," *PhD thesis*, University of Michigan, Ann Arbor, MI, 1975.

[37] J. J. Grefenstette, "Evolvability in dynamic fitness landscapes: A genetic algorithm approach," in *Proceedings of IEEE Congress on Evolutionary Computation*, Washington, DC, pp. 2031-2038, 1999.

[38] B. Tessema and G.G. Yen, "A self-adaptive constrained evolutionary algorithm," in *Proceedings of IEEE Congress on Evolutionary Computation*, Vancouver, BC, pp. 246-253, 2006.

[39] C.M. Fonseca and P.J. Fleming, "Genetic algorithms for multi-objective optimization: formulation, discussion, and generalization," in *Proceedings of International Conference on Genetic Algorithms*, Urbana-Champaign, IL, pp. 416-423, 1993.

[40] N. Srinivas and K. Deb, "Multi-objective function optimization using non-dominated sorting genetic algorithms," *Evolutionary Computation*, **2**(3), pp. 221-248, 1994.

[41] J.D. Knowles and D.W. Corne, "Approximating the non-dominated front using the Pareto archived evolution strategy," *Evolutionary Computation*, **8**(2), pp. 149-172, 2000.

[42] E. Zitzler and L. Thiele, "An evolutionary algorithm for multi-objective optimization: the strength Pareto approach," *Technical Report*, Switzerland Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Munich, Switzerland, 1998.

[43] T. Bäck and F. Hoffmeister and H. Schwefel, "A survey of evolution strategies," in *Proceedings of the International Conference on Genetic Algorithms*, San Diego, CA, pp. 2-9, 1991.

[44] J. Joines and C. Houck, "On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs," in *Proceedings of the Congress on Evolutionary Computation*, Orlando, FL, pp. 579-584, 1994.

[45] J.C. Bean and A.B. Alouane, "A dual genetic algorithm for bounded integer programs," *Technical Report TR 92-53*, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, MI, 1992.

[46] R. Farmani and J. Wright, "Self-adaptive fitness formulation for constrained optimization," *IEEE Transaction on Evolutionary Computation*, **7**(5), pp. 445-455, 2003.

[47] A.C.C. Lemonge and H.J.C. Barbosa, "An adaptive penalty scheme in genetic algorithms for constrained optimization problems," in *Proceedings of Genetic and Evolutionary Computation Conference*, New York, NY, pp. 287-294, 2002.

[48] K. Deb, "An efficient constraint handling methods for genetic algorithms," *Computer Methods in Applied Mechanics and Engineering*, **186**(2), pp. 311-338, 2000.

[49] T.P. Runarsson and X. Yao, "Stochastic ranking for constraint evolutionary optimization," *IEEE Transaction on Evolutionary Computation*, **4**(3), pp. 344-354, 2000.

[50] T. Takahama and S. Sakai, "Constrained optimization by applying the α-constrained method to the nonlinear simplex method with mutations," *IEEE Transactions on Evolutionary Computation,* **9**(5), pp. 437-451, 2005.

[51] T. Takahama and S. Sakai, "Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elite," in *Proceedings of the IEEE Congress on Evolutionary Computation*, Vancouver, Canada, pp. 308-315 , 2006.

[52] Y. Wang and Z. Cai, "A multi-objective optimization based evolutionary algorithm for constrained optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation*, Edinburgh, UK, pp. 1081-1087, 2005.

[53] S. Venkatraman and G.G. Yen, "A generic framework for constrained optimization using genetic algorithms," *IEEE Transaction on Evolutionary Computation*, **9**(4), pp. 424-435, 2005.

[54] J. Aidanpaa, J. Anderson and A. Angantyr, "Constrained optimization based on a multi-objective evolutionary algorithm," in *Proceedings of Congress on Evolutionary Computation*, Canberra, Australia, pp. 1560-1567, 2003.

[55] C.A.C. Coello and A.D. Christiansen, "MOSES: a multi-objective optimization tool for engineering design," *Engineering Optimization*, **31**(3), pp. 337-368, 1999.

[56] T.T. Binh and U. Korn, "MOBES: A multi-objective evolution strategy for constrained optimization problems," in *Proceedings of International Conference on Genetic Algorithms*, East Lansing, MI, pp. 176-182, 1997.

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

[58] F. Jimenez, A.F. Gomez-Skarmeta, G. Sanchez and K. Deb, "An evolutionary algorithm for constrained multi-objective optimization," in *Proceedings of Congress on Evolutionary Computation*, Honolulu, HI, pp. 1133-1138, 2002.

[59] T. Ray, K. Tai and K.C. Seow, "An evolutionary algorithm for multi-objective optimization," *Engineering Optimization*, **33**(3), pp. 399-424, 2001.

[60] D. Chafekar, J. Xuan and K. Rasheed, "Constrained multi-objective optimization using steady state genetic algorithms," in *Proceedings of Genetic and Evolutionary Computation Conference*, Chicago, IL, pp. 813-824, 2003.

[61] A. Osyczka and S. Kundu, "A new method to solve generalized multi-criteria optimization problems using the simple genetic algorithm," *Structural Optimization*, **10**(2), pp. 94-99, 1995.

[62] V. Chankong and Y.Y. Haimes, "*Multi-Objective Decision Making Theory and Methodology*," New York: North-Holland, 1983.

[63] M. Tanaka, H. Watanabe; Y. Furukawa, and T. Tanino, "GA-based decision support system for multi-criteria optimization," *in IEEE International Conference on Systems, Man and Cybernetics*, vol. 2, Vancouver, BC, pp.1556-1561, 1995.

[64] K. Deb, "*Multi-Objective Optimization Using Evolutionary Algorithms*," Chichester, UK: Wiley, 2001.

VITA

Yonas Gebre Woldesenbet

Candidate for the Degree of

Master of Science

Thesis: UNCERTAINITY AND CONSTRAINT HANDLING IN EVOLUTIONARY ALGORITHMS

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Addis Ababa, Ethiopia, on February 25, 1982.
Current Address: 36 S University Pl, Apt 4, Stillwater, OK.

Education: Graduated as Valedictorian from St. Joseph High School, Addis Ababa, Ethiopia, in June 1999.
Received Bachelors of Science degree in Electrical Engineering from Bahir Dar University, Bahir Dar, Ethiopia, in July 2004 with a Gold-Medal for outstanding academic achievement.
Completed the requirements for the Masters of Science degree with major in Electrical Engineering at Oklahoma State University, Stillwater, OK in July 2007.

Experience: CEAT Lab Assistant, Oklahoma State University, Stillwater, OK.
Teaching Assistant, Oklahoma State University, Stillwater, OK.
Simulation and Software Engineer, Danotek Motion Technologies, Addis Ababa, Ethiopia.
Intern, City Business Computers, Addis Ababa, Ethiopia.

Professional Memberships: IEEE Computational Intelligence Society.

Name: Yonas Gebre Woldesenbet Date of Degree: July, 2007

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: UNCERTAINTY AND CONSTRAINT HANDLING IN EVOLUTIONARY ALGORITHMS

Pages in Study: 82 Candidate for the Degree of Master of Science

Major Field: Electrical Engineering

Scope and Method of Study: This paper proposes two evolutionary algorithms. Firstly, a dynamic evolutionary algorithm is proposed that uses variable relocation vectors to adapt the current population to the new environment. The relocation vectors introduce a certain radius of uncertainty to be applied specifically to each individual and in effect restoring diversity and accelerating exploration. Furthermore, the algorithm provides higher re-usage, faster convergence and better adaptation. As a technique to be used at transient periods, the proposed algorithm provides the next evolutionary cycle with better initial population than any other randomly generated population. The algorithm can be easily integrated into standard evolutionary algorithms and other uncertainty handling techniques.

Secondly, this paper proposes a new constraint handling technique for multi-objective evolutionary algorithms based on adaptive penalty functions and distance measures. Through this design, the objective space is modified to account for the performance and constraint violation of each individual. The modified objective functions are used in the non-dominance sorting to facilitate in evolution of optimal solutions not only in the feasible space but also in the infeasible space. The number of feasible individuals in the population is used to guide the search process either toward finding more feasible solutions or toward locating optimal solutions. The proposed method is simple to implement and does not need any parameter tuning.

Findings and Conclusions: The performance of each proposed algorithm is tested on several benchmark problems and both algorithms have shown superior results with lesser computational cost.

ADVISER'S APPROVAL: Dr. Gary Yen