Adaptive Mutation Operators for Evolutionary Algorithms

by

Imtiaz Ali Korejo

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the
Department of Computer Science
University of Leicester

2011

Declaration of Authorship

The content of this submission was undertaken in the Department of Computer Science, University of Leicester, and supervised by Dr. Shengxiang Yang during the period of registration. I hereby declare that the materials of this submission have not previously been published for a degree or diploma at any other university or institute. All the materials submitted for assessment are from my own research, except the reference work in any format by other authors, which are properly acknowledged in the content.

Part of the research work presented in this submission has been published or is under preparation to be submitted for publication in the following papers:

1. S. Yang and I. Korejo. Multi-population methods with adaptive mutation for multi-modal optimization problems. To be submitted for journal publication, 2012.
2. I. Korejo, S. Yang, and C. Li. A directed mutation operator for real coded Genetic Algorithms. *Proceedings of the 10th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoApplications 2010)*, LNCS 6024, pp. 491-500, 2010. Springer.
3. I. Korejo, S. Yang, and C. Li. A comparative study of adaptive mutation operator for function optimization. *Post-Conference Volume of the 2009 Metaheuristics International Conference (MIC 2009)*, 2011.
4. C. Li, S. Yang, and I. Korejo. An adaptive mutation operator for particle swarm optimization. *Proceedings of the 2008 UK Workshop on Computational Intelligence*, pp. 165–170, 2008.

Adaptive Mutation Operators for Evolutionary Algorithms

by

Imtiaz Ali Kojero

Abstract

Evolutionary algorithms (EAs) are a class of stochastic search and optimization algorithms that are inspired by principles of natural and biological evolution. Although EAs have been found to be extremely useful in finding solutions to practically intractable problems, they suffer from issues like premature convergence, getting stuck to local optima, and poor stability. Recently, researchers have been considering adaptive EAs to address the aforementioned problems. The core of adaptive EAs is to automatically adjust genetic operators and relevant parameters in order to speed up the convergence process as well as maintaining the population diversity.

In this thesis, we investigate adaptive EAs for optimization problems. We study adaptive mutation operators at both population level and gene level for genetic algorithms (GAs), which are a major sub-class of EAs, and investigate their performance based on a number of benchmark optimization problems. An enhancement to standard mutation in GAs, called directed mutation (DM), is investigated in this thesis. The idea is to obtain the statistical information about the fitness of individuals and their distribution within certain regions in the search space. This information is used to move the individuals within the search space using DM. Experimental results show that the DM scheme improves the performance of GAs on various benchmark problems.

Furthermore, a multi-population with adaptive mutation approach is proposed to enhance the performance of GAs for multi-modal optimization problems. The main idea is to maintain multi-populations on different peaks to locate multiple optima for multi-modal optimization problems. For each sub-population, an adaptive mutation scheme is considered to avoid the premature convergence as well as accelerating the GA toward promising areas in the search space. Experimental results show that the proposed multi-population with adaptive mutation approach is effective in helping GAs to locate multiple optima for multi-modal optimization problems.

Acknowledgements

Starting in the name of Almighty ALLAH, who is the most kind and merciful to mankind, this is a great opportunity for me to thank those people whose direct or indirect contribution has made the writing of this thesis possible for me. The most significant contributions come from my research supervisor Dr. Shengxiang Yang, who is not only a great person but also a great teacher and supervisor. During this period of four years, he provided encouragement, suggestions, good company, and a lot of good ideas, and supervised me in a manner which has led to the successful completion of my PhD study.

I would also like to thank those people whose contribution also played an important role in my success. I wish to express my warm and sincere thanks to all the faculty members, especially, Prof. Rajeev Raman, Prof. Rick Thomas, Prof. Thomas Erlebach, Dr. Fer-Jan de Vries, and Dr. Stanley P Y Fung, for their moral and technical support, advice, encouragement, and keen interest in assessing my yearly reports, presented to them, from time to time. My sincere gratitude to the administrative staff for their supportive behaviour, and swift and effective solutions of the issues that emerged during the course of my PhD study.

It is very important and necessary to offer my gratitude to my mother university and employer, i.e., University of Sindh, Jamshoro, Sindh, Pakistan. All the financial support for this study was provided by my university, as I was awarded a foreign scholarship for my PhD research under the faculty strengthening and development program of the Higher Education Commission Pakistan. It was simply impossible for me to complete my study here, if I would have not been provided this opportunity.

I would also like to thank those colleagues/friends here in Leicester, who expended their time and shared their knowledge, in order to help and improve my research work, especially Changhe Li, who provided me technical support, and Muhammad Muzammal, who provided me a wonderful company and emotional support, whenever I felt homesickness or lonely in this foreign land, thousand miles away from my home.

I also wish to extend my thanks to my colleagues and friends here in the Department of Computer Science, who shared their joys and happy moments with me.

Their kindness, generosity, and help made my life easier in Leicester, whenever I felt stranger here in different culture and atmosphere.

In the end, I would like to thank my entire family for providing me a loving environment during my entire life. My mother, my brothers, and my wife, in particular, were very supportive in my absence and managed many troubles on their own, without putting me into any stress. They always prayed for me to successfully complete my PhD study. I would like to present this thesis as a gift to my kids Kousar and Muhammad Ihsaan.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

ABC	**A**rtificial **B**ee **C**olony
ACO	**A**nt **C**olony **O**ptimization
AGA	**A**daptive **G**enetic **A**lgorithm
APGA	**G**enetic **A**lgorithm with **A**daptive **P**opulation size
AREA	**A**daptive **R**epresentation **E**volutionary **A**lgorithm
CMA-ES	**C**ovariance **M**atrix **A**daptation **E**volution **S**trategy
CNF	**C**onjunctive **N**ormal **F**orm
COBRA	**CO**st **B**ased operator **R**ate **A**daptation
DM	**D**irected **M**utation
DMGA	**D**ynamic **M**utation **G**enetic **A**lgorithm
DNA	**D**eoxyribo **N**ucleic **A**cid
DV	**D**irected **V**ariation
EA	**E**volutionary **A**lgorithm
EP	**E**volutionary **P**rogramming
ESs	**E**volutionary **S**trategies
GAs	**G**enetic **A**lgorithms
GAVaPS	**G**enetic **A**lgorithm **Va**rying **P**opulation **S**ize
GBAM	**G**ene **B**ased **A**daptive **M**utation
GBAM_FAD	**G**ene **B**ased **A**daptive **M**utation with **F**itness and **A**llele **D**istribution correlation
GEP	**G**ene **E**xpression **P**rogramming
GLAM	**G**ene-**L**evel **A**daptive **M**utation

GP	**G**enetic **P**rogramming
MAX-SAT	**MAX**imum **SAT**tisfiability
PLAM_GA	**P**opulation-**L**evel **A**daptive **M**utation **G**enetic **A**lgorithm
PLAM_PSO	**P**opulation-**L**evel **A**daptive **M**utation **P**article **S**warm **O**ptimization
PRoFIGA	**P**opulation **R**esizing **O**n **F**itness **I**mprovement **G**enetic **A**lgorithm
PSO	**P**article **S**warm **O**ptimization
PWAM	**P**artition based multi-population **W**ith **A**daptive **M**utation
SAGA	**S**elf-**A**daptive **G**enetic **A**lgorithms
SANUM	**S**tatistics-based **A**daptive **N**on-**U**niform **M**utation
SANUX	**S**tatistics-based **A**daptive **N**on-**U**niform **X:**crossover
SGA	**S**imple **G**enetic **A**lgorithm
SPSO	**S**pecies-based **P**article **S**warm **O**ptimization
SSRGA	**S**ite **S**pecific **R**ate **G**enetic **A**lgorithm
SWAM	**S**pecies based multi-population **W**ith **A**daptive **M**utation

Symbols

α, β	parameters for SANUM
α	reduction factor
δ	minimum selection ratio for each operator in PLAM_GA
ϵ	distance threshold
η_1, η_2	acceleration constants in PSO algorithms
γ	minimum selection ratio for each operator in PLAM_PSO
μ	number of individuals
D	domain of a problem
f	fitness of an individual
$\bar{f}$	average fitness of the population
$\acute{f}$	fitness of an individual to be crossed
f_1	frequency of 1's in a locus
f_{max}	maximum fitness
f_{min}	minimum fitness
L	total number of intervals
(l, n)	length of individual, population size
max_{gen}	maximum number of generations
min_dist	minimum distance among individuals of a sub-population
N	number of mutation operators
n	number of dimensions
P_c	crossover probability
P_m	mutation probability

P_{max}	maximum probability
P_{min}	minimum probability
$P_{mutation}$	initial mutation probability of each mutation operator
r	radius of a species/sub-population
$sgn(x)$	sing function, which returns the value 1, 0, or -1
T	local search size for the global best particle
U_f	update frequency for each operator

Dedicated to my family...

Chapter 1

Introduction

This world is full of problems and many of these problems can be translated into optimization. Optimization is useful in various fields and plays an important role in engineering, management science, medicine, computer science, applied mathematics, and many more areas. The concept of optimization may be different for different fields. But, the main objective remains the same, namely, making an optimum decision. The applicability of optimization in many different disciplines makes it difficult to provide a single specific definition of optimization. Mathematicians, for instance, aim to get the maxima or minima of a real function within a set of variables. From the computing and engineering point of view, the objective of optimization is usually to maximize the system or application performance with minimal runtime and resources possible [112]. Remarks on optimization were given by Cherkaev [19] and we quote as follows:

> "the desire for optimality (perfection) is inherent for humans. **It seems a natural instinct to search for extremes in all endeavour of life** (personal emphasis). The search for extremes inspires mountaineers,

> scientists, mathematicians, and the rest of the human race. The mathematical theory of optimization has been developed since the sixties when computers became available. The goal of the theory is the creation of reliable methods to catch the extremum of a function by an intelligent arrangement of its evaluations. This theory is vitally important for modern engineering and planning that incorporate optimization at every step of the complicated decision making process."

Global optimization is a branch of applied mathematics and numerical analysis, which deals with the optimization of a function or a set of functions for finding the best possible solution(s) from a solution set. Simple conventional global optimization problems can be solved by using deterministic algorithms. However, if the objective function of a problem is not differentiable (if there is no clear difference in between local and global optimal solutions), has too many local optima, and/or has a very high dimensional search space, it becomes impossible to apply deterministic approaches to solve these global optimization problems. Due to the above properties, it is impossible for deterministic methods to enumerate all local optima within an endurable time. Due to this reason, probabilistic algorithms are required to achieve the best possible solution(s), which may be a bit inferior to the global optimum, instead of the global optimum, which needs, for instance, 10^{100} years, to be found.

Evolutionary algorithms (EAs) are stochastic search and optimization methods which are inspired by principles of natural evolution and genetics. EAs have been used for solving different optimization problems due to the properties of self-learning, self-organization, and self-adaptation, as well as the property of implicit parallelism. In the past few decades, different approaches were introduced along with their properties. Genetic algorithms (GAs) [49], evolution strategies (ESs) [72], evolutionary programming (EP) [36], and genetic programming (GP) [58] are major variants of

EAs. Some other schemes are resemblance with the above branches of EAs. These methods, e.g., particle swarm optimization (PSO) [27, 57], ant colony optimization (ACO) [22], and artificial bee colony (ABC) [56], have also become important research areas.

When EAs are applied for a specific problem, we first need to consider the search space of candidate solutions to that problem and decide upon a representation for the solutions. Then, we need to determine a set of relevant parameters. After that, we can initialize a population of candidate solutions to the problem, which are usually randomly generated. New solutions are generated from the current population by using selection, recombination and mutation operators. For instance, for a conventional genetic algorithm, we may decide to employ binary representation, uniform crossover, bit-flip mutation, tournament selection, and generational replacement. In addition, further more information of relevant parameters is needed for a full specification of an EA, e.g., the population size, the probability of mutation, the probability of crossover, and the tournament size for the tournament selection scheme. These relevant parameters are called **algorithm parameters** or **strategy parameters**. The values of these parameters greatly affect the behaviour of an algorithm, e.g., whether it will find an optimal or near-optimal solution, and whether it will search such a solution efficiently. Selecting suitable parameter values is a very difficult task, if not impossible.

Generally speaking, there are two major classes of methods to set parameter values: **parameter tuning** and **parameter control**. Parameter tuning sets parameters before the execution of an algorithm while parameter control adjusts parameters during the running process of an algorithm.

Adaptation of strategy parameters and genetic operators has become an important and promising research area in the domain of EAs. Nowadays, many researchers are focusing on solving numerous optimization problems by using adaptive techniques.

The objective of adaptive algorithms is to modify the genetic operators and relevant parameters to maintain the diversity of the population and obtain good results to a problem within a reasonable amount of time.

1.1 Motivation

Adapting the values of strategy parameters of an EA in order to achieve good performance has become an important and promising research area in EAs. Some of the typical strategy parameters are the population size, the number of crossover points, the probabilities for crossover and mutation operators. These parameters can be configured by experimenting with different values and choosing the suitable one that provides the best results on the optimization problem in hand. Typically, one parameter is modified at a time, which may lead to an algorithm getting stuck on a local optimum, since usually it is not known how the parameters are related to each other. However, various parameters and their different values determine a lot of combinations. Therefore, it is a time-consuming activity. For instance, given four parameters and five values associated with each of them, there are $5^4 = 625$ distinct setups. If an algorithm is allowed 100 independent runs for each setup, 62,500 runs are needed just to develop a good algorithm design.

Another parameter tuning approach is to find the values of parameters before the execution of an algorithm. This rigid form of static parameter setting contrasts the dynamic nature of EAs. For example, a large mutation step can help the exploration of the search space in the early generations and a small mutation step may be useful for an EA to exploit a more accurate solution in the late generations. This is impossible with conventional fixed parameters approaches.

A number of studies have addressed the issue empirically and theoretically, showing that different values of parameters and operators may be optimal at different stages of the evolutionary process of an EA [7–9, 25, 30, 47, 79, 81, 91, 92]. Due to this reason, static parameter setting (i.e., parameter tuning) has been gradually discouraged in the EA community. In order to tackle the limitations of static parameter setting for EAs, a straightforward way is to replace a parameter p by a function $p(t)$, where t is the index of the current generation (or any other method of counter). In this (deterministic) approach, the parameter value $p(t)$ is modified by a "blind" rule, activated by the current generation value t, without taking any information of the actual progress in solving the problem, i.e., without using any feedback information of the current state of the search process. This may not be appropriate for EAs.

Another approach, which overcomes the limitations of fixed parameters, is to use adaptive techniques. Parameters are modified by taking into account the current state of the search process. Different parameters and operators can be updated in this mechanism by using the quality of solutions during the evolutionary process. It is necessary to differentiate between deterministic and adaptive schemes, as the second one uses feedback information from the search space and the first one does not.

1.2 Aims and Objectives

The primary aim of this thesis is to test the hypothesis of effective adaptive mutation operators for EAs for global optimization problems. GAs and particle swarm optimization (PSO) are used for the implementation of these adaptive approaches.

The main objectives of this thesis are summarized as follows:

1. To study in detail adaptive mutation approaches for EAs.

2. To compare adaptive mutation techniques for GAs and PSO algorithms.

3. To propose new ideas of adaptive mutation operators and apply them to improve the performance of GAs and PSO algorithms for global optimization problems.

4. To obtain empirical results to validate our introduced ideas.

1.3 Methodology

Nowadays, a few researchers presented analytical proof of EA approaches on very simple problems. But, it is quite difficult to analyse most EAs. Generally speaking, it is very difficult, if not impossible, to give an accurate analysis of the convergence of these algorithms. Hence, one can not perform an accurate behaviour analysis of EA approaches and predict their general search behaviour, which depends on their working mechanism as well as the problem being solved.

Another important method to test the performance of EAs is to observe and analyse the empirical results on benchmark problems. Different benchmark problems have been suggested to test different sort of properties of EAs in the literature. It is very hard to measure the performance of algorithms on all benchmark problems. In the common practice, an algorithm's performance is investigated on various benchmark problems that have distinct characteristics.

In this thesis, we will follow the second research method, i.e., evaluating and justifying the performance of algorithms based on empirical results on benchmark problems. The adaptive mutation schemes to be investigated in this thesis will be tested on a series of well-known benchmark optimization problems taken from the literature. These optimization problems will be explained in Chapters 4, 5, and 6, respectively. Furthermore, comparing the performance of proposed algorithms with other existing algorithms under the same criteria is also carried out in this thesis.

1.4 Contributions

Different mutation techniques are suggested and investigated in this thesis to improve the performance and reduce the disadvantages of conventional EAs, e.g., to maintain the diversity in the population by adaptive mutation, directed mutation, and multi-population with adaptive mutation schemes. These mutation approaches introduce new individuals into the population, which are guided by the feedback information from the current generation.

From the research, the following contributions are made:

1. An experimental analysis shows that an adaptive mutation operator proposed for PSO can greatly improve the performance of PSO on a set of benchmark optimization problems. In the adaptive mutation operator for PSO, each solution has a set of three operators to cope with different situations in the search process. The interaction of the three operators is used by an adaptive framework at the population level: each particle can select a suitable operator at the appropriate level according to the property of its local search space for attaining the best performance by the adaptive approach. It is possible that different operators may be optimal at different stages of an evolutionary process on different optimization problems.

2. We review a number of techniques that have been introduced to adjust the mutation probabilities based on the global behaviour of the population during the evolutionary process. This approach investigates several adaptive mutation operators, including the population level and gene level adaptive mutation

operators, for PSO and GAs and compare their performance based on a set of uni-modal and multi-modal benchmark functions. The gene level adaptive mutation operators show great advantage over the traditional fixed mutation probability operators and the population level adaptive mutation operator for GAs. The adaptive mutation operator for PSO which is also a population level operator outperforms the gene level adaptive mutation and population level adaptive mutation operator for GAs.

3. A directed mutation technique is suggested for GAs to address uni-modal and multi-modal problems. This approach introduces new solutions in the search space guided by the information acquired in the previous generations. The effectiveness of the new scheme is investigated on different benchmark optimization problems and compared with other approaches from the literature. The result shows that the directed mutation mechanism is able to produce good results on various benchmark problems.

4. A multi-population with adaptive mutation approach is proposed for GAs to locate the multiple peaks of multi-modal optimization problems. In order to address the convergence problem of GAs, the multi-population with adaptive mutation approach can be used to maintain the diversity, guide the algorithms for fast convergence, and prevent GAs from being trapped into local optima. The developed GA is investigated on benchmark problems with different levels of difficulties in comparison with other four adaptive algorithms taken from the literature. The experimental results show that the proposed GA is a good choice for locating multiple optima in multi-modal optimization problems.

1.5 Structure of the Thesis

The rest of this thesis is organised as follows.

Chapter 2 introduces the concept of optimization and identifies the maximization and minimization problem and its local and global optimum. The overview of global optimization algorithms and their classification are mentioned in this chapter. Evolutionary computation, especially genetic algorithms and their components are also explained in this chapter.

Chapter 3 starts with an introduction of adaptation, followed by the short history of adaptation and taxonomy of adaptation in evolutionary algorithms. Afterwards, this chapter provides a review of empirical research of adaptive methods applied in GAs, evolutionary strategies and evolutionary programming (in general EAs).

The comparative study and performance of population level and gene level different adaptive mutation schemes of GAs and particle swarm optimization are explained in Chapter 4, these approaches are tested on several distinct benchmark global optimization problems.

In Chapter 5, directed mutation is used to modify conventional mutation operator in GAs along with discussions on some common issues, e.g., how to get the lost information during the early generations, how to increase the diversity in the population, and how to get the permanently lost genetic information. These common issues can be solved by the directed mutation approach.

The multi-population with adaptive mutation approach proposed in this thesis is described in Chapter 6. This technique can be used to reduce the limitations of GAs on multi-modal optimization problems. It can help GAs to reduce the probability of getting stuck to local optima, maintain the diversity in the population, and hence increase the performance and locate multiple optima in multi-modal optimization problems.

Finally, Chapter 7 summarizes the research carried out in this thesis. Some directions for future research of adaptive mutation operators in EAs are also discussed.

Chapter 2

Global Optimization and Evolutionary Algorithms

This chapter presents an overview of optimization and global optimization methods, an explanation of evolutionary computation, a detailed description of the key genetic operators and relevant parameters of genetic algorithms (GAs), which are a major class of evolutionary algorithms (EAs) and are the major concern of the research in this thesis. We also provide a brief introduction to computational complexity and statistical test, which are relevant to the study of global optimization and EAs, at the end of this chapter.

2.1 Optimization

In general, optimization is the process of finding the best solution for a given problem. It is made up of three basic components [20, 59, 111]: the decision variables, which determine the components of the problem and can be modified to generate distinct possibilities; constraints, which specify the limitations on the variables; and the objective function, which determines the quality of individuals. The aim of an optimization problem is to find the best solution that satisfies the constraints and maximize(or minimize) the objective function.

2.1.1 Maximization problem

A maximization optimization problem is to maximize $f(x)$, where f is the real valued objective or cost function and $x \in S$ is a feasible solution, where S is the set of feasible solutions. A global optimum is a solution $x' \in S$ such that $f(x') \geq f(x), \forall x \in S$. The corresponding value of the objective function is denoted by f', i.e., $f' = f(x')$. The set of global optimal solutions S' is defined by $S' = \{x \in S | f(x) = f'\}$.

2.1.2 Neighbourhood and Local Optimum

Let S be the set of feasible solutions. For each solution $x \in S$, a solution $y \in S$ is in the neighbourhood of x if the distance D between x and y is less than or equal to a given distance threshold ϵ. The neighbourhood function $N(x)$ of x can be defined as follows: $N(x) = \{y \in S | D(x, y) \leq \epsilon\}$.

A solution is a local optimum if the objective function of the solution is the best among the solutions in its neighbourhood. Neighbourhood may help to find local optima of a problem. Hence, local optimum based on the neighbourhood approach is usually applied.

Let S be a set of feasible solutions and $N(x)$ be the neighbourhood of a solution $x \in S$. A solution $x' \in S$ is a local maximum w.r.t $N(x)$ if $f(x') \geq f(x), \forall x \in N(x')$ and local minimum w.r.t $N(x)$ if $f(x') \leq f(x), \forall x \in N(x')$. Local optimum can either be a local minimizer or a local maximizer.

2.1.3 Global Optimum

Let S be a set of feasible solutions. A solution $x' \in S$ is a global maximum if $f(x') \geq f(x), \forall x \in S$ or a global minimum if $f(x') \leq f(x), \forall x \in S$. A global optimum can either be a global minimum or a global maximum.

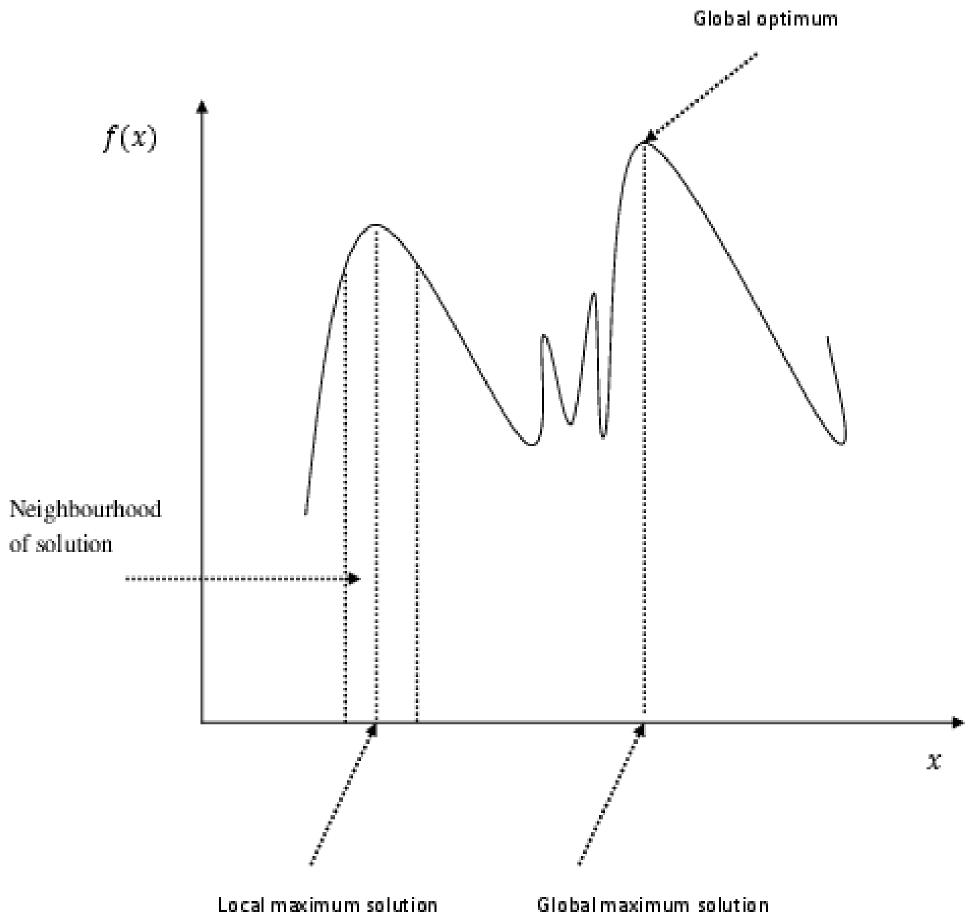


Figure 2.1: A maximization problem.

Figure 2.1 and Figure 2.2 illustrate the above points in the fitness landscape. In the context of maximization problems, a global optimum is the highest peak in the search space while other peaks with lower heights are local optima. There is no any other peak which is higher than a local optimum within its neighbourhood. There are various efficient algorithms for finding the candidate solutions to some kinds of

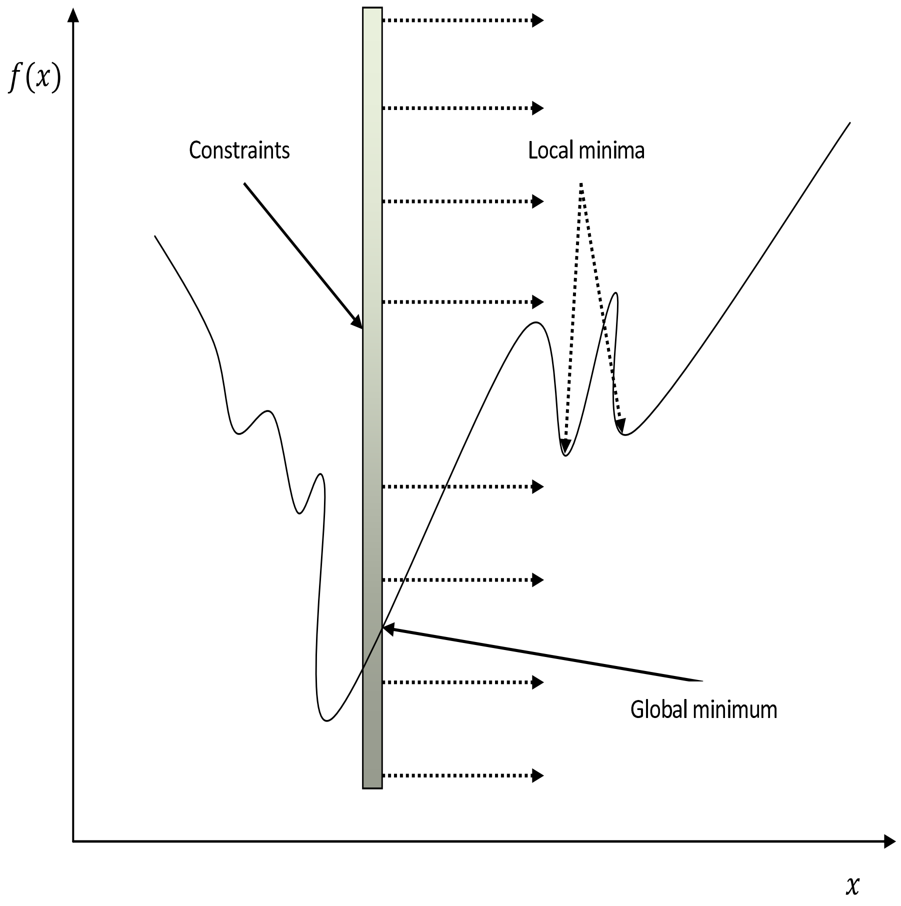


Figure 2.2: A constraint minimization problem.

optimization problems. For complex problems, the objective function becomes very difficult to search the global optima because of the properties like too many local optima, not continuous objective function, high dimensions of the objective function, and unclear differences between local and global optimal solutions. Therefore, algorithms face difficulty to find the optimal solutions in the search space for such complex problems and an algorithm may get stuck on local optima.

Non-linear optimization or non-linear programming is a branch of applied mathematics that deals with finding the maximum or minimum value of a function subject to constraints or restrictions on the variables of the function. There are a lot of applications of these problems in the real world, such as engineering design, space planning, networking, data analysis, logistic management, financial planning, and many more. These problems often require a global search approach to get their optimal solutions. Generally speaking, they are very hard to solve due to many reasons. First, as described in the literature, optimization problems from these applications are often NP-hard in nature [12, 21, 37, 123]. Secondly, in order to solve benchmark and real world problems by using some requirements and constraints, these problems are either hard (in the context of solvability) or conflicting [4, 12]. Other reasons were mentioned in [112], where some fundamental issues are considered for solving difficult optimization problems.

2.2 Global Optimization Methods

The goal of global optimization is to find the best candidate solution to a given problem within a reasonable time limit. It is a swiftly increasing area in many fields of study. These include engineering, management science, medicine, computer science, economics, biotechnology, computational chemistry, and applied mathematics, to bring up a few examples. Different fields of study may view global optimization from different perspectives. But, the overall goal of the whole process is the same, namely, making an optimum decision.

There are a wide range of different optimization techniques, which face difficulties to solve global optimization problems. In [112], the authors suggested some key features of optimization problems, which include ruggedness, causality, deceptiveness, epistasis, robustness, overfitting, over-simplification, and dynamic fitness. An optimization problem can be regarded as a decision making problem. As mentioned earlier, a particular optimization technique is only suitable for a specific type of problems.

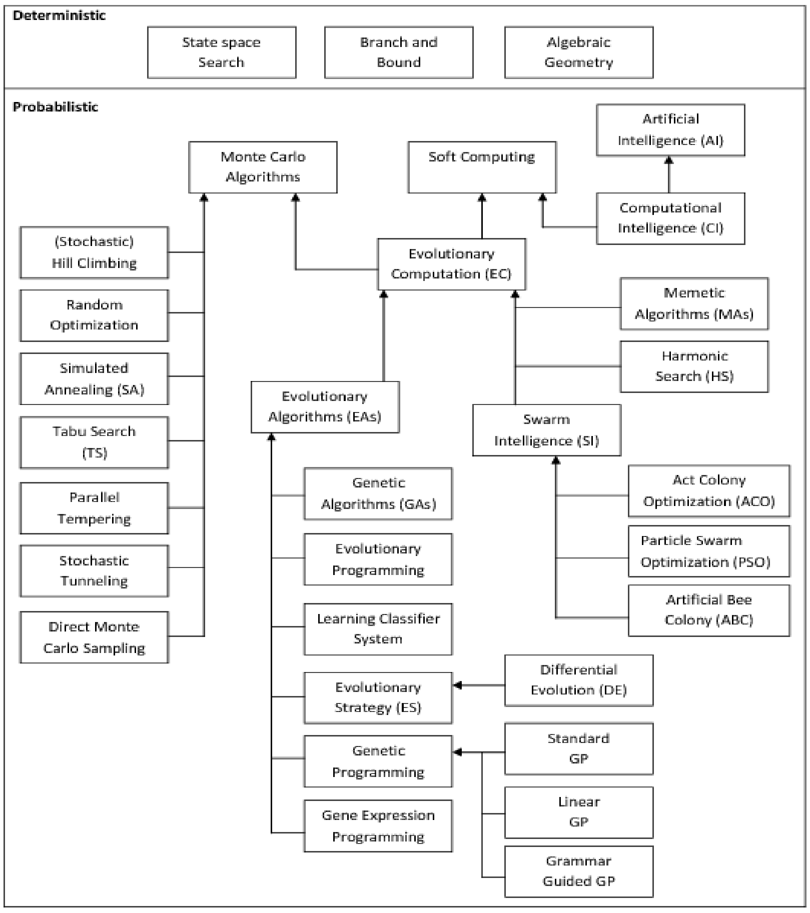


Figure 2.3: Taxonomy of global optimization algorithms.

The general view of global optimization methods is given in this section. Figure 2.3 illustrates a rough classification of global optimization algorithms by Weise *et al.* [111]. Generally speaking, global optimization algorithms can be classified into two basic classes: deterministic and probabilistic methods [111]. It becomes harder for deterministic algorithms to find the global optima, if the relation between a candidate solution and its fitness is not clear, the number of dimensions is very high, or the objective function is too complicated. In these cases, deterministic methods may look like a blind search in a black box where the fitness landscape is very complex; trying them in this case would possibly be an exhaustive enumeration of the search space or just searching in a local search space without the complex fitness landscape.

Probabilistic or Non-deterministic methods can be used when problems have a huge number of local optima, the fitness landscape is very complex, or the dimensionality of the search space is very high. These methods are used to achieve the optimal or near-optimal solution(s) within a reasonable time. Probabilistic methods have the advantages over deterministic methods due to the properties of simplicity and easy-to-use for finding good solutions in a complex search space.

Monte Carlo methods (or Monte Carlo experiments) are a class of computational algorithms that deal with random calculation and most stochastic schemes are included in the Monte Carlo approach. Heuristics and meta-heuristics play a very important role in most of probabilistic algorithms, which use the current information gathered during the evolutionary process to decide how to create a next candidate solution or which individual should be processed next. Generally speaking, this type of algorithms can use feedback information achieved from samples in the search space. Usually, they use some abstract models from natural phenomena. The most popular methods are simulated annealing and evolutionary algorithms (EAs).

2.3 Evolutionary Computation

Evolutionary computing is a rapidly growing area. It plays an important role within the field of computer science. Evolutionary computing uses computational models of evolutionary processes as key elements for designing and implementing computer-based problem solving systems. Several evolutionary computation models have been proposed and studied in the literature, which are known as evolutionary algorithms.

They use common properties of simulating the evolution of individual structures through the process of selection and recombination. These mechanisms depend on the perceived quality of the individual structures as determined by the environment. Over the past few decades, different approaches of EAs have been introduced with their properties; some of them are briefly explained in the following sections.

In the literature, various types of EAs have been proposed. All of them share the following basic properties: (i) An EA uses the collective learning process of a population of individuals. Each individual has its own representation in the search space for a given problem; (ii) Each individual is evaluated in its environment. The quality or fitness can be specified for individuals, and during the selection process fitter individuals have more chances to be selected for next generation than less quality individuals; (iii) offspring are generated by applying variation genetic operators (e.g., mutation and recombination).

2.3.1 Evolution Strategies (ESs)

In the 1960s, ESs were introduced by Rechenberg and Schwefel [80] in Berlin, Germany. They were considered as the search heuristic technique for solving optimization problems in the area of engineering. The genotype of solutions is represented by a vector of real values. The earliest evolutionary strategies contained only a single individual in the population, with the solution competing in each generation for survival with a single offspring. This scheme is called (1+1)-ES. The idea of Rechenberg was extended by Schwefel, who applied recombination and more than one solutions in the population, and determined a better comparison of ESs with previous optimization approaches. There are different versions of multi-membered ES or steady state ES, which are called $(\mu + \lambda) - ES$ and $(\mu, \lambda) - ES$. The general framework of multi-membered ESs is given in Algorithm 1.

Algorithm 1 The basic structure of evolution strategies

```
1: $t := 0$;
2: Initialize a population of $\mu_{parent}$ individuals;
3: Evaluate the fitness of each individual in the population;
4: while $t < max\_gen$ do
5:     Select and recombine parents from the population to generate offspring $\mu_{child}$;
6:     Mutate the offspring $\mu_{child}$;
7:     Evaluate the offspring $\mu_{child}$;
8:     Select the best $\mu_{parent}$ solutions from the offspring and parent populations
       according to the quality of solutions;
9:     Use the selected $\mu_{parent}$ offspring as parents for the next generation;
10:    $t := t + 1$;
11: end while
```

2.3.2 Evolutionary Programming (EP)

In [36], Fogel *et al.* first proposed the concept of EP, which is a stochastic technique for solving optimization problems. EP has been successfully applied in the field of numerical and combinatorial optimization problems [33–35]. The main motivation behind EP is to generate an alternative scheme to artificial intelligence. The different representations used in the traditional EP are tailored to the problem domain (see Spear *et al.* [95]). For example, each individual of the population can be a real-valued vector, which is specifically for real valued optimization problems, an ordered list, which is applied for solving the travelling salesman problem, or a finite state machine, which is used for graph applications.

After randomly generated population, the general selection scheme is used to choose all the solutions from the population to be the N parents. Then, the N parents are mutated to give the life of new offspring. These offspring are measured and N survivors are selected from the $2N$ solutions to form the next generation.

Algorithm 2 The general framework of evolutionary programming

```
1: $t := 0$;
2: initialize $P(t)$;
3: evaluate $P(t)$;
4: **repeat**
5:     select $P(t)$ from $P(t-1)$
6:     alter $P(t)$
7:     evaluate $P(t)$
8:     $t := t + 1$;
9: **until** The stop criterion is satisfied (or maximum number of generations)
```

2.3.3 Genetic Algorithms (GAs)

GAs are powerful search methods, which were inspired by Darwin's theory of survival of the fittest. GAs were first introduced by John Holland in 1960s in USA. Generally speaking, GAs have been successfully applied for solving many optimization problems due to the properties of easy-to-use and robustness for finding good solutions to difficult problems [39]. A set of individuals of population is first initialized and then evolved from generation to generation by an iterative process of evolution, selection, recombination, and mutation. The size of the population is usually constant during the evolutionary process. The basic structure of simple GAs is presented in Algorithm 3. The detailed description of GAs will be presented in the following section of this chapter.

2.3.4 Genetic Programming

In the mid eighties of the last century, John Koza first invented the idea of genetic programming (GP), which is a branch of EAs. It uses evolutionary processes to automatically generate programs. GP is a growing popular approach in the community of EAs. The representation of each GP solution is a tree structure that consists of functions (sub-trees) and values (leafs). The tree is composed of sub-trees. Each sub-tree is evaluated by resulting the measurement of its sub-trees. The whole tree

Algorithm 3 The general framework of GAs

```
1: Input: A problem instance
2: $t := 0$;
3: Randomly generate an initial population $P(t)$;
4: Evaluate the fitness of each individual of $P(t)$;
5: while $t < max\_gen$ do
6:    Selection $P(t)$;
7:    Crossover $P(t)$;
8:    Mutation $P(t)$;
9:    Evaluation $P(t)$;
10:   $t := t + 1$;
11: end while
12: Output: A (sub-optimal) solution
```

represents a single function, which can be evaluated according to a left-most depth-first manner. GP is almost similar to GA except the variation operators, which are determined according to a tree representation. The sub-tree crossover operator is used by taking randomly chosen complete sub-trees in two parent solutions and swapping them between the parents.

2.3.5 Particle Swarm Optimization

Particle swarm optimization (PSO) was first proposed in 1995 [27, 57]. PSO is a mechanism applied to explore the search space for the particular problem to find the settings or parameters required to maximize a specific objective, which is inspired from the social behaviour of organisms, due to the attributes of bird flocking and fish schooling. In PSO, a number of particles "fly" in the search space. Each particle in the swarm keeps track of the best position achieved by itself so far and the best position achieved by its topological neighbors in order to share the information with its topological neighbors and move toward the best position found by its neighbors.

Different models have been suggested to increase the success rate of finding the global optima. The *gbest* and *lbest* models are specified for the original PSO. In the *gbest* model, all the particles are neighbors to each other, which share information globally. Particles converge quickly with the help of the gbest mechanism because they are all attracted to a global best position found so far by the whole swarm. However, this model may get stuck to local optima due to the loss of diversity in the swarm. On the contrary, in the *lbest* model, only a fixed number of particles share information with a particle, and convergence occurs slowly as compared to *gbest*. Therefore, the *lbest* has a greater chance of finding the global optimum than the *gbest* model. These two models give different performances on different problems.

PSO has become a popular and efficient heuristic optimization tool for solving optimization problems. Recently, many researchers have focused their attention on this promising research area. The general concept regarding the PSO is that: it gives better results in a faster and cheaper way as compared to other techniques. There are few parameters to adjust in the PSO, which is the second reason to motivate researchers to use this tool to solve optimization problems. There are many applications using one version of PSO with slight variations.

As mentioned above, PSO stimulates the social behaviour of organisms, such as bird flocking and fish schooling. Before presenting the original PSO algorithm, the scenario presented by Hu in [52] is shown as follows:

> “a group of birds are randomly searching food in an area. There is only one piece of food in the area being searched. All the birds do not know where the food is. But they know how far the food is in each iteration. So what’s the best strategy to find the food? The effective one is to follow the bird which is nearest to the food.”

Ideally, PSO acquires information from scenario and apply it to solve optimization problems. In PSO, the “bird” represents the solution in the search space of the problems. Each solution of the swarm has fitness value, which is measured by the objective function at its current location which needs to be optimized, and has velocity, which guides in flying the particle.

There are several major versions of PSO algorithms. The following version, which is modified by Shi and Eberhart [82], is used in this thesis. The PSO algorithm maintains and evolves a population of particles. Each particle is represented by a position and a velocity, which are updated as follows:

$$\vec{v}_i' = \omega\vec{v}_i + \eta_1 r_1(\vec{p}_i - \vec{x}_i) + \eta_2 r_2(\vec{p}_g - \vec{x}_i) \tag{2.1}$$

$$\vec{x}_i' = \vec{x}_i + \vec{v}_i' \tag{2.2}$$

where $\vec{x}_i'$ and $\vec{x}_i$ represent the current and previous positions of particle i, $\vec{v}_i$ and $\vec{v}_i'$ are the previous and current velocity of particle i, $\vec{p}_i$ and $\vec{p}_g$ are the best-so-far position of particle i and the best position found in the whole swarm so far, respectively. $\omega \in (0, 1]$ is an inertia weight which determines how much the previous velocity is preserved, η_1 and η_2 are acceleration constants, and r_1 and r_2 are random numbers generated from the interval [0.0, 1.0]. The basic structure of the original PSO algorithm is shown in Algorithm 4.

There are some advantages of PSO for solving optimization problems: 1) it is easy to explain; 2) it is simple to implement; 3) it has a fast convergence speed; 4) it is robust to solve different problems by tuning parameters. However, on the other hand, there are some disadvantages of using PSO techniques: 1) It is easy to be trapped into local optima due to the fast convergence speed; 2) there is no global level information sharing approach; 3) there are relatively fewer application areas for PSO than for other EAs.

Algorithm 4 Basic PSO Optimizer

```
1: Initialize a population of particles by randomly generating the position and
   velocity for each particle;
2: Evaluate the fitness of each particle;
3: repeat
4:     for $i := 0$ to $popsize$ do
5:         update particle $i$ according to Eqs. (2.1) and (2.2);
6:         if $f(\vec{x}_i) < f(\vec{p}_i)$ then
7:             $\vec{p}_i := \vec{x}_i$;
8:             if $f(\vec{x}_i) < f(\vec{p}_g)$ then
9:                 $\vec{p}_g := \vec{x}_i$;
10:            end if
11:        end if
12:    end for
13: until A termination criterion is satisfied (e.g., the maximum number of genera-
    tions is reached)
```

2.4 GAs: Key Components

In Section 2.3, we presented a brief overview of GAs. Now, it is necessary to fully understand and examine GAs and their key components in detail. GAs are population-based meta-heuristic approaches that maintain and evolve a population of solutions based on Darwin's theory of survival of the fittest [39]. The flowchart of traditional GAs is shown in Figure 2.4. GAs start with a population of randomly generated candidate solutions. These solutions are encoded with a fixed length representation and different encoding schemes are available in the literature. The solutions are evaluated by an objective function, which produces the fitness value of each solution. Then, the selection approach is applied to identify the parents based on the fitness values to reproduce offspring. The selected parents are paired together to exchange genes via crossover, followed by mutation to create new offspring. Once all of these steps are completed, the GA enters the next generation. This process is continued generation by generation until a certain termination condition is satisfied.

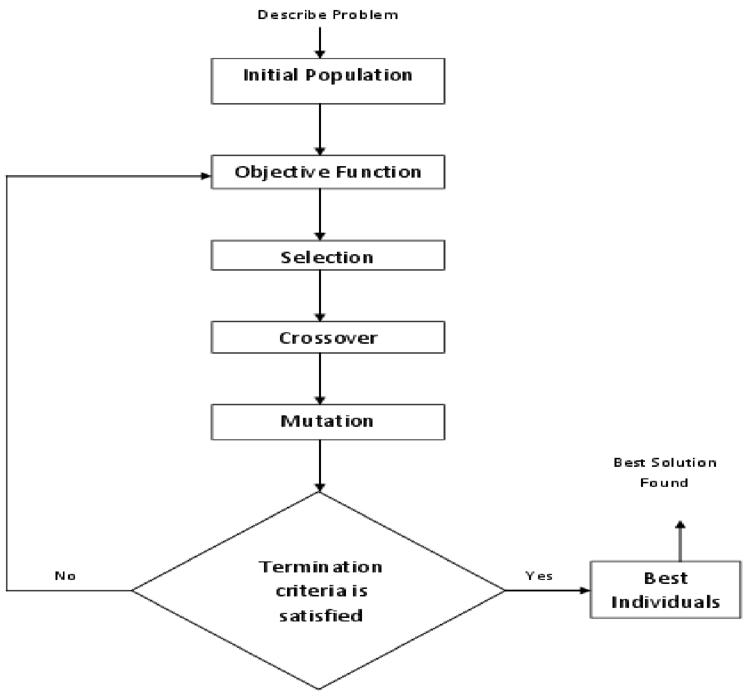


Figure 2.4: Flowchart of a conventional genetic algorithm

2.4.1 Representation of Solutions

When applying GAs to solve a given problem, we first need the determine how to represent a solution to the problem in hand and the representation scheme determines the search space, which should contain all possible solutions for the given problem. Genome is another similar name of the search space and the elements are called genotypes. In nature, genotypes include the whole hereditary information of an organism encoded in the DNA. DNA is the string of base pairs of genetic information which determines phenotypical properties of the creature it belongs to. Each point of genomes in GAs represents a string or linear sequence of certain data type [111]. These genotypes are also called solutions due to the properties of linear structure. Generally speaking, these solutions are used in GAs, which are strings of same data type, for example, bits or real numbers. Different encoding schemes are available in the literature. Here, we explain two most important and widely applied representation schemes and summarize a few of their main properties. In this thesis, it is very hard to re-examine all representation which have been used. A detailed description of different types of representations can be found in [10].

2.4.1.1 Binary Representation

In GAs, one of the most common ways of encoding a solution is based on binary representation. In this case, the search space in GAs can be denoted by $S_g = \{0,1\}^l$, where l is the fixed length of a binary string $x_g = (x_1^g, x_2^g, ..., x_l^g) \in \{0,1\}^l$. Sometimes, the binary representation of solutions are complemented with the application of gray coding during the genotype-phenotype mapping. Gray coding is used to increase the causality and ensure that small changes in the genotype will also lead to small changes in the phenotype [18].

2.4.1.2 Real-Valued Representation

This representation scheme uses real-valued solutions. The search space S_g is denoted as $S_g = \mathbb{R}^l$, where l represents the length of real-valued chromosome. Many different real-world problems can easily be represented by real-valued solutions and corresponding genotype-phenotype mappings.

2.4.2 Objective Function

In GAs, the fitness function is usually the same as the objective function, but is problem dependent. Each individual is evaluated by an objective function, which not only determines the quality of the solution but also corresponds to how much the individual is close to the optimal solution. The fitness of a solution is usually the value of the objective function for that solution.

2.4.3 Selection Schemes

The selection approach is the process to select the individuals from the population probabilistically according to their relative fitness. These individuals are chosen for reproduction, where a number of offspring are produced by their parents. The question is how to select the individuals from the population. There are different number of selection approaches available in the literature to determine the individuals from the population, for example, the roulette wheel selection, rank selection, tournament selection, and a few others [41, 85].

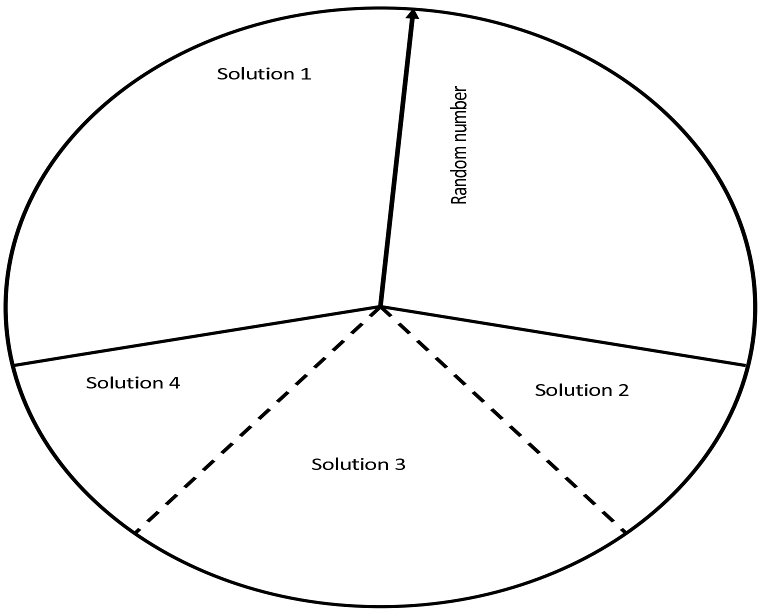


FIGURE 2.5: Roulette wheel selection.

The roulette wheel selection is the simple selection scheme typically used in conventional GAs. In this scheme, a commonly applied reproductive operator is the fitness proportionate selection operator. Here, an individual is chosen from the current population according to a probability proportional to its relative fitness. Figure 2.5 shows a population of four individuals. Each individual is assigned a probability of being selected by using the fitness of that individual, divided by the sum of the fitness of individuals in the population. For example, the probability of the first individual to the fourth individual is 0.1, 0.1, 0.2, and 0.6, respectively. These candidate solutions are diagrammatically shown in Figure 2.5 with their associated probabilities. From Figure 2.5, it can be analysed that a better solution has a higher chance to be selected. The roulette wheel is spun every time when a parent is needed. In each spin, an individual is selected as a parent if the wheel's marker falls in the area represented by that individual.

There are some disadvantages of using the roulette wheel selection approach. In [41, 85], some problems regarding the roulette wheel selection were discussed. When there are large differences between the fitness values among the individuals, then the individuals with a higher fitness will have very large chances to be selected. For example, if the fitness of a candidate solution is 90% of the sum of the fitness of all individuals, then the rest of the individuals will have only 10% of chance to be selected. In this case, the premature convergence (i.e., whole population of EAs enters into the status where all solutions becomes similar due to the loss of diversity. When this happens, it is very difficult for an algorithm to make any further progress in the searching of better solutions) problem may occur since the selection causes the search to narrow down too quickly, and due to the fast convergence, the algorithm may get stuck into local optimum quickly.

2.4.3.1 Rank Selection

Baker suggested the idea of rank selection for GAs. This scheme helps minimizing the problems in the roulette wheel selection. In rank selection, individuals are sorted according to their measured fitness values, and they are ranked by their fitness as follows: the first individual is considered as the worst and the last individual as the best. Then, each individual is assigned a probability for selection according to their rank. The selection probability of individual i is calculated as follows:

$$P_i = \frac{1}{N}(\eta^- + (\eta^+ - \eta^-)\frac{i-1}{N-1}); \qquad i \in \{1, ..., N\} \tag{2.3}$$

Here $\frac{\eta^-}{N}$ and $\frac{\eta^+}{N}$ represent the probability of the worst and best individuals to be selected. The size of population is constant, both conditions $(\eta^+ = 2 - \eta^-)$ and $\eta^- \geq 0$ must be satisfied. Here the important point is that every individual has a distinct rank, such that, if all the individuals have the same fitness value, they still use different selection probabilities.

2.4.3.2 Tournament Selection

The tournament selection method randomly chooses t individuals, where t represents the tournament size, from the population and keep the best solution for further evolutionary processing, and remain repeated until the mating pool is filled. In common practice, the tournament technique is applied only on two individuals (binary tournament) but larger tournament sizes can also be used for picking candidate solutions. A value of $t > 2$ can be used in order to increase the selection pressure of the tournament selection.

2.4.4 Crossover Operators

Generally speaking, crossover is considered as a basic genetic operator in GAs, which exchanges genetic information between two parents to generate the offspring without altering the gene values. It is different from mutation because it does not create new genes but combines the elements of individuals to form new solutions. Combining

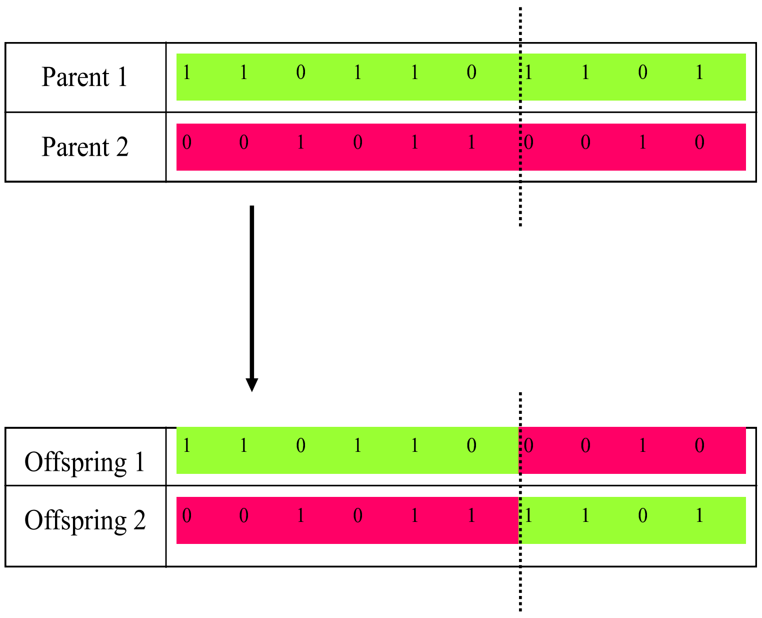


FIGURE 2.6: One-point crossover

the genetic information can be accomplished by several different crossover schemes; some of them are explained below.

2.4.4.1 One-Point Crossover

The simplest crossover operator is the one-point crossover, where two mating individuals are cut once at a specific point and gene values are exchanged after the cut point. One-point crossover is illustrated by Figure 2.6. Here, a point is randomly selected on the mating parents and bits after that point are exchanged between parents. It can be observed that the gene values next to the crossover point are combined to generate two new offspring.

2.4.4.2 Two-Point Crossover

After one-point crossover operator, several crossover operators have been proposed, where two or more cut points are applied. The advantage of using more than one cut

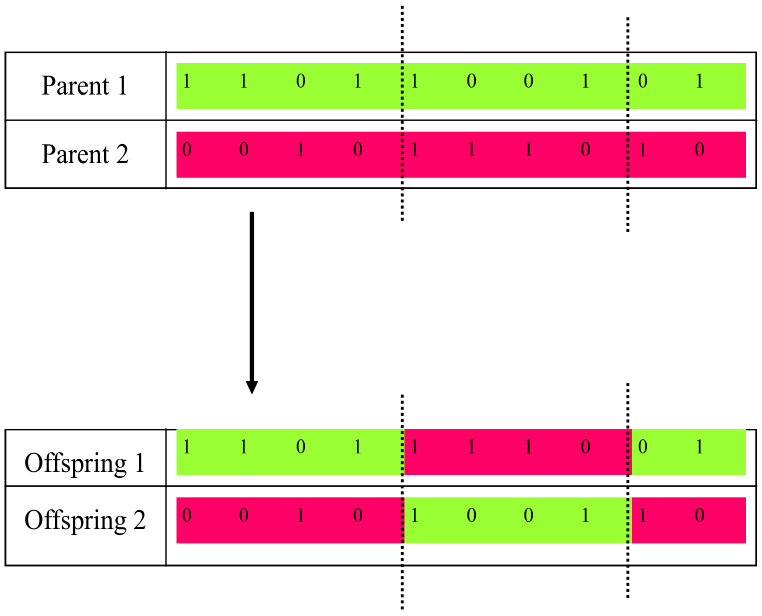
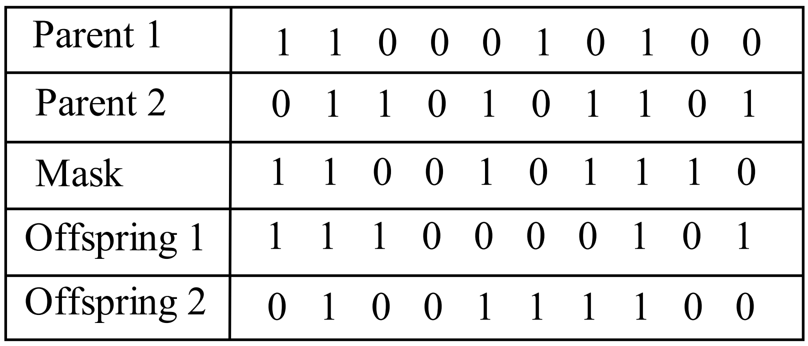


FIGURE 2.7: Two-point crossover.

point is that the problem space can be searched more properly. Two-point crossover randomly selects two points on the parents and combines segments between the two mated individuals, as illustrated in Figure 2.7, where the dotted lines represent the crossover points. Hence, the genetic material between these points are exchanged between the individuals to generate new offspring for the next population.

2.4.4.3 Uniform Crossover

Uniform crossover is a different approach from the multi-point (N-point) crossover. Each bit in the offspring is generated via swapping the associated bit from both parents based on a randomly created binary crossover mask of the same length as the parents. If the bit value is 1 in the crossover mask, then the gene from the first parent is copied into the bit for the offspring; otherwise, the gene from the second parent is copied to create the first child. For the second child, if the bit value is 1 in the crossover mask, then the gene from the second parent is copied into the bit for the offspring; otherwise, the gene from the first parent is copied. For each pair



Parent 1	1	1	0	0	0	1	0	1	0	0
Parent 2	0	1	1	0	1	0	1	1	0	1
Mask	1	1	0	0	1	0	1	1	1	0
Offspring 1	1	1	1	0	0	0	0	1	0	1
Offspring 2	0	1	0	0	1	1	1	1	0	0

FIGURE 2.8: Uniform crossover.

of parents in the population, a new crossover mask is randomly generated. Hence, the offspring created by the uniform crossover include a mixture of genes from each parent. Figure 2.8 shows a uniform crossover operation.

2.4.5 Mutation

Mutation has been conventionally considered as a background operator, which randomly modifies gene values during the evolutionary process. We quote Goldberg's remarks regarding mutation from his book [39] as follows.

> "By itself mutation is a random walk through the search space. When used sparingly with reproduction and crossover, it is an insurance policy against premature loss of important notions."

Mutation plays an important role in recovering lost genetic materials as well as for randomly disturbing genetic information. It works as an insurance policy against the permanent loss of genetic material. Mutation helps jump out from local optima and maintain the diversity in the population. Many different types of mutation operators for different kind of representations were developed in the literature [85] pp 56-57. Bit flipping, interchanging, and reversing mutation operators are some examples under binary representation. These mutation operators will be explained in Chapter 4 later.

2.4.5.1 Fixed Mutation Probability

The mutation probability (p_m) is an important parameter in the mutation technique. Solution will be mutated based on p_m. If new children are created instantly after crossover without any change, there is no concept of mutation. If mutation is used, usually, some genetic information is changed. If the mutation probability is 1.0, the entire individual will be altered. On the other hand, If it is 0, similar individuals will be generated.

Originally, GAs have used a fixed mutation probability to mutate the gene values of individuals. A lot of work has been done regarding the fixed mutation probability, aiming to find the optimal mutation probability for GAs. Different settings of a constant mutation probability were suggested by different researchers [8, 42, 54, 65–68]. However, if not impossible, it is very difficult to find the optimal setting of the mutation probability. There is one problem of using a fixed mutation probability. For example, a very high mutation probability converts a GA into a random search procedure and a too low rate may cause the process to be trapped at local optima. Adaptation process can tackle this problem in GAs [9].

2.4.5.2 Adaptive Mutation

Nowadays, adaptive mutation operators are commonly used in the GA for solving optimization problems, especially for real-world problems. For adaptive mutation operators, the mutation probability is modified during the execution of an algorithm. There are three adaptive mutation schemes, i.e., the variable mutation where the mutation rate is changing based on the progress of the search, the directed mutation where the mutation probability at a particular locus is linked to some properties of that position, and the hybrid mutation which is the combination of both variable and directed mutation schemes. These three adaptive mutation schemes are briefly explained in the following sub-sections.

Variable Mutation

It can be considered as an approach that prevents the premature convergence of GAs to a local optimum. The diversity of the population may not be constant throughout the process. At certain stages of the evolutionary process, different mutation rates can be applied to achieve the best performance. The variable mutation determines that in for a GAs to be effective, a higher mutation rate is necessary when the population has converged this is another advantage of using variable mutation over the fixed mutation probability. In other words, crossover operator is more responsible for search during the early stage of an evolutionary process of GAs and that's why the mutation rate is set at a low value to allow minimal disruption. As process continues, crossover operator becomes less successful and so mutation rate increases. Due to this, variable mutation can be effective during the whole process of GAs to enhance the performance.

Directed Mutation

Directed mutation does not modify the probabilistic nature but using the feed-back information from the previous generation. This approach will deterministically generate new solutions in the search space, in certain environment. The earlier individuals guide new solutions toward the optimum. Hence, it is called directed mutation.

Hybrid Mutation

Hybrid mutation is to integrate variable and directed mutation approaches at different levels of the evolutionary process. This mutation scheme may have the ability to escape from local optima and explore solutions in different regions in the search space.

2.5 Computational Complexity

In theoretical computer science, computational complexity aims to classify the computational problems according to their built-in difficulty.

A computational problem is typically solved by a computer, and therefore, it can be written as a set of mathematical instructions. Typically, a computational problem has an input and an associated solution for that input. Note that there are infinite inputs possible, and there exists a solution for every input. In short, a computational problem is an abstract problem that is typically solved by a computer. For example, in the subset-sum problem, given a set of integers I (e.g., $I = \{2, -3, 4, -1, 5\}$), the question is that "is there a subset of integers having a sum equal to zero". The answer to this question may be "yes" or "no" according to the input. In the above example, it is "yes" as for $\{-3, 4, -1\}$, the sum is zero.

A problem is regarded as intrinsically difficult if it requires considerable resources, no matter which algorithm is used. Computational complexity formalizes the computational needs of a problem by mathematically modelling the time and space required to solve the problem.

In order to measure how difficult it is to solve a problem, it may be appropriate to find how much time (or space) an optimal algorithm will require to solve the problem. However, the resources required (time or space) to solve a problem may depend on the input size. It is, therefore, reasonable to model the resources required (time or space) to solve a problem, as a function of input size.

It is also understood that problems of the same size might have different computational requirements. For example, sorting an array of size n using the bubble sort algorithm will require n operations if the array is already sorted, and n^2 operations if it is in the reverse order. From the complexity analysis viewpoint, the worst case scenario is considered in such circumstances. To suggest approximate limits on the resources required to solve a problem, it is common to model the resources required using the Big-O family of notations [111].

Typically, before identifying the complexity of a problem, a computation model must be considered for that problem. In [13], the authors suggested well-known models of computation, for instance, Turing machines, random access machines, circuits, and probabilistic models. The complexity of a specific problem may vary, depending on which model is chosen for the associated problem. Problems can be further classified in terms of the computational complexity into, e.g., **P** and **NP** classes.

2.5.1 Definition of the P Class

An optimization problem belongs to the class **P** if it can be solved within a polynomial time with a deterministic Turing machine, or the running time of the problem on a deterministic Turing machine is polynomial in length of its input instance. The 2SAT, minimum spanning tree, unary knapsack problem, linear programming, and some other problems, are examples of the **P** class.

2.5.2 Definition of the NP Class

An optimization problem belongs to the class **NP** if the running time of the problem on a non-deterministic Turing machine is polynomial in length of its input. For example, the travelling salesman problem, multi-dimensional knapsack problem, 3-SAT, and many more combinatorial problems, are the hardest problems in **NP**.

According to the above definition of the complexity classes, it is clearly determined that $\mathbf{P} \subseteq \mathbf{NP}$. Whether **P** and **NP** are equal or not, is still a research question so far. For the moment, there is no any algorithm, which is able to solve an NP-complete problem in a polynomial time on a deterministic computer. Initially, computational complexity was just introduced for decision problems and later on for optimization problems. Decision problems take a decision into two possible solutions: Yes or No. Global optimization problems have a similarity to decision problems but these problems are more difficult than decision problems. Meta-heuristics and heuristics are applied to find the near-optimal or global optimum of candidate solutions of **NP** problems in reasonable time. In this thesis, we uses GAs to solve these problems.

2.6 Statistical Test

T-test is a statistical test that is used to find out if there is a real difference between the means (averages) of two different groups. It is sometimes used to see if there is a significant difference in response to treatment between groups in a trial. The T-test is probably the most widely applied statistical data analysis procedure for hypothesis testing. Generally speaking, there are different types of t-test. The general term used for t-test is “two-sample t-test”. It is said to be the “Student’s t-test” or the “independent sample t-test”. For the detailed procedure of using t-test, the readers are referred to [1–3].

The two-sample t-test simply measures whether the means of two independent populations are statistically different from each other. For example, suppose the research hypothesis in our mind is that rich people have a different quality of life than poor people. We conduct the survey regarding the quality of life from a random sample of rich people and a random sample of poor people. A "null" hypothesis assumes that there is no difference between the quality of life in rich and poor people; and this would be assumed true until proved wrong. The t-test is used to determine a *p*-value that indicates how likely we could have gotten these results by chance. By convention, if there is less than 5% chance of getting the observed differences by chance, we reject the null hypothesis and say we found a statistically significant difference between the two groups.

2.6.1 One-sample t-test

One-sample t-test is used to test the mean score of a sample to a known value. Generally, the known value becomes a population mean. For example, the teacher of a school claims that an average student of his school studies 8 hours per day during weekends, and we have to test the truth of this claim. In this context, the "null" hypothesis would be that average students of the school do not study 8 hours per day during weekends. We take a group of students of the school, say 10, and acquire the data on how long they study during weekends. Suppose that the sample mean of the data is 6.5 hours. On the basis of this, we can not infer any thing directly whether the claim is to be accept or reject. In this case, we use one-sample t-test to accept or reject the null hypothesis.

2.6.2 Paired t-test

The paired t-test is used in the situation when data is taken from the same sample before and after the occurrence of some events. For example, paired t-test can be used to determine the significance of the performance of students in some test before and after the tutoring session. A “null” hypothesis assumes that there is no difference in the performance of the student before and after the tutoring session. Improperly applied independent tests in this case would not be able to reject the null hypothesis.

2.7 Chapter Summary

This chapter has presented the basic concepts of optimization, global optimization, and classification of global optimization algorithms. The chapter has described the detailed operation of traditional GAs. GAs work on a population of individuals. Selection, crossover, and mutation operators are utilized to the individuals of the current population to generate new offspring for the next population. Hence, different genetic operators play important roles in GAs. Computational complexity and statistical test, which are relevant to the research of global optimization and EAs, are also briefly discussed in this chapter.

Chapter 3

Adaptation in Evolutionary Algorithms

Adaptation of genetic operators and relevant parameters has become a promising area of research in evolutionary algorithms (EAs) during the last 20 years. It is considered as a state-of-the-art approach to improve the performance of EAs for optimization problems. In this chapter, we review the history of adaptation in EAs, provide a comprehensive classification of adaptation in genetic operators, and pay particular attention to adaptive mutation operators.

3.1 Introduction

Originally, the word adaptation was taken from biology. Adaptation is a basic phenomenon in biology. It is the evolutionary process whereby a population becomes better and better accommodated to its habitat. This process takes place over many generations.

At the beginning of evolutionary computation, EAs were considered robust with respect to relevant parameters. However, it has been observed that the right parameter values do greatly affect the performance of EAs in a specific problem. Hence, in the last two decades, setting proper values for parameters has been considered cruicial for the good performance of EAs. Some key parameters in EAs are the population size, representation, selection pressure, crossover probability, and mutation probability. These parameters are usually called algorithm parameters or strategy parameters. The challenge is how to set the right values for these strategy parameters in order to achieve the best performance of EAs for a given problem.

Several researchers have proposed different constant values for key parameters in EAs in order to find good solutions for a particular fitness landscape [8, 42, 54, 65–68]. These parameter settings are derived from experience or by trial-and-error, and are fixed before the execution of algorithms. However, it is very difficult, if not impossible, to find suitable parameter setting for the optimal performance of EAs, and the approach of finding proper parameters is also time consuming. Morevoer, static values are enventually discouraged by the EA researchers because different values of parameters and different operators may be suitable at different stages of the evolutionary process of an EA. So, no common optimal parameter setting can be found initially [115]. In order to address this deficiency, researchers have diverted their attention towards adapting the parameters during the evolutionary process for finding better solutions to the problem in hand.

It is a natural idea to adjust genetic operators and relevant parameters during the execution of an algorithm. In the early days, the adaptation of crossover and mutation operators was used to improve the performance of genetic algorithms (GAs) [25, 32, 77, 96, 114]. These schemes aim to maintain sufficient diversity in the population to prevent GAs from being trapped into local optima and determine the direction and/or magnitude of changing the parameters according to the feedback information from the search space. More recently, many researchers have applied different methods to integrate adaptation into different EAs [30, 61, 101, 105–108, 110, 120, 124]. These approaches are used to solve various optimization problems. In this chapter, we focus our attention on variation operators for EAs in general and on the mutation operators for EAs in particular.

The rest of this chapter is arranged as follows. Section 3.2 briefly reviews the history of adaptation in EAs. The comprehensive classification of adaptation techniques is presented in Section 3.3. Section 3.4 describes the generation consideration for adaptation in EAs. An adaptation of the population size is presented in section 3.5. Section 3.6 describes an adaptation of representation. An adaptation of selection operators are presented in section 3.7. Section 3.8 describes an adaptation of variation operators. Finally, the summary is given in Section 3.9.

3.2 Short History of Adaptation in EAs

A brief overview of the historical development in adaptation approaches for EAs is presented in this section. Since the beginning of evolutionary computation, modifying the values of various parameters in EAs has been a research topic.

In 1967, Rosenberg [74] suggested an approach which modifies the crossover probability during the run time in the context of evolutionary strategies (ESs). The techniques for adapting the parameters during the evolutionary process were proposed in the early 1970s by Rechenberg [72]. The 1/5 success rule is a well-known example of parameter adaptation. It records the ratio of successful mutation to unsuccessful mutation within a certain number of generations. If the ratio is more than 1/5, then the mutation strength should be increased; otherwise, it should be decreased.

In [25, 26], Davis proposed an effective algorithm for updating the probabilities of the operators according to their performance. This scheme specifies that the modification of operator rates is associated with the fitness of solutions generated by the operators; this has been used for steady state GAs. Julstrom [55] proposed a slightly different idea; in his approach, an operator was chosen if the generated child is better than its own parent. This should be done on the basis of calculation of the reward value to the parents of the good child, which is easier than Davis's approach.

In [32], Fogarty proposed a dynamic mutation rate control scheme for GAs, where the mutation rate decreases exponentially over the number of generations. Whitley *et al.* [114] suggested an adaptive mutation technique, which has been shown to bring in significant performance improvement. The probability of mutation is adapted according to the Hamming distance between the parent individuals. The diversity is maintained in the population by associating the similar solutions to increase the mutation rate. It has been introduced for the steady state GA. Another adaptive scheme was proposed by Srinivas [96], which is almost similar to the adaptive scheme by Whitley [114]. In this approach, the procedure of varying the probabilities of crossover and mutation takes into account the fitness values of solutions. This technique also maintains the diversity of the population without influencing the convergence property, which has been applied for the generational GA.

In [51], Hong *et al.* proposed a dynamic mutation scheme for GAs, which simultaneously applies several mutation operators in generating the next population. All the mutation operators have the same initial ratio and each mutation operator is used according to its assigned selection ratios. The selection ratio of each mutation operator is updated based on its average progress value at every generation.

A directed variation (DV) technique was proposed by Zhou and Li [125]. This algorithm does not introduce any scheme for completely generating an average step size but adjusts some individuals by using the feedback information from the current population. Suppose the population is a set of N individuals $X = \{\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_N\}$ and each individual is a K-dimensional vector, denoted by $\vec{x}_i = [x_{i1}, x_{i2}, \cdots, x_{iK}]$. We denote the minimal d-th component of the individuals at generation t by x_d^L and the maximum by x_d^U, that is, the range of the d-th dimension at time t is $R_d(t) = [x_d^L, x_d^U]$. This range can be equally divided into L intervals. The fitness of a nonempty interval, say, the j-th interval of the d-th dimension, is defined as:

$$F_{dj} = \sum_{i=1}^{N} I(x_{id} \in B_{dj}) f_{Norm}(\vec{x}_i) \tag{3.1}$$

$$I(x_{id} \in B_{dj}) = \begin{cases} 1, & \text{if } x_{id} \in B_{dj} \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

where B_{dj} denotes the range (lower and upper bounds) of the j-th interval of the d-th dimension, N represents the population size, $I(.)$ is the indicator function, and the fitness of each solution vector $\vec{x}_i$ is normalized as follows:

$$f_{Norm}(\vec{x}_i) = \frac{f(\vec{x}_i) - f_{min}}{f_{max} - f_{min}} \tag{3.3}$$

where f_{max} and f_{min} represent the maximum and minimum fitness of the whole population, respectively.

Within DV, in each generation, some individuals are selected for directed variation in each component. DV is applied on a component, say, the d-th component, only when the range of the d-th dimension of all solutions in the current generation decreases in comparison with that of the previous generation, i.e., the population converges regarding that dimension. DV works as follows. First, the fitness of interval, i.e.,

F_{dj}, is calculated according to Eq. (3.1). Then, DV is applied for an individual component by component, where each component of the individual may be shifted from its current interval to a neighboring interval that has a higher fitness with a certain probability, as described below.

In DV, for each component $x_{id} \in B_{dj}$ of an individual $\vec{x}_i$, whether it is mutated or not depends on the value F_{dj} and the fitness of its neighboring intervals, i.e., $F_{d,j-1}$ and $F_{d,j+1}$. If F_{dj} is greater than both $F_{d,j-1}$ and $F_{d,j+1}$, then DV is not applied to the d-th component of any selected individuals with $x_{id} \in B_{dj}$. If F_{dj} is in the middle, without loss of generality, suppose $F_{d,j-1} > F_{dj} > F_{d,j+1}$, the probability of directed variation, P_{dj}^{DV}, can be calculated as follows:

$$P_{dj}^{DV} = 1 - \frac{F_{dj}}{F_{d,j-1}} \tag{3.4}$$

With this probability, x_{id} is replaced with a number, randomly generated between x_{id} and the center of $B_{d,j-1}$. If F_{dj} is smaller than both $F_{d,j-1}$ and $F_{d,j+1}$, then either $B_{d,j-1}$ or $B_{d,j+1}$ is randomly selected with an equal probability and x_{id} moves towards the selected interval, i.e., replaced with a number randomly generated between x_{id} and the center of the selected interval.

A statistics-based adaptive non-uniform mutation (SANUM) operator was introduced for GAs by Yang [118]. The basic idea of SANUM is to make use of feedback information implicitly contained in the population to explicitly direct the mutation operation. Uyar *et al.* proposed an asymmetric gene-based adaptive mutation (GBAM) technique [84]. In GBAM, each gene locus has two different mutation probabilities: pm^1 which is used for the loci having the value 1 and pm^0 which is used for the loci having the value 0. The probabilities of pm^1 and pm^0 are automatically updated based on the feedback information from the search space, according to the relative success or failure of those chromosomes having a 1 or 0 at that locus for each generation. Yang and Uyar [120] suggested another gene-based adaptive mutation with fitness and allele distribution correlation (GBAM_FAD), where the mutation rate of each gene locus was adaptively modified based on the correlated feedback information from the search process, according to the relative success or failure of the solutions.

Recently, an adaptive operator selection scheme was proposed by Thierens [101, 102], who suggested an adaptive pursuit strategy for allocating operator probabilities. It pursues the optimal operator that currently has the maximal estimated reward. In order to do this, the algorithm increases the selection probability of the optimal operator and decreases the probabilities of all other operators. The pursuit algorithm was inspired from the field of learning automata, which are rapidly convergent algorithms for the learning automata introduced by Thathachar and Sastry [100]. In [24], the authors suggested an adaptive operator selection approach based on the well-known Multi-Armed Bandit (MAB) paradigm in order to combine the MAB concept with the statistical Page-Hinkley test, which has been applied in dynamic environments and is efficient in terms of detecting changes in time series.

3.3 Taxonomy of Adaptation in EAs

Several researchers have introduced different terminologies on the adaptation of genetic operators and their parameters for EAs [5, 28, 91, 93]. Generally speaking, there are two classification criteria: the type of adaptation (i.e., how a parameter is modified) and the level of adaptation (i.e., where the changes occur). Below, we first introduce the classification proposed by Angeline [5], and then present the classification proposed by Eiben *et al.* [30], which extends and broadens the idea by Angeline [5].

According to Angeline's classification [5], based on the type of adaptation, adaptive EAs can be categorised into two distinct classes of approaches: one with absolute update rules and the other with empirical update rules. For the first category, with absolute update rules, the feedback information over a number of generations or populations is calculated. Then, based on the feedback information, some deterministic rules regarding when or how to alter the strategy parameters are determined. Rechenberg's 1/5 rule [72] is a typical example of this category. For the second category, with empirical update rules, the strategy parameters in EAs are modified via self-adaptation. The parameters to be modified are encoded into chromosomes and further undergo the process of genetic variation. Hence, the parameters should be able to self-adapt themselves. Some researchers have done a lot of work regarding this self-adaptation approach [16, 63, 79, 92].

Later in 2007, Eiben *et al.* [30] introduced a taxonomy of parameter setting approaches for EAs, as shown in Figure 3.1. According to this taxonomy, parameter setting is divided into two main sub-fields: parameter tuning and parameter control. Parameter tuning (also called static adaptation) means to set suitable values for parameters before the run of EAs. These parameters will remain constant during the execution of EAs. Static adaptation is done by an external process (e.g., a person or a program) for selecting appropriate values. Many researchers have suggested well-known heuristics to set different parameters in order for EAs to find good solutions for a problem [39, 42, 54, 66–68, 76]. These values are derived from experiences or via trial-and-error methods. In addition, it has been empirically and theoretically shown that different operators as well as different values of parameters may be optimal at different stages of the evolutionary process [7, 9, 25, 47]. The term of parameter control is applied to categorize algorithms where strategy parameters are adjusted using various methods during the execution of EAs. There are three different distinct sub-classes, which are determined according to the adaptation

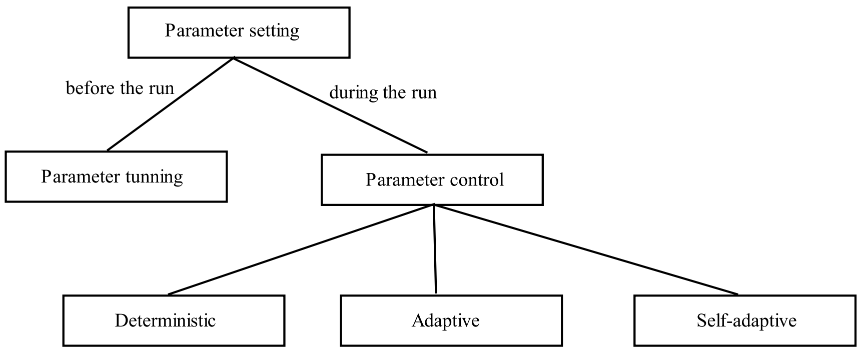


Figure 3.1: Global taxonomy of parameter setting in EAs.

mechanism. They are deterministic, adaptive, and self-adaptive mechanisms.

Deterministic adaptation adjusts the values of parameters according to some deterministic rule without using any feedback information from the search space. Examples of this approach, like the time-varying modification of the probability of mutation, are shown in [50]. The adaptive approach modifies the algorithm parameters using the feedback information from the search space. This information is used as an input in the adaptive approach to determine the direction or magnitude of changes for the algorithm parameters. Several examples of this type of adaptive adaptation methods are available in the literature [25, 30, 32, 55, 72, 114].

In the case of self-adaptive adaptation, the parameters are altered by EAs themsleves. Strategy parameters to be modifed are encoded into chromosomes and undergo variation with the rest of the chromosome. The basic concept of self-adaptation is better explained in [8].

According to [5, 30], both classification approaches have suggested three distinct levels at which adaptation can take place in adaptive EAs. These levels of adaptation can be applied with each type of adaptation. The first one is the population level adaptation, in which strategy parameters are modified globally for the whole population and are applied on all individuals of the population. In general, this population level of adaptation is used with the deterministic or adaptive approach. Many researchers have proposed various well-known and popular parameter control schemes based on the population level adaptation [23, 25, 55, 84, 118, 120]. The second level is the individual level of adaptation, where the strategy parameters are modified for each individual of the population independently. There are also various well-known approaches suggested to alter the algorithm parameters regarding the individual level adaptation [8, 48, 77, 90, 93, 96]. Finally, the third level of adaptation is the component-level, where strategy parameters are changed for some gene or component of an individual in the population. Some examples of this level of adaptation have been introduced in the literature [8, 84, 87–89, 118, 120].

3.4 General Considerations for Adaptation in EAs

In order to apply the adaptation approach in EAs, there are several key aspects to be studied, which are described as follows:

1. What components/parameters are changed?

   The parameter control techniques are categorized in the perspective of what components or parameters are changed. It is necessary to agree upon the set of all main components of an EA; which is a hard task itself. In this context, each of the following components can be parameterised in EAs.

   - population (size, topology)
   - Represenation of solutions
   - Genetic operators and their probabilities
   - Selection scheme and some others.

2. How is the change made?

   Algorithm parameters are modified on the basis of various methods during the execution of EAs. The value of algorithm parameters can be classified into one of the three categories: deterministic, adaptive, and self-adaptive. These three types are taken from the parameter control. These have been briefly explained in the section 3.3.

3. Which evidence is used to make the change?

   This category of parameter control classification concerns with the evidence used for the alteration of parameter values. In general, the progress of the search is observed, e.g., by examining the performance of operators, the diversity of the population, and so on. The parameters are adjusted by using the feedback information, such information is gathered by monitoring the process. The evidence is further classified into two sub-fields: absolute evidence and relative evidence. The detailed description of these sub-fields was given in [30].

4. What is the scope/level of the adaptation?

   Any component of an EA can be changed, which may affect a gene, a whole solution, the entire population, and other component (e.g., the evolution function). This is called the scope or level of adaptation. However, the scope or level is not an independent factor. It always depends upon the component of the EAs where the modification may occur. The best example of this approach is the change of the mutation step size. It may affect a gene, an individual, or the entire population, depending on the specific implementation.

3.5 Adaptation of the Population Size

In traditional GAs, the population size is usually set to a constant value during the execution of GAs. The population size is an important parameter in GAs. Researchers in the GA community have been studying the effect of the population size on the performance of GAs since the early days of GAs [40]. If the population size is too small, there is a potential problem that the population may get stuck on local optima (or GAs may not be able to find new candidate solutions). In this context, the diversity will be lost among the solutions and then the algorithm can hardly make any progress. However, if the population size is too large, then the computational cost of GAs may become very high. The standard setting (50-100 individuals) of the population size is altered by experimenting with a number of different sizes and selecting the right choice that gives the best performance on different problems in hand. It is a very difficult task, if not impossible, to find the suitable population size for the optimal performance. In addition, several researchers have investigated that different population sizes may be optimal at different stages of a single run of an EA.

3.5.1 Adaptive Population Sizing Approaches

There are several schemes for adjusting the population size during the run of GAs. Lobo and Lima [61] presented a review of adaptive population sizing schemes in GAs. Some of them are presented here in this thesis. Interested researchers can find more adaptive populaltion sizing schemes in [61]. They raised three main motivations for developing these techniques:

1. The recognition that an adequate sizing of the population is problem dependent and is hard to estimate.

2. The observation that a GA with an adaptive population sizing scheme may find a better solution quality and better performance than a GA with fixed (and poorly set) population size.

3. To make life easier for users by removing the population sizing parameter.

3.5.1.1 Population Sizing in GAVaPS

The GA with Varying Population Size (GAVaPS) was introduced in [6], which is based on the concept of age and lifetime of an individual. When a solution is generated, either during the initial generation or through a genetic operator, its age is set to zero. After that, for each generation that the individual survives, its age is incremented by 1. Every individual is allotted a lifetime by birth, which corresponds to the number of iterations that the solution is allowed to survive in the population. An individual would die and be removed from the population when the age of the individual exceeds its lifetime. A fraction ρ (called the reproduction ratio) of individuals are regenerated for the current population at every generation. Every individual of the population has the same probability of being selected for reproduction. Hence, there is no explicit selection operator in GAVaPS. Individuals are chosen indirectly for the selection process through their own lifetime. An individual with above average fitness has a longer lifetime than an individual with below average fitness. Reinforcement of best solutions should result in above-average assignment of their offspring in the auxiliary populations. While each individual have an equal probability to undergo the genetic recombination, the expected number of the solution's offspring is proportional to its lifetime value. Therefore, solutions keeping above-average fitness values should be allowed higher lifetime values.

Three lifetime strategies (proportional, linear, and bi-linear) have been used based on the concept of reinforcing individuals with high fitness and controlling individuals with low fitness. There are two important parameters, i.e., $MinLT$ and $MaxLT$, which denote the minimum and maximum lifetime value given for an individual, respectively. These techniques rely on $MinLT$ and $MaxLT$. The values of $MinLT$ and $MaxLT$ were set to 1 and 7, respectively. The authors did not clearly mention how these parameters should be set. Interested researchers can find detailed description of this mechanism and slightly different adaptive population size approaches in the paper [61].

3.5.1.2 Strategy Adaptation by Competing Sub-Populations

This approach is based on the adaptation of the population size by using competing sub-populations [78], which is inspired by the biological discovery that the species increase (or decrease) in size, when these species are competing for the same food.

In this approach, there are a set S of competing populations, each of which runs a different search mechanism independently. It is also possible for populations to compete with each other at regular intervals. The concept is to increase the size of the best acting population and decrease the size of the rest of the populations.

Quality criterion and gain criterion are two important parameters in this scheme. The first one is based on the fitness of the best individual of the competing population, which marks the performance of a population, and the second gives reward or penalty of the competing populations.

An improved version of the same technique was given by the same authors, which assigns a consumption factor γ for each population. The idea was motivated by the recognition that different search strategies might produce the optimal performance at different population sizes. For example, the authors investigated that mutation performs more efficiently with small populations and the recombination operator gives the best result with large populations. In contrast to the basic competition model, the improved version allows the total population size to alter during the whole evolutionary process. The authors reported that the improved version of the competition scheme is usable for locating good regions of attraction with a breadth first search algorithm.

However, a serious disadvantage of the competing scheme is that the sum of population sizes of all strategies remains constant during the whole simulation. This problem has been tackled by the improved version of the competing scheme, but it is not clear how sensitive the algorithm is with respect to the consumption factor γ, the loss factor applied in the gain criterion and the evaluation interval between successive competitions.

3.5.1.3 Population Sizing in SAGA

A Self-Adaptive GA (SAGA) was suggested in [48], which considers an adaptive population sizing technique to specify three parallel GAs, each with its own population. The three populations are denoted as P1, P2, and P3 with $size(P1) < size(P2) < size(P3)$, where $size(P1)$, $size(P2)$, and $size(P3)$ denote the population size of P1, P2, and P3, respectively. The sizes of the three populations are adjusted using some rules in order to “optimize” the performance of the middle-sized population P2. The initial population sizes are set to 50, 100, and 200, respectively. The range of the population size is set to between 10 and 1000. The difference among the population sizes should always be 20 in order to avoid population sizes to be too similar.

The population sizes are updated based on the best fitness found in each GA at regular intervals (an *epoch* in the authors’ terminology) according to a number of rules. Specially, there is a “move-apart” rule which denotes that two populations must be moved apart if the difference of the fitness of their best solutions is within a threshold $\epsilon = 10^{-9}$, by halving the size of the small population and doubling the size of the large population while moving the populations apart. This condition also happens when all three populations are within the ϵ fitness threshold. In this context, without any modification of the middle-sized population, the other two populations either grow or shrink.

The following rules are applied when the best fitness of three populations of solutions are not within the ϵ value:

$$if \quad f(P1) < f(P2) < f(P3) : move\, right$$

$$if \quad f(P3) < f(P2) < f(P1) : move\, left$$

$$if \quad f(P1) < f(P3) < f(P2) \quad or \quad f(P2) < f(P1) < f(P3) : compress\, left$$

$$if \quad f(P2) < f(P3) < f(P1) \quad or \quad f(P3) < f(P1) < f(P2) : compress\, right$$

where f represents the fitness of the best individual, and "move right", "move left", "compress left", and "compress right" are defined as follows:

$$\begin{aligned} move\ \ right : size(P1) &= size(P2);\\ size(P2) &= size(P3);\\ size(P3) &= size(P3) * 2; \end{aligned}$$

$$\begin{aligned} move\ \ left : size(P1) &= size(P2)/2;\\ size(P2) &= size(P1);\\ size(P3) &= size(P2); \end{aligned}$$

$$compress\ \ left : size(P1) = (size(P1) + size(P2))/2;$$

$$compress\ \ right : size(P3) = (size(P2) + size(P3))/2;$$

The main motivation behind these rules is to grow or shrink all the population sizes, if the “ideal” population size seems to be out of the range of the current population sizes; otherwise, we should adapt the size of the worst performing population so that its size becomes closer to the size of the other two populations.

The limitations of SAGA are that there exists a maximum population size and the time span of the epoch parameter is bounded. It is not clear why 1000 fitness evaluations are employed for the time span (epoch).

3.5.1.4 Population Sizing in PRoFIGA

A Population Resizing on Fitness Improvement GA (PRoFIGA) was introduced by Eiben and Valk [29]. This algorithm bears a close resemblance to a traditional GA. In PRoFIGA, however, the population size can grow or shrink according to whether there is an improvement of the best fitness in the population. There are two possibilities to grow the population size: (1) there is an improvement in the best fitness, or (2) there is no improvement in the best fitness for a ”long time”. If both of these conditions do not hold, then the population size shirks by a small percentage (1-5). The authors reported that the motivation of PRoFIGA is to use large population sizes for exploration and small population sizes for exploitation. The growth rate X for the population size is defined as follows:

$$X = F_{increase} \times (maxEvalNum - currEvalNum) \times \frac{maxFitness_{new} - maxFitness_{old}}{maxFitness_{init}}$$

where $F_{increase}$ denotes the increase factor in the interval $(0, 1)$, $maxEvalNum$ is the maximum number of fitness evaluations allowed for the whole evolutionary process, $currEvalNum$ is the current number of evaluations, and $maxFitness_{new}$, $maxFitness_{old}$, and $maxFitness_{init}$ represent the best fitness value in the current generation, previous generation, and initial generation, respectively.

Although the conventional constant population size has been eliminated from PRoFIGA, some parameters need to be assigned by the user before the execution of the algorithm, which are the initial population size, minimum and maximum population size where the algorithm must follow, the increase factor, the ”long time without improvement (V)”, and the decrease factor. In the experiments carried out in [29], PRoFIGA used the initial population size 100, the increase factor 0.1, $V = 500$, the decrease factor 0.4, the minimum population size 15, and the maximum population size 1000. Unfortunately, it is difficult from the user’s point of view to decide how to set the values for these parameters in order to solve a particular problem.

The authors reported that the performance of PRoFIGA was compared with a GA with adaptive population size (APGA) and a parameter-less GA. The performance of these approaches is ordered in the following sequence: APGA, PRoFIGA, and the parameter-less GA. Those claims, however, have been determined to be unjustified [62].

3.6 Adaptation of Representation

Grosan and Oltean [43] developed an Adaptive Representation EA (AREA) for single objective optimization problems, which uses an ever-changing alphabet for encoding individuals. Several examples can be found from nature in order to justify AREA. For example, DNA is an important component of human body, which contains the basic genetic information of living organisms. Actually, DNA is a string of nucleotides over the alphabet {T (thymine), C (cytosine), G (guanine), A (adenine)}. Similarly, simple EAs also use strings over the alphabet {0, 1}, which consists of binary values.

However, if a small size (e.g., 2) is used for nucleotide, the length of the human DNA (in this case, the diversity will be lost) can be very large. The solution can be significantly changed by applying too small mutation on it and the acquired individual can hardly survive in future generations. If a large size (10) is considered for nucleotide, the length of the human DNA (same above problem) could have been very small. In this case, the mutated individual can create a significant change and the diversity of the population increases.

The main idea behind this approach is to encode each solution of the population over different alphabets. In addition, the representation of a particular individual is adaptively changed during the search process as an effect of the mutation operator.

In order to address the representation of solutions, each AREA individual contains two variables (x, Y), where x is a string encoding component variable and Y determines the alphabet applied for encoding x. Y is an integer number, $Y \geq 2$, and x is a string value from the alphabet $\{0, 1, ..., Y - 1\}$. Each solution has its own encoding alphabet. The alphabet of x can be altered during the evolutionary process. An example of an AREA solution is defined as follows: $C = (231054, 6)$. An improved version of AREA was introduced with multi-objective optimization problems in [69].

3.7 Adaptation of Selection Operators

The selection operator is one of the main components in EAs, which selects the individuals from the current population to form a mating pool. Chosen individuals from the population may undergo recombination operations to generate new offspring to make next population.

The selection pressure is the critical parameter in the selection mechanism, which can be used to control the convergence rate of EAs, with a higher selection pressure resulting in a higher convergence rate. However, if the selection pressure is too high, there is a potential problem: the high convergence rate may lead to the lose of diversity of the population, and hence, the EA gets stuck on local optima. In this situation, the algorithm can not make any further progress. However, if the selection pressure is too low, the convergence rate will be slow and it will take the algorithm unnecessarily longer to find the optimal solution. These problems were reported in [64] and it also analyzes the effect of noise on different selection approaches for EAs.

To address the above problem regarding the selection operator, some kind of adaptation of the selection pressure is highly desirable. Two EAs have been recently introduced to incorporate adaptation of selection into them [44, 71], which are described below.

3.7.1 Adaptive Selection Routine for EAs

Pham and Castellani [71] proposed an adaptive selection routine for EAs, which adjusts the stochastic noise level in the determination of the mating pool in order to regulate the selection pressure. This approach is composed of two main levels, namely noise addition and mating.

Let us start from the first level, in which a noise addition procedure is used to create two ordered lists of selected individuals. This procedure is based on the following equation:

$$m(i) = f(i) + (f_{max} - \mu_f) \times rand \tag{3.5}$$

where $m(i)$ denotes the mating chance of individual i, $f(i)$ is the fitness of individual i, f_{max} and μ_f represent the maximum and average fitness of the solutions, respectively, and $rand$ is a random number in the range $(0, 1)$, which serves as the noise factor. These solutions are ordered regarding their mating chance, and the best fitness half is contained into a temporary list A according to its ranking order. Another mating chance is measured for each individual by using the above equation once again. The population is marked in the descending order of the mating chance of individuals, and the top half is moved into a second temporary list B.

The main idea behind this level is to adjust the selection pressure according to the progress of search and to speed up the convergence of the population towards the global optimum. At the early generations, the difference $(f_{max} - \mu_f)$ is large and the noise factor plays an important role in the determination of the mating chance. The major consequence of population convergence is the loss of diversity since the population converges somewhere in the search space and all individuals are similar. As the population converges and the diversity of the population is reduced, the difference $(f_{max} - \mu_f)$ becomes increasingly small. In this context, the stochastic noise factor in the above equation decreases and the search becomes increasingly deterministic and rapid.

In the second level, based on the two lists already generated in the level one, individuals are selected for reproduction in the following way: the first chromosome of list A is combined with the last element of list B, and so on, while both lists are scanned in the opposite direction. The motivation of this level is to gradually improve the average fitness of the population, preventing potentially similar individuals from pairing up, and avoiding the premature convergence.

3.7.2 An Adaptive Tournament Selection

Gwozdz and Szlachcic [44] developed an adaptive tournament selection approach for EAs for the capacitated vehicle routing problem. This technique is used to make the compromise between the potential selective pressure and the wide search range of solutions. The tournament size is updated during the evolutionary process according to the index $p > 0$, where $p \in \{2, 3, ..., p_I\}$. The number of p_i solutions chosen in one tournament may change. The number n of tournament sizes is mostly less than the number of iterations I. After a fixed number of iterations m, where $m = \llcorner I/n \lrcorner$, the tournament size increase according to $\{p_1, p_2, ..., p_n\}$, where $p_i < p_{i+1}$ for $1 \leq i \leq n$.

The motivation of this mechanism is the possibility of controlling the selective pressure during the whole running process of the algorithm. In the initial phase of the running process, a small tournament size provides the possibility of searching in a wide area. On the basis of earliest generations, the search area becomes narrower when we increase the tournament size, which allows us to accelerate the convergence of the adaptive selection EA.

3.8 Adaptation of Variation Operators

Variation operators and relevant parameters greatly affect the performance of EAs. However, selecting the suitable operators and parameters is a difficult task and there are no general guidelines to help determine an optimal configuration for these operators and parameters. In common practice, they are determined by trial-and-error experiments for a particular domain in advance, and then are fixed during the running of EAs. Clearly, this type of constant parameter setting technique is computationally expensive. If the parameters are inappropriately set, then it can lead to sub-optimal performance. The optimal parameter settings might be different with the evolutionary process of EAs. Due to this reason, researchers have focused their attention towards the adaptive methods. They have considered adaptive algorithms as a means of enhancing the performance of EAs [25, 30, 32, 51, 61, 96, 102, 114, 116–118, 120].

3.8.1 Adaptation of Crossover Operators

Since the beginning, the crossover operator is considered as the primary genetic operator in GAs. Different authors have long been suggesting approaches of incorporating adaptation into crossover operators in GAs. Yang [116] suggested three distinct levels where adaptation in crossover operators can occur from top to bottom. These levels are briefly described as follows.

3.8.1.1 Adapting the Type of Crossover

In the top level adaptive approach, Davis [25] proposed that the GA might not use both crossover and mutation for selected parents; operators can be selected with fixed probabilities from a set of operators. In [31], the authors introduced an adaptive technique, where if the population is converged, then the population would be partially or fully randomized except the best solutions; during this re-start process the population switches between two crossover operators based on their performance. In [93], one extra tag bit is added with each individual, which co-evolves with the individual. Crossover operator is selected on the basis of the tag bit. If both the tag bits are 1 for both the selected solutions, then the two-point crossover is applied; if both are 0, then the uniform crossover is applied; otherwise, a crossover operator is randomly chosen between the two-point and uniform crossover and is applied.

3.8.1.2 Adapting the Rate of Crossover

In the medium level, a different Cost Operator Based Rate Adaptation (COBRA) approach was suggested by Corne *et al.* [23] for modifying the rates of applying operators for solving timetabling problems. According to the COBRA approach, the GA statistically swaps assigned i constant probabilities between i operators by assigning the highest probability to the operator which obtains the greatest growth in fitness. In [103], the authors extended the idea introduced in [23], by encoding each individual's crossover and mutation probabilities as real numbers (normalized to 1) that are applied by and co-evolve with the individual.

3.8.1.3 Adapting the Crossover Position or Swapping Rate in Each Locus

In the final bottom level, the position of crossing or exchanging probability of each site is altered during the evolutionary process of GAs. The notion of recombination distribution was introduced by Booker [17], which determines the probability of all achievable recombination events. Distinct operators represent different probability distributions. In [113], White and Oppacher introduced an adaptive uniform crossover scheme, where each site of a solution in the population is increased automatically at each bit position; it describes the relation to a crossover probability for that bit location. A different mechanism was applied in the Statistics-based Adaptive Non-Uniform Crossover (SANUX) method [116], which uses the feedback information of the allele distribution in each locus to adaptively modify the crossover operation.

3.8.2 Adaptation of Mutation Operators

After the consideration of the crossover operator, the mutation operator is another important variation operator in EAs. Holland [50] proposed mutation as a kind of background operator for GAs that randomly alters bits of a solution with an associated small mutation probability per bit. It can also be a key variation operator in GAs to explore the seach space [94]. However, in the mutation operator, it has been investigated that the mutation rate is not only based on problem being optimized but also on the search space and feedback information from the current state of search [9]. A lot of work has been done on adaptation in mutation for GAs [8, 23, 25, 28, 30, 32, 55, 103, 118]. Some learning rules for choosing the operator have been mentioned in the literature [98, 101, 102]. There are two different levels at which adaptation can occur in mutation operators, as suggested by Yang [116].

3.8.2.1 Adapting the Probability of Mutation

Different adaptive mutation approaches which belong to the top level were proposed by researchers [55, 101, 102, 105, 107, 108]. These techniques adjust the probability of mutation during the running of a GA. In [55], Julstrom introduced an adaptive scheme, which adjusts the ratio between mutation and crossover according to their performance. An adaptive GA was proposed in [96], in which the probabilities of mutation and crossover are adapted depending on the fitness values of the individuals. This technique provides two important factors of maintaining diversity in the population and sustaining the convergence capacity of GAs.

A dynamic mutation GA (DMGA) was introduced in [51], which simultaneously uses several mutation operators in solving optimization problems. Each mutation operator is used according to its progress ratio. The mutation operators that result in higher fitness values of children have their progress ratios increased. The mutation operators that result in lower fitness values of children have their progress ratios decreased. Here, the pursuit algorithm was originally taken from the learning automata, which were introduced by [100] and are a class of rapidly converging algorithms. The learning automata represent adaptive allocation rules that adapt the probabilities of selecting operators in order to pursue the selected operator that currently has the maximal estimated rewards. This approach increases the selection probability of that operator and decreases the probabilities of all other operators [102].

In [107], the authors proposed to dynamically adapt the probabilities of genetic operators according to the global behaviour of the population for each generation. There are two main components mentioned in the proposed scheme, which are assigning rewards to operators based on the fitness improvement of the solutions and updating the probabilities of these operators at each generation. Vafaee and Nelson [105] developed a novel approach which adaptively updates mutation rates in GAs. This approach integrates the basic statistical framework of biological evolutionary models into the framework of EAs. Adapting the mutation rate is performed by using these evolutionary models.

3.8.2.2 Adapting the Mutation Rate of Each Locus

The bottom level of mutation adaptation approaches modifies the probability of mutation during the evolutionary process of a GA, uniformly or non-uniformly, over the loci. Bäck [8] suggested a self-adaptive mutation approach. In this mechanism, each solution is attached a vector of mutation probabilities, which is as long as the length of the solution. The mutation scheme first mutates the mutation probability by itself then uses the mutated value as the mutation probability to mutate the i-th objective variable. The authors proposed a general expression which apparently modifies the mutation probability with time as follows:

$$P_m(t) = (\alpha/\beta)^{1/2} exp(-\gamma t/2)/(NL^{1/2}) \tag{3.6}$$

where α, β, and γ are fixed factors, N is the population size, L is the length of string, and t is the generation counter.

Fogarty [32] introduced a mechanism to vary the mutation rate over the generation for GAs with discrete representation. It is a deterministic approach that decreases the mutation rate exponentially over time. This scheme also provides bits of different significances with different schedules.

A statistic-based adaptive non-uniform mutation (SANUM) was proposed for GAs by Yang [118]. It takes into account the information of the allele distribution in each locus to adaptively alter the mutation rate of that locus with generation. Yang [118] has demonstrated that SANUM gives significant performance improvement over the traditional mutation operator on a set of test problems. An explorative and exploitative mutation scheme was suggested by [106], which is capable of exploring the unseen region of the solution space in search of the global optimum and simultaneously exploiting the already found promising area. It is a multi-population adaptive mutation approach and specifies different mutation rates for different sites of the solutions. The probabilities of sites are adapted according to the fitness and distribution of population during the evolutionary process.

In addition, some other researchers [84, 120] have suggested mechanisms to adapt the rate of mutation of each locus over generation.

3.9 Chapter Summary

In this chapter, we presented the background of adaptation in EAs, with special focus on adaptive mutation operators. The global taxonomy of parameter setting in EAs is also given in this chapter. We have discussed several schemes regarding the parameter settings of EAs that have been used to solve the optimization problems. Each scheme has its own benefits and drawbacks. It is evident that various adaptive algorithms in EAs have been introduced by researchers in order to find good solutions in the search space, e.g., adaptive population size approaches, adaptive mutation operators, adaptive crossover operators, and adaptive selection schemes.

We found that many researchers tested their approaches with fixed parameters and genetic operators in EAs in order to find the good solutions for a specific problem. However, it is very difficult to find the suitable parameter setting for the optimal performance, and this approach is also time consuming. Therefore, researchers have diverted their attention towards adjusting the operators and relevant parameters during the evolutionary process of EAs for finding good solutions to the problem in hand. It is evident that the optimal performance of EAs can be determined by adjusting genetic operators and relevant parameters. In addition, adaptation also holds a strong potential to be extended further in order to improve the performance of EAs both in real-world applications and in theoretical and empirical research in EAs.

Chapter 4

Adaptive Mutation Operators for Function Optimization

This chapter provides a comparative analysis of different population level and gene level adaptive mutation operators for evolutionary algorithms (EAs) for function optimization. We first briefly review each adaptive mutation scheme and then move on to compare these approaches. Several adaptive mutation operators have been investigated in this study. At the end of this chapter, the experimental results are presented and analyzed regarding the comparison of these operators on different benchmark function optimization problems.

4.1 Introduction

The adaptation of variation operators and relevant parameters was first suggested into evolutionary strategies where the mutation step size was successfully adjusted by self-adaptation, e.g., see Schwefel [79]. Recently, researchers have shown an increasing interest in the use of adaptive operators within the EA to enhance its performance. Generally speaking, different problems may have different properties due to different features of the fitness landscape. So, individuals in the EA may require different learning approaches to deal with these diverse properties. It may also be true even for a particular problem because the shape of a local fitness landscape in different sub-areas of a particular problem may be quite different, e.g., see Suganthan [97].

Adaptive Operator Selection (AOS) is the on-line adjusting technique that selects a genetic operator amongst a set of operators relevant to the problem. It is an important direction toward self-tuning for EAs, which uses the feedback information from the searching proccess. The overall framework of AOS consists of two main components: a credit assignment mechanism, which calculates a reward for each operator at-hand based on some rules from the statistical information of offspring; and an adaptation mechanism, which is capable to automatically choose one operator from different operators based on the allotted credit values.

The efficiency of EAs is dependent on not only the algorithms, representation, and operators for a problem, but also the setting of parameter values and operator probabilities for EAs. Operator adaptation can been classified into the population level, individual level, and component or gene level operator adaptation based on how operators are adjusted. At the population level, strategy parameters are evaluated from all offspring whom it has created from; different parents and genetic operators are modified globally for the whole population. At the individual level, operators are altered independently for each individual in the population. Finally, the component or gene level operator adaptation is done for some gene or component of individuals in the population. In this study, we will considere only Population Level Adaptive Mutation (PLAM) and Gene Level Adaptive Mutation (GLAM) for EAs.

4.2 Population-Level Adaptive Mutation Operators

Traditional EAs apply a single mutation operator at one iteration. A mutation operator can be used to increase the diversity of population in EAs. It is difficult to locate the best result by appling a single mutation operator. Therefore, various mutation operators may be used at different levels on a single problem to achieve the best results. Different mutation operators can be used to avoid being trapped into local optima. There are two population based mutation approaches considered in this section. One is an adaptive mutation operator in PSO and the other is an adaptive mutation operator in GAs.

4.2.1 Adaptive Mutation Operator for PSO

Different mutation operators can be used to help PSO jump out of local optima. However, a mutation operator may be more effective than others on a certain type of problems and may be worse on another type of problems. In fact, it is the same even for a specific problem at different stages of the optimization process. That is, the best mutation results can not be achieved by a single mutation operator, instead several mutation operators may have to be applied at different stages for the best performance. This study designs a mutation operator that can adaptively select the most suitable mutation operator for different problems. Before presenting the adaptive mutation operator, three mutation operators designed for the global best particles are described as follows.

4.2.1.1 Three Mutation Operators

A. Cauchy mutation operator

$$\vec{V}_g' = \vec{V}_g exp(\delta) \tag{4.1}$$

$$\vec{X}_g' = \vec{X}_g + \vec{V}_g' \delta_g \tag{4.2}$$

where $\vec{X}_g$ and $\vec{V}_g$ represent the position and velocity of the global best particle. δ and δ_g denote Cauchy random numbers with the scale parameter of 1.

B. Gaussian mutation operator

$$\vec{V}_g' = \vec{V}_g exp(N) \tag{4.3}$$

$$\vec{X}_g' = \vec{X}_g + \vec{V}_g' N_g \tag{4.4}$$

where $\vec{X}_g$ and $\vec{V}_g$ represent the position and velocity of global best particle. N and N_g are Gaussian distribution numbers with the mean 0 and the variance 1.

C. Levy mutation operator

$$\vec{V}_g' = \vec{V}_g exp(L(\alpha)) \tag{4.5}$$

$$\vec{X}_g' = \vec{X}_g + \vec{V}_g' L_g(\alpha), \tag{4.6}$$

where $L(\alpha)$ and $L_g(\alpha)$ are random numbers generated from the Levy distribution with a parameter α. In this study, α is set to 1.3.

In paper [60], a PSO with a population based adaptive mutation operator, denoted PLAM_PSO, was introduced. The proposed adaptive mutation operator uses the three mutation operators described above. All mutation operators have an equal initial selection ratio with 1/3. Each mutation operator is applied according to its selection ratio (i.e., the probability of the mutation operator to be selected among a set of mutation operators to perform a mutation operation) and its offspring fitness is evaluated. The mutation operators that result in higher fitness values of offspring have their selection ratios increased. The mutation operators that result in lower fitness values of offspring have their selection ratios decreased. Gradually, the most suitable mutation operator will be chosen automatically and control all the mutation behaviour in the whole swarm. Without loss of generality, we discuss the minimization optimization problems in this study.

First, some definitions are given below: The progress value $prog_i(t)$ of operator i at generation t is defined as follows:

$$prog_i(t) = \sum_{j=1}^{M_i} (f(p_j^i(t)) - \min(f(p_j^i(t)), f(c_j^i(t)))), \tag{4.7}$$

where $p_j^i(t)$ and $c_j^i(t)$ denote a parent and its child produced by mutation operator i at generation t and M_i is the number of particles that select mutation operator i to mutate.

The reward value $reward_i(t)$ of operator i at generation t is defined as follows:

$$reward_i(t) = exp(\frac{prog_i(t)}{\sum_{j=1}^{N} prog_j(t)}\alpha + \frac{s_i}{M_i}(1-\alpha)) + c_i p_i(t) - 1 \tag{4.8}$$

where s_i is the number of particles whose children have a better fitness than themselves after being mutated by mutation operator i, $p_i(t)$ is the selection ratio of mutation operator i at generation t, α is a random weight between $(0,1)$, N is the number of mutation operators, and c_i is a penalty factor for mutation operator i, which is defined as follows:

$$c_i = \begin{cases} 0.9, & \text{if } s_i = 0 \text{ and } p_i(t) = \max_{j=1}^{N}(p_j(t)) \\ 1, & \text{otherwise} \end{cases} \tag{4.9}$$

if the previous best operator has no contribution at current generation, then the selection ratio of the current best operator will decrease.

With the above definitions, the selection ratio of mutation operator i is updated according to the following equation:

$$p_i(t+1) = \frac{reward_i(t)}{\sum_{j=1}^{N} reward_j(t)}(1 - N * \gamma) + \gamma, \tag{4.10}$$

where γ is the minimum selection ratio for each mutation operator, which is set 0.01 for all the experiments in this study. This selection ratio update equation considers four factors: the progress value, the ratio of successful mutations, previous selection ratio, and the minimum selection ratio. Another important parameter for the adaptive mutation operator is the frequency of updating the selection ratios of mutation operators. That is, the selection ratio of each mutation operator can be updated at a fixed frequency, e.g., every U_f generations, instead of every generation.

The framework of a PSO algorithm without the adaptive mutation operator is shown in Algorithm 5. The PSO with the adaptive mutation operator differs from the algorithm 5 in Step 6. At Step 6, we select one of the three mutation operators according to their select ratios to mutate P_g for T times and then update the selection ratio for each mutation operator according to Eq. (4.10).

4.2.2 An Adaptive Mutation Operator for GAs

In [51], a GA with a population based adaptive mutation operator, denoted PLAM_GA, was proposed. This algorithm uses four mutation operators, denoted $M1$ to $M4$, respectively. The first operator $M1$ invert a bit value 0 to 1 and 1 to 0. An example is given below.

$$\text{Solution:} \quad 0\,1\,0\,1\,0\,0\,1\,1 \longrightarrow 1\,0\,1\,0\,1\,1\,0\,0$$

Algorithm 5 PSO without adaptive mutation

```
1: Generate the initial particles by randomly generating the position and velocity
   for each particle;
2: Evaluate each particle's fitness;
3: For each particle, if its fitness is better than the fitness of its previous best ($P_i$),
   update $P_i$;
4: For each particle, if its fitness is better than the fitness of the best one ($P_g$) of
   all the particles, update $P_g$;
5: Update each particle according to Eqs. (2.1) and (2.2) in Chapter 2;
6: Mutate $P_g$ according one of the three mutation operators for $T$ times ($T$ is the
   local search size for the global best particle). Then, compare the best one $P_g^*$
   with $P_g$, select a better one as the new global best particle;
7: Stop if the stop criterion is satisfied; otherwise, goto Step 3.
```

The $M2$ operator swaps any two bits in a single individual. An example is given below.

$$\text{Solution:}\quad 0\ 1\ \underline{\mathbf{0}}\ 1\ 0\ 0\ \underline{\mathbf{1}}\ 1 \longrightarrow 0\ 1\ \underline{\mathbf{1}}\ 1\ 0\ 0\ \underline{\mathbf{0}}\ 1$$

The third one reverses the interval order of bits in an individual. An example is given below.

$$\text{Solution:}\quad 0\ \underline{1\ 0\ 1\ 0\ 0}\ 1\ 1 \longrightarrow 0\ \underline{0\ 0\ 1\ 0\ 1}\ 1\ 1$$

The last one ($M4$) just changes one bit in an individual. An example is given below.

$$\text{Solution:}\quad 0\ 0\ 1\ 1\ 1\ 1\ 0\ \underline{\mathbf{0}} \longrightarrow 0\ 0\ 1\ 1\ 1\ 1\ 0\ \underline{\mathbf{1}}$$

These four mutation operators are used adaptively in the GA. All mutation ratios of these operators are assigned initial values, e.g, 0.1. Each mutation operator is applied by its mutation ratio. After the mutation operation, the progress value is calculated using the following equation:

$$progress_i(t) = \sum_{j=1}^{M_i}(\max[f(p_j^i(t)), f(c_j^i(t))] - f(p_j^i(t))) \tag{4.11}$$

where $progress_i(t)$ is the progress value of operator i at generation t, f is the fitness of an individual, $p_j^i(t)$ and $c_j^i(t)$ are the parent and its offspring produced by mutation operator i at generation(t), and M_i represent the total number of individuals that select the mutation operator i to mutate. The mutation ratio of operator i is updated according to their average progress value at generation(t), according to the following equation:

$$p_i(t+1) = \frac{progress_i(t)}{\sum_{j=1}^{N} progress_j(t)}(P_{mutation} - N * \delta) + \delta \tag{4.12}$$

where $p_i(t)$ is the mutation ratio of mutation operator i at generation t, N is the total number of mutation operators, $\delta = 0.01$ is the minimum mutation ratio for each mutation operator and $P_{mutation}$ means the initial mutation probability. The key idea behind PLAM is to apply more than one mutation operator on different stages to achieve the best result for a specific problem, at the same time the mutation ratio of the operator is updated by using the above formula.

The framework of the GA with the adaptive mutation operator is shown in Algorithm 6. An algorithm 6 is different from a standard GA in a few steps, such as Step 4, 10, and 11.

4.3 Gene-Level Adaptive Mutation Operators

In [118], a statistics-based adaptive non-uniform mutation (SANUM) was proposed for GAs, which is a gene level adaptive mutation operator. SANUM calculates the frequency of ones for each locus in the current population to adapt the mutation probability for that locus during the execution of the GA. If the amount of ones in alleles for a gene locus is increased (or decreased) over the population, that gene locus is called 1-inclined (or 0-inclined). A gene locus is called non-inclined if there is no trend of increasing or decreasing of 1's in the gene locus. The probability of

Algorithm 6 GA with adaptive mutation

```
1: $t := 0$;
2: Randomly initialize a population of individuals;
3: Evaluate the fitness of each individual;
4: Initialize the mutation ratio for each mutation operator equally;
5: repeat
6:     Assign various mutation operators according to their mutation ratios;
7:     Employ the crossover operator to create offspring;
8:     Mutate each offspring according to one of the four mutation operators;
9:     Evaluate the fitness of each offspring;
10:    Calculate the progress value of each operator according to Eq. (4.11);
11:    Update the mutation ratios of mutation operators according to their progress
       values Eq. (4.12);
12:    Select best individuals according to their fitness to form the next population;

13:    $t := t + 1$;
14: until the stop condition is satisfied
```

mutation for each locus i at generation t is adjusted by using following equation.

$$pm(i,t) = P_{max} - 2 * |f_1(i,t) - 0.5| * (P_{max} - P_{min}) \tag{4.13}$$

where $f_1(i,t)$ represent the frequency of 1's in the locus i over the population at generation t, $|x|$ returns the absolute value of x, P_{max} and P_{min} are the maximum and minimum value of the mutation probability for a locus.

In [84], the authors used an unparallel adaptive technique on each locus of a chromosome, called Gene Based Adaptive Mutation (GBAM). In GBAM, each gene locus has two different mutation probabilities: pm^1 is used for those loci that have the value of 1 and pm^0 is used for those loci that have the value of 0. Initially, all mutation probabilities are assigned to a value, e.g, 0.02. The probabilities of pm^1 and pm^0 are automatically updated based on the feedback information from the search space, according to the relative success or failure of those chromosomes having a "1" or "0" at that locus for each generation. The new mutation probability for each locus i at generation $t + 1$ is updated using the following equations in the case of a maximization problem.

$$pm^0(i,t+1) = \begin{cases} pm^0(i,t)+\gamma, & \text{if } G^1_{avg}(i,t) > P_{avg} \\ pm^0(i,t)-\gamma, & \text{otherwise} \end{cases} \tag{4.14}$$

$$pm^1(i,t+1) = \begin{cases} pm^1(i,t)-\gamma, & \text{if } G^1_{avg}(i,t) > P_{avg} \\ pm^1(i,t)+\gamma, & \text{otherwise} \end{cases} \tag{4.15}$$

where γ is the updated value for the mutation rate, $G^1_{avg}(i,t)$ is the average fitness of individuals with allele "1" for locus i at generation t, and $P_{avg}(t)$ is the average fitness of the population at generation t. The above update mechanism is used for each locus separately.

Another gene based adaptive mutation method, called GBAM_FAD, was proposed by Yang and Uyar [120] . This method constructs probabilities of each gene locus with the combination information of fitness and allele distribution. GBAM_FAD also uses two different mutation probabilities for each gene locus, just as in GBAM. The probabilities of each gene locus are adaptively updated based on the correlated feedback information from the search process, according to the relative success or failure of individuals. The new mutation probabilities for each locus i at generation $t+1$ are updated using the following equations in the case of maximization problems.

$$pm^0(i,t+1) = pm^0(i,t) + \gamma * sgn((G^1_{avg}(i,t) - P_{avg}(t))(f_1(i,t)-0.5)) \tag{4.16}$$

$$pm^1(i,t+1) = pm^1(i,t) - \gamma * sgn((G^1_{avg}(i,t) - P_{avg}(t))(f_1(i,t)-0.5)) \tag{4.17}$$

where $G^1_{avg}(i,t)$ and $P_{avg}(t)$ have the same meaning as in Eqs. (4.14) and (4.15), $f_1(i,t)$ is calculated frequency of ones in the alleles in the locus i over the population at generation t and the method $sgn(x)$ returns the value 1, 0, or -1 if $x>0$, $x=0$, and $x<0$, respectively. The GBAM_FAD algorithm efficiently solves the deception problems.

The framework of GAs with the above three gene based adaptive mutation operators is shown in Algorithm 7, where two additional steps (line 5 and line 10) are added to the design of Algorithm 6. At line 5, a statistics process calculates the feedback information of the current population, which is used in the adaptive mutation scheme in this algorithm. If the best individual of the current population is worse than the best individual of the previous population, the worst individual of the current population is replaced by the best one of the previous population, i.e., the elitism scheme is used in the algorithm (line 10).

Algorithm 7 GA with the gene based adaptive mutation operator

```
1:  $t := 0$;
2:  Initialize population with Random individuals;
3:  Evaluate each individual's fitness;
4:  while ($t < Tot\_gen$) do
5:      Statistics {feedback statistics taken from the current population};
6:      Select {parents};
7:      Crossover {pairs of parents};
8:      Mutate {the resulting offspring};
9:      Evaluate {new potential solutions};
10:     Elitism;
11:     $t := t + 1$;
12: end while
```

The aforementioned GLAM operators have already been investigated on simple uni-modal functions (having single peak) and multi-modal functions (having more than one peaks). The detailed description of these functions are given in [84, 118, 120]. The PLAM operators have also been investigated on various multi-dimensional problems. These operators are implemented on different benchmark optimization problems. It is very difficult to say which operator is more suitable for which problem. In order to better understand these operators, we compare their performance on a set of benchmark problems in this study.

4.4 Complexity Analysis

This section briefly analyzes the computational complexity of the five adaptive mutation operators studied in this study. Population Leve Adpative Mutation PLAM_PSO and PLAM_GA are population level adaptive mutation operators. The population size P and the total number of operators (N) are the key components for the analysis of computational complexity for the two adaptive mutation operators used in PLAM_PSO and PLAM_GA. The time complexity of each algorithm is $O(NP)$ in the worst case for one iteration. Usually, N is much less than P. So, we can say that the complexity of the mutation operators is $O(P)$. GBAM_FAD, GBAM, and SANUM are gene level adaptive mutation operators. There are three major components regarding to the time complexity of these adaptive operators, which are the population size (P), the length of individual (L), and the number of dimensions (n). The computional time of each gene level adaptive mutation operator is $O(nPL)$ in the worst case for a single iteration. Usually, P is greater than L and P is also greater than n. So, we can say that the time complexity of each gene level adaptive algorithm is $O(P^3)$.

4.5 Experimental Study

4.5.1 Test Functions

Experiments were carried out to compare the performance of several GAs with adaptive mutation operators. They are the PLAM_GA and the three GAs with SANUM, GBAM, and GBAM_FAD respectively, which are described in Section 4.2.2 and 4.3. We also tested the PLAM_PSO algorithm. The experiments were conducted on 26 different benchmark optimization functions, including traditional, shifted, rotated shifted and hybrid composition functions, which are widely used in the literature. These functions are classified into four different groups according to their properties: traditional functions f_1 to f_{15}, shifted functions $f_{16} - f_{20}$, hybrid composition functions f_{21}-f_{22} and rest of the functions are rotated and shifted f_{23}-f_{26}. The detailed description of these test functions is shown in Table 4.1.

1) Traditional test functions: we considered these functions to compare the performance of four GAs and one PSO with adaptive mutation operators. Fifteen functions are used in this section. Five out of fifteen are unimodal functions, which are Sphere (f_1), Rosenbrock (f_8), Schwefel_2_22 (f_{11}), Schwefel_1_2 (f_{12}), and Schwefel_2_21 (f_{13}) functions. The unimodal functions are relatively simple, with no local optimum. The rest of the functions in this group are multimodal problems. These problems have several local optima throughout the search space. These functions are used as more challenging tasks. Function f_7 is a noisy quartic function, where U(0,1) is a uniform distribution number within [0, 1].

2) Shifted test functions: Shifted functions $f_{16} - f_{20}$ are modified from present traditional test functions by moving the global optimum to a random point around the search space. The global optima of these functions were shifted according to:

$$F(\vec{x}) = f(\vec{x} - \vec{o}_r)$$

The new global optimum is estimated as: $\vec{o}_{new} = \vec{o}_r + \vec{o}_{old}$, where $\vec{o}_{old}$, $\vec{o}_r$, and $\vec{o}_{new}$ are the previous global optimum, the random position, and the newly generated global optimum, respectively.

3) Rotated shifted test functions: The performance of various adaptive mutation operators are also investegated with the use of the rotated shifted functions.

Table 4.1: The twenty six test functions, where n is the number of variables (dimensions) of a problem and D represents the domain of a problem ($D^n \subseteq R^n$), f_{min} the minimum value of each function.

Test Function	n	D	f_{min}
$f_1(x) = \sum_{i=1}^{n} x_i^2$	10	$[-100, 100]$	0
$f_2(x) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$	10	[-5.12, 5.12]	0
$f_3(x) = \sum\limits_{i=1}^{n} (\sum\limits_{k=0}^{k_{max}} [a^k \cos(2\pi b^k(x_i + 0.5))]) - n \sum\limits_{k=0}^{k_{max}} [a^k \cos(\pi b^k)]$ $, a = 0.5, b = 3, k_{max} = 20$	10	[-0.5,0.5]	0
$f_4(x) = \frac{1}{4000} \sum_{i=1}^{n}(x_i - 100)^2 - \prod_{i=1}^{n}\cos(\frac{x_i-100}{\sqrt{i}}) + 1$	10	[-600, 600]	0
$f_5(x) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i))$ $+20 + e$	10	[-32, 32]	0
$f_6(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$	10	[-100,100]	0
$f_7(x) = \sum_{i=1}^{n} i x_i^4 + U(0, 1)$	10	[-1.28, 1.28]	0
$f_8(x) = \sum_{i=1}^{n} 100(x_{i+1}^2 - x_i)^2 + (x_i - 1)^2)$	10	[-30, 30]	0
$f_9(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i \rvert})$	10	[-500, 500]	-4189.829
$f_{10}(x) = 418.9829 \cdot n + \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i \rvert})$	10	[-500, 500]	0
$f_{11}(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	10	[-10, 10]	0
$f_{12}(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$	10	[-100, 100]	0
$f_{13}(x) = \max_{i=1}^{n} \lvert x_i \rvert$	10	[-100, 100]	0
$f_{14}(x) = \frac{\pi}{30}\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2 \cdot [1 + 10\sin^2(\pi y_{i+1})]$ $+(y_n - 1)^2\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4), y_i = 1 + (x_i + 1)/4$	10	[-50, 50]	0
$f_{15}(x) = 0.1\{10\sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)^2 \cdot [1 + \sin^2(3\pi x_{i+1})]$ $+(x_n - 1)^2[1 + \sin^2(2\pi x_n)]\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	10	[-50, 50]	0
$f_{16}(x) = \sum_{i=1}^{n} z_i^2,\ \vec{z} = \vec{x} - \vec{o}$	10	$[-100, 100]$	0
$f_{17}(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} z_j)^2,\ \vec{z} = \vec{x} - \vec{o}$	10	[-100, 100]	0
$f_{18}(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} z_j)^2.\ (1 + 0.4\ .U(0, 1)),\ \vec{z} = \vec{x} - \vec{o}$	10	[-100, 100]	0
$f_{19}(x) = \sum_{i=1}^{n} 100(z_{i+1}^2 - z_i)^2 + (z_i - 1)^2),\ \vec{z} = \vec{x} - \vec{o}$	10	[-30, 30]	0
$f_{20}(x) = \sum_{i=1}^{n} (z_i^2 - 10\cos(2\pi x_i) + 10),\ \vec{z} = \vec{x} - \vec{o}$	10	[-5.12,5.12]	0
$f_{21}(x) =$ Hybrid Composition function (F15) in [97]	10	[-5, 5]	0
$f_{22}(x) =$ Rotated Hybrid Composition function (F16) in [97]	10	[-5, 5]	0
$f_{23}(x) = \frac{1}{4000} \sum_{i=1}^{n}(z_i - 100)^2 - \prod_{i=1}^{n}\cos(\frac{z_i-100}{\sqrt{i}}) + 1,$ $\vec{z} = (\vec{x} - \vec{o}) * M$	10	[-600, 600]	0
$f_{24}(x) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} z_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}$ $\cos(2\pi z_i)) + 20 + e,\ \vec{z} = (\vec{x} - \vec{o}) * M$	10	[-32, 32]	0
$f_{25}(x) = \sum_{i=1}^{n} (z_i^2 - 10\cos(2\pi z_i) + 10),\ \vec{z} = (\vec{x} - \vec{o}) * M$	10	[-5.12, 5.12]	0
$f_{26}(x) = \sum\limits_{i=1}^{n} (\sum\limits_{k=0}^{k_{max}} [a^k \cos(2\pi b^k(z_i + 0.5))]) - n \sum\limits_{k=0}^{k_{max}}$ $[a^k \cos(\pi b^k)],\ \vec{z} = (\vec{x} - \vec{o}) * M$	10	[-0.5,0.5]	0

In [97], the authors applied a method for rotated and shifted, which is used as follows:

$$F(\vec{x}) = f((\vec{x} - \vec{o}_r) * M)$$

where M is a linear transformation matrix. A rotation matrix $R_{ij}(\theta)$ is obtained by rotating the projection of $\vec{x}$ in the plane $i - j$ by an angle θ from the i-th axis to the j-th axis. The transformation matrix M is developed by Algorithm 8.

Algorithm 8 Transformatin Matrix

```
1: Randomly select $l$ dimensions ($l$ is an even number) from the $n$ dimensions to
   compose a vector $r = [r_1, r_2, \cdots, r_l]$.
2: For each pair of dimension $r[i]$ and dimension $r[i+1]$, construct a rotation matrix
   $R_{r[i],r[i+1]}(\theta)$, $\theta = U(0, 360)$, where $U$ reprents the random number of uniform
   distribution.
3: The transformation matrix M is obtained by: $M = R_{r[1],f[2]}(\theta) \cdot$
   $R_{r[3],r[4]}(\theta) \cdots R_{r[l-1],r[l]}(\theta)$
```

4) Composition Test Functions: In order to further evaluate the performance of various adaptive mutation operators, two hybrid composition functions, as in [97], are applied.

4.5.2 Parameter Setting

For the presentation of individuals, PLAM_PSO uses the real coding and GAs use the gray encoding scheme, where 100 bits are used for each dimension of the functions. The parameters of PLAM_PSO were set as suggested in [109] as follows: the acceleration constants $\eta_1 = \eta_2 = 1.496180$ and the inertia weight $\omega = 0.729844$. For all the GAs, the genetic operators were set as follows: the tournament selection with the tournament size of 2, elitism of size 1, 2-point crossover with a probability 1.0 and the population size $N = 250$. The initial selection ratio was 1/3 for each adaptive mutation operator and the minimum selection ratio γ was set to 0.001

for each adaptive mutation operator, the update frequency U_f was set to 5 and T in [60] was set to 10 for PLAM_PSO. For PLAM_GA, the initial probability was set to $P_{mutation} = 0.1$ and $\delta = 0.01$. For SANUM, the parameters were fixed as: $(\alpha, \beta) = (0.05, 0.04)$ and $P_{min} = 0.0001$ (i.e, $pm(i,t) \in [0.0001, 0.05]$ for each locus i). For GBAM and GBAM_FAD, the following parameters were used: $\gamma = 0.001$, $[P_{min}, P_{max}] = [0.0001, 0.2]$, and initially $pm^1(i,0) = pm^0(i,0) = 0.01$ for each gene i.

4.5.3 Experimental Results and Analysis

This section presents the average result of 50 independent runs of each algorithm on the test functions. For each run of an algorithm on a function, 300 generations were allowed. The experimental results are shown in Table 4.2.

From Table 4.2, several results can be seen. Firstly, the performance of PLAM_PSO is better than all other algorithms on all optimization benchmark functions except f_2, f_4, f_9, f_{10}, f_{20}, and f_{26}. For function f_2, the effiency of adaptive mutation operators is ranked in the following sequence: GBAM_FAD, GBAM, PLAM_GA, PLAM_PSO, and SANUM. On function f_4, GBAM is better than other four algorithms. The PLAM_PSO, GBAM_FAD, and GBAM algorithms obtain the global minimum optimum result within a few number of generations on f_6. The result of GBAM_FAD and GBAM are equal and better than the other three adaptive mutation algorithms on function f_9 and PLAM_GA gets a close result to GBAM_FAD and GBAM on the same function. GBAM, GBAM_FAD, and PLAM_GA are more efficient than the other two adaptive mutation operators on f_{10}. It can be observed that among the four algorithms with adaptive mutation operator gets optimum result on functions f_{20}, and f_{26}. Specially the effiency of PLAM_GA is better than

Table 4.2: Average result over 50 independent runs of algorithms on the test functions.

Function	*PLAM_PSO*	*GBAM_FAD*	*GBAM*	*SANUM*	*PLAM_GA*
f_1	**7.864e-19**	0.03678	0.04727	29.7191	0.17517
f_2	1.97063	**0.00400**	0.00609	13.5803	0.953409
f_3	**1.388e-07**	0.03773	0.03542	2.99221	0.769488
f_4	1.08447	0.21244	**0.19436**	1.11027	0.34151
f_5	**4.370e-10**	0.32794	0.30930	3.43966	2.43678
f_6	**0**	**0**	**0**	17	0.02
f_7	**0.00090**	0.00186	0.00209	0.19893	0.01141
f_8	**12.7173**	452.22	454.798	101665	821.919
f_9	-4061	**-4189**	**-4189**	-3800	-4188
f_{10}	204.562	0.04354	**0.03823**	369.191	1.45351
f_{11}	**6.010e-11**	0.00653	0.00885	0.562058	0.05199
f_{12}	**5.609e-05**	8.91879	11.3487	2455.3	26.7143
f_{13}	**2.027e-07**	0.60960	0.79703	17.9564	3.14116
f_{14}	**3.376e-20**	1.18793	0.83958	1.1945	0.21389
f_{15}	**2.371e-19**	24.8396	6.96587	15.9828	0.74665
f_{16}	**0.0378785**	6.17991	11.3346	227.919	7.78063
f_{17}	**2.46237**	301.771	233.052	3786.39	103.533
f_{18}	**7.796e-08**	205.474	214.617	4864.53	173.374
f_{19}	**1.93557**	7.22322	6.82506	30.1953	7.13815
f_{20}	3.79834	**2.01078**	2.22618	18.0262	3.65105
f_{21}	**40.1652**	233.092	245.237	509.324	257.529
f_{22}	**80.4926**	236.584	226.073	923.067	250.783
f_{23}	**0.197994**	2.18459	2.69248	15.7312	2.46165
f_{24}	**20.2595**	20.3125	20.3082	20.4051	20.4233
f_{25}	22.4549	19.3109	17.0176	73.3473	**17.0002**
f_{26}	5.10594	**3.86784**	3.90365	8.10701	4.68206

other mutation algorithms on f_{25}. Generally, the PLAM_PSO performed well among the four adaptive mutation operator on eighteen out of twenty six problems.

Secondly, the statistical test of five mutation operators in two groups (population-level adaptive mutation and gene-level adaptive mutation operator) is carried out using the two-tailed t-test with a 98 degree of freedom at a 0.05 level of significance. Table 4.3 shows the t-test results for pairs of algorithms, where the result is shown as "+", "−", or "∼" if the first algorithm in a pair is significantly better than, significantly worse than, or statistically equivalent to the second algorithm, respectively. From Table 4.3, it can be seen that PLAM_PSO, GBAM_FAD and GBAM algorithms are statistically better than the other two adaptive approaches in finding the optimum value.

Figures 4.1 and 4.2 show the results of the evolutionary process of the five algorithms on f_1 to f_5, f_7, f_9, f_{10}, f_{17} to f_{22}, f_{25}, and f_{26}. The results on f_1, f_3, and f_7 were presented by a log scale. From Figure 4.1, it can be noticed that the convergence speed of PLAM_PSO is much faster than other adaptive techniques on f_1 and f_3 functions, but it is interesting to see that different adaptive techniques have different convergence speeds on different functions. From Figure 4.2, it can be observed that the convergence speed of GBAM_FAD, and GBAM is close to that of PLAM_PSO. In Figure 4.2, GBAM_FAD and GBAM have a higher convergence speed than other four adaptive algorithms on f_{26}. The convergence of different adpative algorithms varies on different functions.

The overall performance of the five adaptive mutation algorithms are reasonably good except SANUM. Generally speaking, PLAM_PSO is the most efficient on most benchmark optimization functions. The performance of GBAM_FAD, GBAM, and PLAM_GA is better than PLAM_PSO on only a few problems. The reason lie in the encoding scheme used in these algorithms, which results in a discrete search space and makes it difficult for the algorithms to locate optimal solutions.

4.6 Chapter Summary

This chapter presents a comparative study of a population-level adaptive mutation operator with gene-level adaptive mutation operators for GAs and PSO on

TABLE 4.3: Statistical comparison of adaptive mutation operators on the test functions

Test function:	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
PLAM_PSO – GBAM_FAD	+	−	+	−	+	∼	+	+	−
PLAM_PSO – GBAM	+	−	+	−	+	∼	+	+	−
PLAM_PSO – SANUM	+	+	+	∼	+	+	+	+	+
PLAM_PSO – PLAM_GA	+	−	+	−	+	∼	+	+	−
GBAM_FAD – GBAM	∼	∼	∼	∼	∼	∼	∼	∼	∼
GBAM_FAD – SANUM	+	+	+	+	+	+	+	+	+
GBAM_FAD – PLAM_GA	+	+	+	+	+	∼	+	∼	+
GBAM – SANUM	+	+	+	+	+	+	+	+	+
GBAM – PLAM_GA	+	+	+	+	+	∼	+	∼	+
SANUM – PLAM_GA	+	+	+	+	+	+	+	+	+

Test function:	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}
PLAM_PSO – GBAM_FAD	−	+	+	+	+	∼	+	+	+
PLAM_PSO – GBAM	−	+	∼	+	+	∼	+	+	+
PLAM_PSO – SANUM	+	+	+	+	+	+	+	+	+
PLAM_PSO – PLAM_GA	−	+	+	+	+	+	+	+	+
GBAM_FAD – GBAM	∼	∼	∼	∼	−	∼	∼	∼	∼
GBAM_FAD – SANUM	+	+	+	+	∼	∼	+	+	+
GBAM_FAD – PLAM_GA	+	+	+	+	−	∼	∼	−	∼
GBAM – SANUM	+	+	+	+	∼	∼	+	+	+
GBAM – PLAM_GA	+	+	+	+	−	∼	∼	∼	∼
SANUM – PLAM_GA	+	+	+	+	+	+	−	−	−

Test function:	f_{19}	f_{20}	f_{21}	f_{22}	f_{23}	f_{24}	f_{25}	f_{26}
PLAM_PSO – GBAM_FAD	+	−	+	+	+	+	−	−
PLAM_PSO – GBAM	+	−	+	∼	+	+	−	−
PLAM_PSO – SANUM	+	+	+	+	+	+	∼	+
PLAM_PSO – PLAM_GA	+	∼	+	+	+	+	−	∼
GBAM_FAD – GBAM	∼	∼	∼	∼	∼	∼	∼	∼
GBAM_FAD – SANUM	+	+	+	+	+	+	+	+
GBAM_FAD – PLAM_GA	∼	+	∼	∼	∼	+	∼	+
GBAM – SANUM	+	+	+	+	+	+	+	+
GBAM – PLAM_GA	∼	+	∼	+	∼	+	∼	+
SANUM – PLAM_GA	−	−	−	−	−	∼	−	−

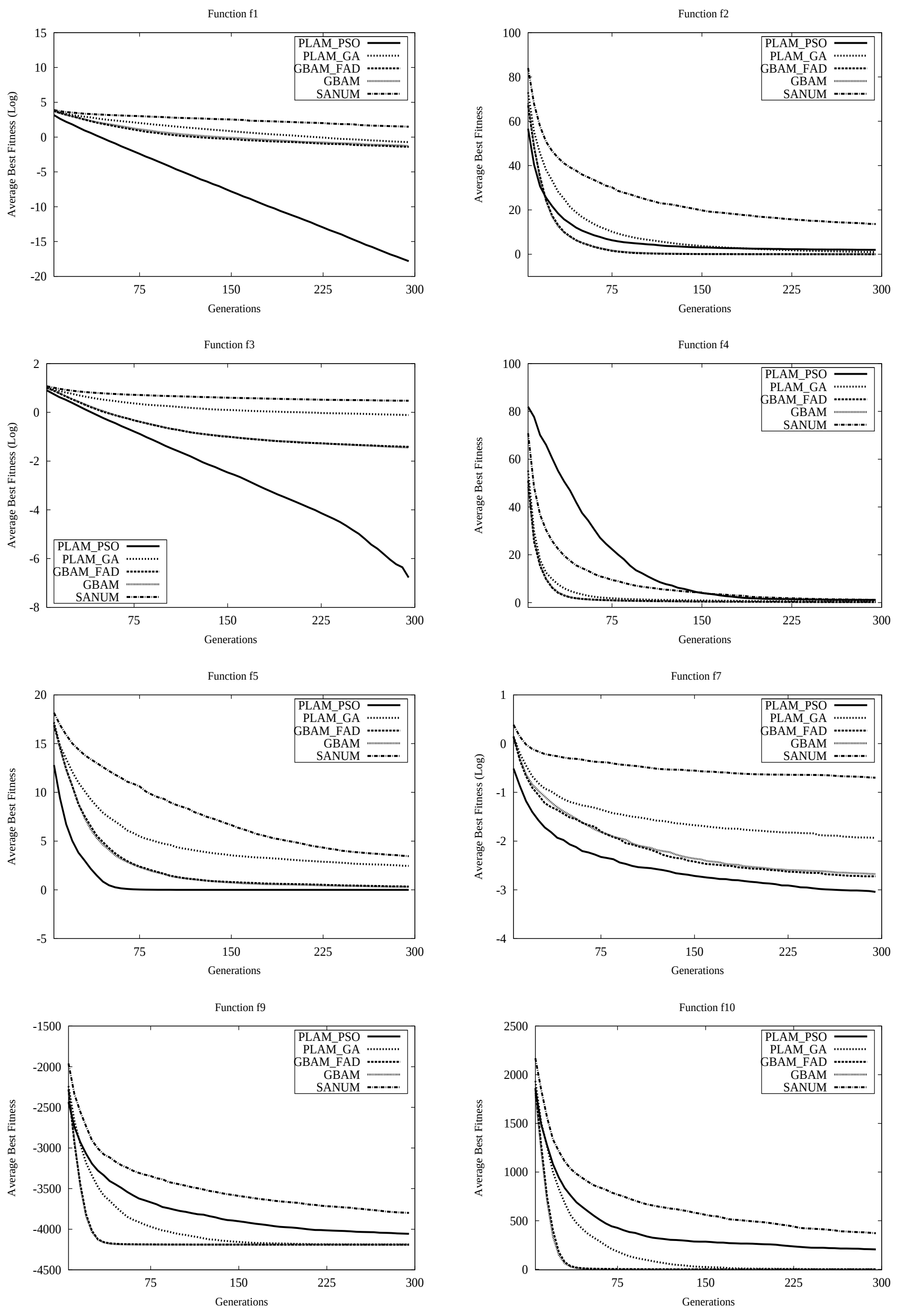


FIGURE 4.1: Experimental results of adaptive mutation operators.

multi-dimensional benchmark functions. The performance of different adaptive mutation operators varies on different functions. From the experimental results, it

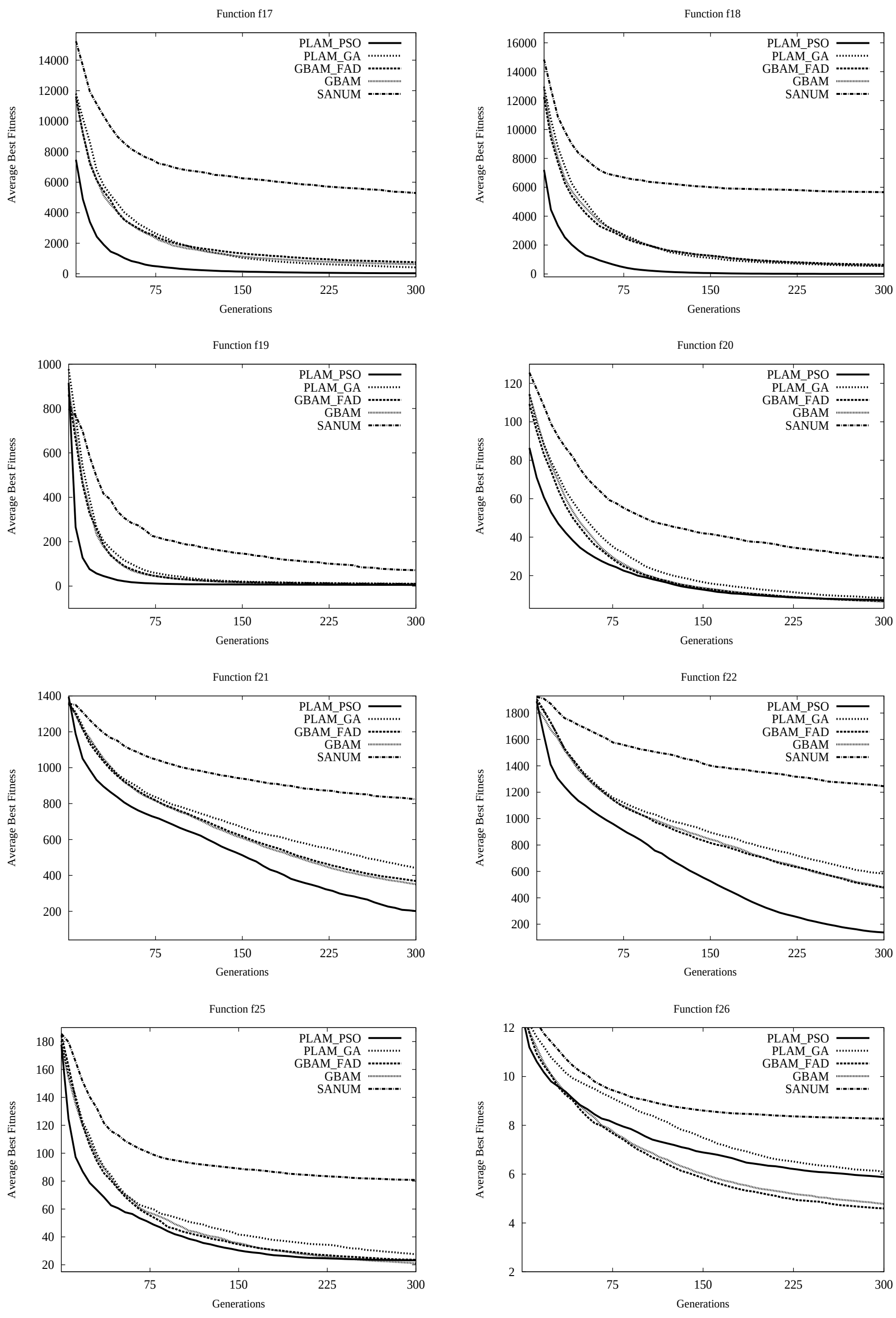


Figure 4.2: Experimental results of adaptive mutation operators.

can be concluded that PLAM_PSO is the most efficient on all benchmark optimization problems with only a few exceptions. On some functions, GBAM_FAD, GBAM, and PLAM_GA mutation algorithms perform better than PLAM_PSO.

With PLAM_PSO, GBAM_FAD and GBAM, the population rapidly converges in a relatively short period of time to a near-optimal solution even for multi-modal functions. PLAM_PSO, GBAM_FAD and GBAM are statistically better than other adaptive approaches for finding the optimum value.

In general, the experimental results indicate that PLAM_PSO provides better solutions with a reduced number of generations on most test problems. It is also investigated that the performance of GBAM_FAD and GBAM is close to that of PLAM_PSO. GLAM operators are better than PLAM_PSO on very few problems only. The drawback with the GLAM operators is that it takes some time to calculate new mutation probabilities for each gene locus at every generation. We will examine the performance of individual level adaptive mutation operators with population level and gene level adaptive mutation operators on benchmark problems GECCO 2009 [73] or CEC 08-05 [97]; and how to reduce the complexity of gene level adaptive mutation operators in the future.

Chapter 5

Directed Mutation for Real-Coded Genetic Algorithms

5.1 Introduction

Mutation is an important operator in genetic algorithms (GAs), which maintains the genetic diversity in the population in order to get better individuals to an optimization problem. The step size and search direction are the major factors that determine the performance of mutation operators. It may be beneficial to use different values during different stages of evolution in order to get a better performance of GAs. Traditional real-coded GAs use one of uniform and non-uniform mutation operators for creating the next generation, which is unable to guide the algorithm regarding fast convergence toward promising regions of the fitness landscape of a given problem.

This chapter investigates a directed mutation (DM) operator for GAs to explore promising areas in the search space. The DM operator uses a concept of induced mutation in biological systems. According to induced mutation, the statistical information is used to guide the mutation of an individual towards the neighbouring interval that has a better statistics result in each dimension.

Several researchers have tried to increase the performance of real-coded GAs by using directed mutation techniques [14, 15, 46, 99]. In [15], the authors proposed a co-evolutionary technique, where each component of a solution vector is added one extra bit to determine the direction of mutation. The direction bit is adapted by using the feedback information from the current population. A directed mutation based on momentum was proposed in [99], where each component of an individual is attached a standard Gaussian mutation and the current momentum to mutate that component.

In the proposed DM method, the statistical information regarding the fitness and distribution of individuals over intervals of each dimension is calculated according to the current population and is used to guide the mutation of an individual towards the neighbouring interval that has the best statistical result in each dimension.

5.2 Directed Mutation for Genetic Algorithms

The main motivation behind the DM operator is to explore promising areas in the search space by using the feedback information from the current population, e.g., the fitness and some other factors. It is a modified version of the standard mutation. Since individuals in directed variation (DV) [125] only move toward the interval with the highest fitness, it may easily cause the premature convergence problem. This chapter introduces a DM technique which aims to explore promising areas of the search space with fixed boundaries according to the fitness of intervals and the percentage of individuals in each interval of each dimension.

In the proposed DM operator, individual shifting is not only based on the feedback information of the average fitness of intervals, but also on the population distribution. By taking into account the information of population distribution, DM efficiently avoids the premature convergence problem. The key idea of the DM operator is illustrated in Figure 5.1.

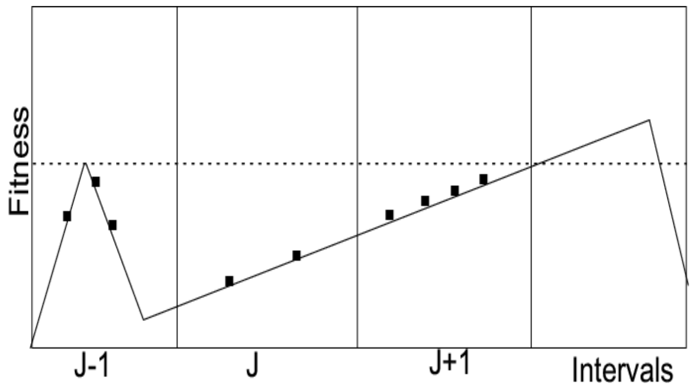


FIGURE 5.1: Fitness landscape of the d-th dimension.

From Figure 5.1, we consider the j-th interval, if we only consider DV, the two individuals of interval j will move toward the $(j-1)$-th interval due to the higher fitness of the $(j-1)$-th interval. However, the right direction should be the $(j+1)$-th interval since the $(j+1)$-th interval is more promising (because it contains more individuals) than the $(j-1)$-th interval. Hence, DM is an enhanced version of DV.

The framework of the GA with the proposed DM operator is given in Algorithm 9. The proposed GA differs from the standard GA in that in each generation, a set of individuals are selected to undergo the DM operation iteratively. As shown in Algorithm 10, the DM operator is applied for each component of a selected solution in a similar way as the DV operator described in Section 3.2. The difference lies in the calculation of the probability of moving a component of a solution from one interval to its neighbouring interval, which is described in detail below.

Algorithm 9 GA with directed mutation (DM)

```
 1: Randomly generate an initial population *pop*
 2: Evaluate the fitness of each individual of *pop*
 3: $t := 0$.
 4: **while** $t < max\_gen$ **do**
 5:    **for** each individual $i$ in *pop* **do**
 6:       Select individual $j$ by the roulette wheel method
 7:       Crossover individual $i$ with individual $j$ using the arithmetic crossover
          method
 8:       Mutate individual $i$ by using the Gaussian mutation with mean zero and
          pre-selected or adaptive standard deviation
 9:    **end for**
10:    Apply DM on a set of individuals randomly selected from the population
11:    $t := t + 1$
12: **end while**
```

Algorithm 10 Directed mutation

```
 1: **for** each dimension $d \in \{1, 2, \cdots, K\}$ **do**
 2:    **if** $|R_d(t)| < |R_d(t-1)|$ **then**
 3:       **for** each interval $j$ **do**
 4:          Calculate $F_{dj}$ according to Eq. (3.1)
 5:          Calculate the number of individuals $P_{dj}$ according to Eq. (5.1)
 6:       **end for**
 7:       **for** each interval $j$ **do**
 8:          Calculate $FP_{dj}$ according to Eq. (5.2)
 9:       **end for**
10:       **for** each interval $j$ **do**
11:          Calculate $P_{dj}^{DM}$ according to Eq. (5.3)
12:       **end for**
13:       Shift the genes $x_{id}$ of selected individuals to their neighbouring interval with
          a higher fitness with the associated probability
14:    **end if**
15: **end for**
```

Similar to the DV operator, the range of the d-th dimension of individuals at generation t, i.e., $R_d(t) = [x_d^L, x_d^U]$, is also equally divided into L intervals. Note that $|R_d(t)|$ and $|R_d(t-1)|$ in Algorithm 10 denote the length of the range at time t and $t-1$, respectively. DM is applied only when the range of the d-th component $R_d(t)$ of all solution vectors of current generation t decreases in comparison with that of previous generation $t-1$. The fitness F_{dj} of each non-empty interval is calculated by Eq. (3.1), Eq. (3.2), and Eq. (3.3). In addition to the fitness of each interval, the percentage of individuals in each interval is also calculated in the DM operator as follows:

$$P_{dj} = \frac{1}{N}\sum_{i=1}^{N} I(x_{id} \in B_{dj}) \tag{5.1}$$

where P_{dj} represents the percentage of individuals with the d-th component in the j-th interval in the current population, and B_{dj} and $I(.)$ are the same as defined before in Eq. (3.1) and Eq. (3.2). From F_{dj} and P_{dj}, we calculate a value that is associated with the j-th interval of the d-th dimension, assuming $F_{d,j-1} > F_{dj} > F_{d,j+1}$, as follows:

$$FP_{dj} = \frac{F_{dj}}{F_{d,j-1}} + \frac{P_{dj}}{P_{d,j-1}} \tag{5.2}$$

With above definitions, the component x_{id} of a selected individual $\vec{x}_i$ is mutated by the associated value of FP_{dj}. Where F_{dj} is bigger than the fitness of both neighbouring intervals, i.e., $F_{d,j-1}$ and $F_{d,j+1}$, no directed mutation will be used to x_{id}. If F_{dj} is in the middle in comparison with the fitness of its two neighbour intervals $j-1$ and $j+1$, without loss of generality, suppose $F_{d,j-1} > F_{dj} > F_{d,j+1}$. Then, move the individual $\vec{x}_i$ towards the interval $j-1$ with a certain probability, which is calculated as follows.

$$P_{dj}^{DM} = \frac{FP_{dj}}{\sum_{j=1}^{L} FP_{dj}} \tag{5.3}$$

where the DM probabilities P_{dj}^{DM} are normalized over all intervals. In this case, the solution $\vec{x}_i$ is moved toward the interval $j-1$ by replacing x_{id} with a number randomly generated between x_{id} and the center of $B_{d,j-1}$ as follows:

$$x_{id} = rand(x_{id}, B_{d,j-1}) \tag{5.4}$$

Otherwise, if $F_{dj} < F_{d,j-1}$ and $F_{dj} < F_{d,j+1}$, then either $B_{d,j-1}$ or $B_{d,j+1}$ is selected with an equal probability and the solution $\vec{x}_i$ moves towards the selected interval with the probability P_{dj}^{DM}.

5.3 Experimental Study

5.3.1 Experimental Setting

In order to test the performance of GA with DM, three f_1, f_6, and f_8 uni-modal functions and eleven multi-modal functions, which are widely used as the test functions in the literature [97, 122], were selected as the test bed in this study. The number of dimensions n is set to 10 for all test functions. The details of these test functions are given in Table 5.1. Function f_9 is a composition function proposed by Suganthan et al. [97], which is composed of ten benchmark functions: the rotated version and shifted version of f_1, f_2, f_3, f_4, and f_5, as also listed in Table 5.1, respectively. Functions f_{10} to f_{14} are rotated functions, where the rotation matrix $\vec{M}$ for each function is obtained using the method in [75].

The idea of DV was taken from [125], which is implemented in the peer GA. The adaptive standard deviation [80] is used in DM. The population size (100) and total number of generations (500) are the same for both DM and DV on all problems, the total number of intervals L was set to 3, 6, 9, and 12, respectively. The mutation probability P_m for the Gaussian mutation is the same for DM and DV, which was set to different values for different test problems, as listed in Table 5.1. Both the GA with DM and the GA with DV were run 30 times independently on each test problem.

Table 5.1: Test functions of $n = 10$ dimensions, where D ($D \in R_n$) and f_{min}, denote the domain and minimum value of a function, respectively, and $\vec{M}$ is the rotation matrix

Test Function	P_m	D	f_{min}
$f_1(x) = \sum_{i=1}^{n} x_i^2$	0.1	$[-100, 100]$	0
$f_2(x) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$	0.01	[-5.12,5.12]	0
$f_3(x) = \sum\limits_{i=1}^{n} \sum\limits_{k=0}^{k_{max}} [a^k \cos(2\pi b^k (x_i + 0.5))] - n \sum\limits_{k=0}^{k_{max}} [a^k \cos(\pi b^k)]$, $a = 0.5, b = 3, k_{max} = 20$	0.01	[-0.5,0.5]	0
$f_4(x) = \frac{1}{4000} \sum_{i=1}^{n} (x_i - 100)^2 - \prod_{i=1}^{n} \cos(\frac{x_i - 100}{\sqrt{i}}) + 1$	0.01	[-600, 600]	0
$f_5(x) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i))$ $+20 + e$	0.01	[-32, 32]	0
$f_6(x) = \sum_{i=1}^{n} 100(x_{i+1}^2 - x_i)^2 + (x_i - 1)^2)$	0.05	[-30, 30]	0
$f_7(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i \rvert})$	0.01	[-500, 500]	-4189.829
$f_8(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$	0.01	[-100, 100]	0
$f_9(x) =$ Composition function (CF 5) in [97]	0.05	[-5, 5]	0
$f_{10}(x) = \sum_{i=1}^{n} 100(y_{i+1}^2 - y_i)^2 + (y_i - 1)^2), \vec{y} = \vec{M} * \vec{x}$	0.05	[-100, 100]	0
$f_{11}(x) = \frac{1}{4000} \sum_{i=1}^{n} (y_i - 100)^2 - \prod_{i=1}^{n} \cos(\frac{y_i - 100}{\sqrt{i}}) + 1,$ $\vec{y} = \vec{M} * \vec{x}$	0.01	[-600, 600]	0
$f_{12}(x) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} y_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi y_i))$ +20+e, $\vec{y} = \vec{M} * \vec{x}$	0.01	[-32,32]	0
$f_{13}(x) = \sum_{i=1}^{n} (y_i^2 - 10\cos(2\pi y_i) + 10), \vec{y} = \vec{M} * \vec{x}$	0.01	[-5, 5]	0
$f_{14}(x) = \sum\limits_{i=1}^{n} \sum\limits_{k=0}^{k_{max}} [a^k \cos(2\pi b^k (y_i + 0.5))] - n \sum\limits_{k=0}^{k_{max}} [a^k \cos(\pi b^k)]$, $a = 0.5, b = 3, k_{max} = 20, \vec{y} = \vec{M} * \vec{x}$	0.01	[-0.5,0.5]	0

5.3.2 Experimental Results and Analysis

The average results of 30 independent runs of the GA with directed mutation and the GA with directed variation on the test problems are shown in Table 5.2. From Table 5.2, it can be seen that the number of intervals used for each dimension is a key parameter. The performance of the GA with DM becomes significantly better on some problems than that of the GA with DV as the number of intervals increases. The performance of both operators is different on different problems.

Table 5.2: Comparison results between DV and DM with the number of intervals for each dimension set to different values for different problems

function		f_1	f_2	f_3	f_4	f_5	f_6	f_7
$L = 3$	DV	8.28e-06	0.0151	0.2590	0.0985	0.0088	49.41	-2667
	DM	7.33e-06	0.0122	0.2304	0.1116	0.0082	33.87	-2062
$L = 6$	DV	7.55e-06	0.0093	0.2030	0.0820	0.0114	95.25	-2692
	DM	8.99e-06	0.0172	0.1837	0.0385	0.0090	18.84	-1797
$L = 9$	DV	6.19e-06	0.0087	0.2181	0.0731	0.0076	68.78	-2574
	DM	9.03e-06	0.0067	0.2558	0.1143	0.0089	22.97	-1857
$L = 12$	DV	7.21e-06	0.0191	0.2271	0.0972	0.0107	16.63	-2578
	DM	7.86e-06	0.0153	0.2196	0.0341	0.0080	5.65	-1905
t-test	DV-DM	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	+	-
function		f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
$L = 3$	DV	2.7677	102	61	0.0929	0.0476	2.0971	0.4200
	DM	3.5671	100	21	0.0891	0.0093	2.8655	0.4794
$L = 6$	DV	2.6971	64	82	0.1147	0.0470	2.5969	0.4283
	DM	1.9957	18	14	0.0386	0.0089	2.1034	0.3483
$L = 9$	DV	3.8969	94	30	0.0726	0.0974	2.1034	0.4099
	DM	3.8160	24	24	0.0998	0.0077	2.7950	0.4833
$L = 12$	DV	2.0960	230	50	0.0826	0.0472	3.0315	0.4175
	DM	2.7178	63	53	0.0444	0.0086	1.9218	0.4180
t-test	DV-DM	$\sim$	+	+	$\sim$	+	$\sim$	$\sim$

When the number of intervals is set to 3, the results of DM are better than DV on half of the test problems. DM is trapped into local optima due to the large range of intervals. It is interesting that DM achieves the best result on f_9, f_{10}, and f_{12} over all different number of intervals.

When we increase the number of intervals to 6, the performance of DM is better than DV on most benchmark problems. Although DV presents better results than DM on f_1, f_2, f_9, and f_{14}, DM obtains close results to DV on these functions.

Similar observations can be made as the number of intervals increases to 9. The results obtained by DM are better than that of DV. Compared with the results of DM with the number of intervals of 6, the performance of DM deteriorates on some multi-modal problems. However, the results of DM with the number of intervals of

9 are better than the results with the number of intervals of 6 on most uni-modal problems.

When $L = 12$, the results of DM are better than the results of DV on more than half of the test functions. Similar results can be viewed as the number of interval 6 but the performance of DM increases compared with the number of interval of 9 on some multi-modal problems.

Table 5.2 also shows the statistical analysis of comparing DM with DV when $L = 6$ by using the two-tailed t-test with a 58 degree of freedom at a 0.05 level of significance, where the t-test result is presented as "$+$", "$-$", or "$\sim$" if the performance of the GA with DM is significantly better than, significantly worse than, or statistically equivalent to the GA with DV, respectively. The DM operator is significantly better on four problems, significantly worse on one problem, and statistically similar on the rest of the problems.

From Table 5.2, three conclusions can be made. First, the overall performance of DM is better than DV on half test functions at least. Especially, on f_9, f_{10} and f_{12}, the performance of DM is better over all different settings of the number of intervals. Second, the interval quantity is a crucial factor to the performance of both DM and DV on different benchmark functions. According to variable number of intervals, the result of DM and DV varies on different test problems. Third, a larger number of intervals is needed on multi-modal problems than uni-modal problems. The smaller number of intervals causes the larger number of local optima within an interval, since multi-modal problems have many local optima.

Figure. 5.2 presents the evolutionary process for DM and DV operators on f_4, f_6, f_7, and f_9 respectively, where the result on f_4 is presented in a log scale. From Figure 5.2, it can be seen that the convergence speed of DM is faster than that of

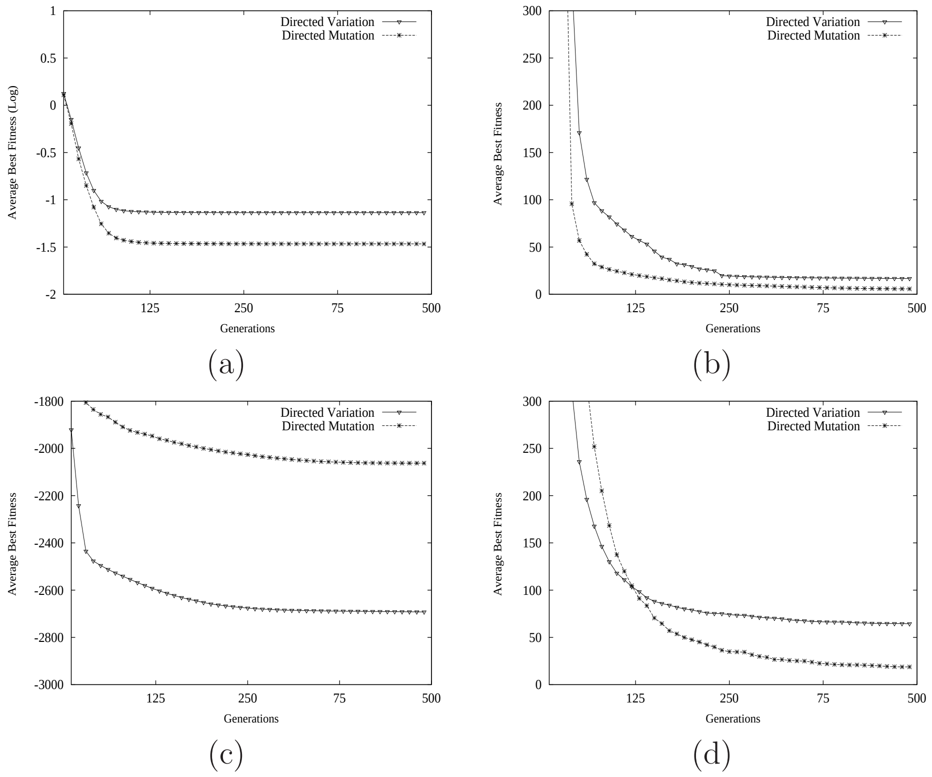


Figure 5.2: Evolutionary progress of directed mutation and directed variation operators on (a) f_4 with $L = 12$, (b) f_6 with $L = 12$, (c) f_7 with $L = 3$, and (d) f_9 with $L = 6$.

DV except on f_9. This result validates our idea that the performance of DV can be enhanced by taking into account the population distribution in the search space.

5.4 Chapter Summary

In this chapter, a directed mutation operator is proposed for genetic algorithms to explore promising solutions in the search space. In the proposed directed mutation, individual shifting is not only based on the feedback information of the fitness of each interval, but also on the population distribution. By taking into account the information of the population distribution, directed mutation greatly improves the performance of directed variation.

In order to justify the proposed directed mutation, a set of benchmark functions was used as the test base to compare the performance of the directed mutation operator with the directed variation operator from the literature [125]. The experimental results show that the efficiency of DV is improved by the proposed enhancement on four out of fourteen functions.

Although the prooposed DM operator achieves better results on the test problems, there is a limitation, i.e., different problems need different optimum values of the number of intervals to achieve the best result. So, how to adaptively adjust the number of intervals is our major work in the future, and we will consider to compare the performance of proposed DM technique with DV on CMA-ES benchmark problems.

Chapter 6

Multi-Population with Adaptive Mutation for Multi-Modal Optimization

This chapter investigates multi-population approaches with adaptive mutation in evolutionary algorithms (EAs) for multi-modal optimization problems. This idea is the combination of multiple population and adaptive mutation approaches. We will start with discussion on some challenging issues in EAs for multi-modal optimization problems. Then, we present the general consideration of the multi-population approach and some recently developed EAs with multi-population approaches. After that, we describe in detail the proposed multi-population with adaptive mutation approach for genetic algorithm (GAs) for multi-modal optimization problems. Thereafter, we will present the experimental study of the proposed algorithm based on a set of benchmark problems.

6.1 Challenges to EAs for Multi-Modal Optimization

Much research has been done on EAs to find multiple peaks of a multi-modal optimization problem. So far, the major challenge on EAs for multi-modal optimization problems is how to avoid the stagnation of EAs to local optima. Generally speaking, in a well-designed algorithm, the population will evolve towards the global optimum. In this case, the diversity in the population will decrease. If all of the solutions have similar gene values, the population is said to be converged. If an algorithm has converged somewhere in the search space, it means no new candidate solution can be generated in the entire search space. An algorithm is said to be prematurely converged if it has converged to a local optimum and there are still better points existing in the fitness landscape than the area that has been currently searched. This problem of premature convergence is a challenging task, especially when the search space contains numerous local optima.

The loss of diversity is a major drawback in EAs. Usually, EAs may get stuck to premature convergence due to the loss of diversity in the population. If so, it is very difficult for the EAs to make any further progress in the searching of better solutions. This problem can be relieved by using some mechanisms to maintain or increase the diversity in the population. It is a very important issue to maintain the diversity in the population, which is relevant to how to balance the exploration and exploitation capacities for EAs. Exploitation is a process of refining solutions with small changes to the current individuals while exploration is the ability of a search algorithm to explore new areas of the search space. Hence, another widespread opinion is that exploitation and exploration are opposite forces, to guide the search behaviour of individuals.

From the algorithm point of view, almost all EAs also face this problem. If an algorithm uses the exploitation mechanism, it normally has a fast convergence speed, but may get stuck to local optima easily. If an algorithm employs the exploration approach, it may never improve potential solutions well enough to find global optima, or it may achieve good solutions by chance but may need a long time.

Several approaches have been recently suggested to help EAs jump out of local optima. Multi-population is an efficient technique to enhance the diversity in the population. The following section first gives the general consideration of the multi-population approach for EAs and then describes a few recently developed EAs with multi-population approaches.

6.2 EAs with Multi-Population Approaches

6.2.1 General Consideration of Multi-Population

The fundamental idea of multi-population approaches is to generate a number of sub-populations from the available whole population of individuals and maintain these sub-populations on different promising peaks of the fitness landscape. This technique can be used to enhance the diversity of the population. The fitness landscape contains more than one peak in the context of multi-modal problems: the highest fitness peak represents the global optimum and other peaks with lower heights are local optima. Hence, the working mechanism of multi-population is to divide the search space into various sub-regions. One or more than one peak might be covered by each sub-area, and each sub-population will reside in one sub-area and exploit that sub-area.

Several key issues are raised, when using the multi-population approach to find multiple peaks in the fitness landscape [119]. These key issues are summarized as follows. The first key issue is how to divide the individuals to different promising sub-areas. If the individuals can not be effectively distributed to different promising sub-regions in the search space, then a GA can not find multiple optima. This issue plays an important role. The motivation behind this issue is that an algorithm should have the ability to improve the global search capability to explore promising sub-regions.

After description of the first issue, we face another problem – how to set the area of each sub-region in the fitness landscape. The number of peaks are determined by the area of sub-regions. If the area of a sub-region is too small, there is a potential problem that a small isolated sub-population may get stuck on local optima. In this case, it can be difficult for the algorithm to make any further progress due to the lose of diversity. However, if a sub-region becomes too large, it is possible that the sub-region, which will be covered by a sub-population, will contain more than one optimum within it.

The third issue is related to how many sub-populations are needed. It is possible that the optimal number of sub-populations is equal to the total number of peaks in the landscape. In this context, the more optima in the search space, the more sub-populations may be needed. However, the limited computational resources may be wasted when using the large number of sub-populations. If there is too small number of sub-populations, GAs can not quickly detect different local optima. Generally speaking, the number of peaks in the search space is unknown in advance, which is the potential problem especially in real-world applications. Although for some benchmark problems, the number of peaks is determined, over all, we should consider that it is unknown to us.

Finally, there is also one open issue, that is, how to create sub-population? Usually, the entire population is divided into sub-populations by using some approaches. Many researchers have recently suggested different mechanisms for EAs [38, 83, 104, 119, 121]. These schemes can be used to enhance the diversity in the population, with the aim of maintaining those sub-populations on different peaks of the fitness landscape.

6.2.2 Recent Multi-Population Approaches for EAs

6.2.2.1 Species-Based Multi-Population Approach

A species-based particle swarm optimization (SPSO) algorithm was proposed by Parrot and Li [70]. A species can be taken as a collection of individuals sharing the common characteristics in terms of similarity of individuals. The similarity between two individuals can be measured by the Hamming distance between them for binary encoded EAs. The smaller the Hamming distance between two individuals, the more similar they are. A species depends on the parameter *min_dist*, which represents the species radius (or similarity threshold). The species radius is estimated by the Hamming distance from the center of species to its boundaries. The center of a species is called the species seed, which is usually the fittest individual in the species.

In SPSO, the algorithm of determining the species seeds is given in Algorithm 11. The algorithm receives one sorted list of all particles in the population. The particles of the list are arranged in the order of non-increasing fitness values. The algorithm for identifying species seeds has to be applied at each iteration of SPSO. All the particles in the ordered list are examined from the best to the worst against the species seeds identified so far. Once the fittest individual of unselected ones in the ordered list has been determined as a species seed, all unselected individuals in the

Algorithm 11 Identifying species seeds

```
1: input: $L_d$ – A list of all particles sorted in non-increasing fitness values
2: output: $L_S$ – A list of particles identified as species seeds
3: $L_S := \phi$;
4: for each particle $p \in L_d$ do
5:    $seed := true$;
6:    for each seed $s$ in $L_S$ do
7:       if $(distance(p, s) <= min\_dist)$ then
8:          $seed := false$;
9:          break;
10:      end if
11:   end for
12:   if $seed = true$ then
13:      add the particle $p$ as a species seed into $L_S$;
14:   end if
15: end for
```

list that are within the species radius distance to the species seed are classified to the same species. If an individual is not around all species seeds (i.e., not within the species radius to all existing species seeds), this individual will become a new species seed. If the radii of species seeds are overlapped, the first determined species seed will dominate over the other seeds that are determined later on. Any individual which is included within the overlapped radii, will be classified to the species corresponding to the dominating seed. For example, suppose that the algorithm creates three different species seeds s_1, s_2, and s_3, in this order (i.e., s_1 is generated first, s_2 the second, and s_3 the last) and one individual x is within the overlapped radii of s_1 and s_2, then x will be classified into the species with the seed s_1 since s_1 dominates s_2.

In SPSO, each sub-population covers one sub-region in the fitness landscape, which may contain one or multiple peaks. To address the major problem with regard to the efficiency of SPSO, a few individuals of the same species may have converged to one optimum before different species converge to other optima. All the individuals of a same species that have converged to the same global optimum are called redundant.

Algorithm 12 Replacing redundant solutions

```
1: input: $L_d$ – A list of all particles sorted in non-increasing fitness values
2: $maxSize := L_d.size()$;
3: for $i := 1$ to $maxSize$ do
4:     if $L_d[i].fitness = L_d[i].seedfitness$ then
5:         Mark the $i$-th individual in $L_d$ as $redundant$;
6:     end if
7: end for
8: Delete all individuals marked as $redundant$ from $L_d$;
9: Add all species seeds back into $L_d$;
10: while $L_d.size() \neq maxSize$ do
11:     Randomly create a new solution $\acute{x}$;
12:     Push $\acute{x}$ into $L_d$;
13: end while
```

There is no any further contribution of these redundant individuals to the search for better solutions. It is better to replace the redundant solutions with randomly generated individuals so that other sub-regions of the search space can be explored. The procedure of replacing redundant individuals in the species is summarized in Algorithm 12, where $L_d.size()$ returns the size of the list L_d (i.e., the number of elements in L_d), $L_d[i].fitness$ denotes the fitness of the i-th individual in L_d, and $L_d[i].seedfitness$ denotes the fitness of the species seed of the species, to which the i-th individual in L_d belongs.

6.2.2.2 Partition Based Multi-Population Approach

Recently, in [106], the authors have proposed a partition technique, which divides the current population into different sub-populations according to the fitness and distribution of individuals in the landscape. Each sub-population can ideally incorporate the basins of attraction of similar optima. The basic idea of the proposed partition approach is to distribute the best individuals away from each other (according to a certain distance measure) under different peaks in the search space.

Algorithm 13 The partition algorithm – Partition(p_t)

```
1: input: Current population $p_t$, reduction factor $\alpha \in (0, 1)$, and distance threshold $\epsilon$
2: $\mu \leftarrow$ Get-Potential-Attractors($p_t$, $\alpha$)
3: $F \leftarrow$ Get-Final-Attractors($\mu$, $\epsilon$)
4: $p_t \leftarrow$ Construct-Partitions($p_t$, $F$)
```

Algorithm 13 shows the pseudo-code of the partition approach. This partition approach has three main different sub-procedures, which are summarized as follows: The first sub-procedure takes two arguments p_t and α, where p_t is the current population and α represents a reduction factor, and returns a set of individuals showed as μ. The aim of this sub-procedure is to select good individuals from the current population as the potential attractors. The selection mechanism is based on the relative candidate fitness values. For example, the selected individuals have quit close (or the same) fitness values to the fitness of the best individual in the population. These individuals are incorporated in the set μ, if the following condition is satisfied:

$$\forall x \in p_t, x \in \mu \text{ if } f(x) \geq \alpha f_{max}(p_t) \tag{6.1}$$

where $f_{max}(p_t)$ and $f(x)$ represent the maximum fitness of the current population p_t and the fitness of individual x. The value of $\alpha = 0.9$ was used in [106].

Two parameters are passed in the second sub-procedure: one is μ, which is already explained in the above paragraph, and the other is ϵ, which is the distance threshold value. This sub-procedure is used to refine the set of those individuals which are contained in μ. The individuals that have the mutual distance (measured in the Hamming distance) greater than a threshold value ϵ are moved into the set F. It works by trying to locate all pairs of individuals in μ that are closer than the predetermined distance threshold ϵ. If such a pair (x, y) exists and suppose x has a higher fitness than y, then y is removed from μ. Finally, all the individuals left in

μ are moved to F. So, the set F will include those individuals that are adequately away from each other.

The last sub-procedure is used to determine sub-populations from the individuals in the set F. The total number of sub-populations is equal to the number of individuals contained in F, each of which is considered as a representative to construct a sub-population. So, suppose there are m individuals in F, then there will be m sub-populations constructed, which can be denoted as p_t^1, p_t^2, $\cdots$, p_t^m and are associated with the representatives $r_t^1, r_t^2, \cdots, r_t^m$ in F, respectively. The m sub-populations are constructed as follows. For each individual $x \in p_t$, if x is closest to the i-th representative r_t^i, then it is included in the i-th sub-population p_t^i.

From the above description of the partition approach, it can be seen that the individuals in a sub-population are near to each other regarding but individuals in different sub-populations are far away in the fitness landscape. This means that different sub-populations may be on different peaks. The main motivation behind this approach is to maintain the diversity in the whole population.

6.3 GAs with Multi-Population with Adaptive Mutation

6.3.1 Motivation

Exploration and exploitation are essential in GAs for optimizing multi-modal functions. Exploration has the ability to explore new regions of the solution space in search of the global optimum. On the other hand, exploitation is the capacity to converge to an optimum (local or global) after finding the region which is holding the optimum. The balance between these properties of GAs can be maintained by using the multi-population with an adaptive mutation approach.

Adaptive techniques are used to accelerate the convergence speed and avoid GAs from being trapped into local optima. The important contribution of adaptive mutation operators is to improve the performance of GAs. So far, adaptive mutation schemes are usually used to a single population in the evolution process of a GA, no matter steady state GA (i.e., a single individual is generated at each iteration) or generational GA (i.e., the whole population is replaced at each generation).

As mentioned before, the premature convergence due to the loss of diversity is a big challenge to GAs. The multi-population approach can be applied to maintain the diversity of the whole population of GAs and retain the convergence capacity of GAs. In this chapter, a multi-population with adaptive mutation approach is proposed for GAs to address multi-modal optimization problems. Basically, an adaptive mutation operator is combined with the two aforementioned multi-population approaches: one is the species based multi-population approach in [70] and the other is the partition based multi-population approach in [106]. We use PWAM and SWAM to represent partition based multi-population with adaptive mutation and species based multi-population with adaptive mutation, respectively, in this study.

Generally speaking, GAs have two searching capacities: exploration and exploitation. But, the problem is how to balance these two capacities during the running process. In order to address this problem, researchers have used separately multi-population approaches and adaptive probabilities of genetic operators to increase the diversity during the evolution process. Our proposed scheme aims to combine both approaches for finding better solutions in multi-modal optimisation problems. This technique is applied to achieve a good balance between the explorative and exploitative capacities of GAs.

Algorithm 14 GA with the multi-population with adaptive mutation scheme

```
1: $t := 0$;
2: Randomly create an initial population $p_t$;
3: Evaluate the fitness of each individual in $p_t$;
4: $pop\_lst_t :=$ Partition($p_t$);
5: while $t < max\_gen$ do
6:     for $i := 0$ to $pop\_lst_t.size()$ do
7:         Statistics($pop\_lst_t^i$);
8:         Selection($pop\_lst_t^i$);
9:         Crossover($pop\_lst_t^i$);
10:        Mutation($pop\_lst_t^i$);
11:        Evaluation($pop\_lst_t^i$);
12:    end for
13:    Overlapping($pop\_lst_t$);
14:    Convergence($pop\_lst_t$);
15:    if $p_t.size() < min\_Inds$ then
16:        $p_{tmp}$ :=Re-initialization($numRemovedInds$);
17:        $new\_pop\_lst_t :=$ Partition($p_{tmp}$);
18:        Merge $pop\_lst_t$ and $new\_pop\_lst_t$ into $pop\_lst_t$
19:    end if
20:    $t := t + 1$;
21: end while
```

6.3.2 Framework of the Proposed GA

The framework of the GA with the proposed multi-population with adaptive mutation scheme is shown in Algorithm 14. This algorithm starts from a randomly generated population of individuals. Then, a list of sub-populations *pop_lst* are created using either the PWAM or SWAM scheme. When sub-populations have been generated, each sub-population $pop\_lst_t^i$ ($i = 0, \cdots, pop\_lst_t.size()$, where $pop\_lst_t.size()$ denotes the number of sub-populations in the list *pop_lst* at generation t) evolves independently just like in a classical GA except that a gene-level adaptive mutation scheme, i.e., the GBAM operator proposed in [84], is used. For each sub-population, we maintain two mutation probability vectors, which are updated using the statistics information from the sub-population and are used to control the bitwise mutation operation for each individual of the sub-population. At the end, after each sub-population has completed one generation, the overlapping and convergence check techniques are applied on each sub-population before starting the next generation.

Some of the tasks in this algorithm are similar to the classical GA. Here, we will focus on the main components of this approach, including partition, statistics, mutation, overlapping and convergence check procedures. These components are the key features of the proposed GA for multi-modal optimization problems. The detailed description of these components is given in the following sections.

6.3.2.1 Partition(p_t)

There are several approaches available in the literature to generate multi-populations. Two approaches, i.e., the species-based multi-population [70] and partition-based multi-population [106] schemes, are considered in this study. Detailed explanation of these techniques has been given in Section 6.2.2. Let p_t represent the current population at generation t, which is passed to the partition procedure as a parameter. The partition procedure will then use the SWAM or PWAM scheme to divide the current population into a list *pop_lst* of different sub-populations, which are expected to cover different areas in the fitness landscape. This approach can be helpful towards getting good results on multi-modal optimization problems.

6.3.2.2 Statistics

After the partition of the whole population into sub-populations, each sub-population evolves independently using a GA with the GBAM scheme. Each sub-population, say sub-population i, maintains two mutation probability vectors, denoted as $\vec{M}^{0i} = \{M_1^{0i}, M_2^{0i}, \cdots, M_l^{0i}\}$ and $\vec{M}^{1i} = \{M_1^{1i}, M_2^{1i}, \cdots, M_l^{1i}\}$, respectively, where l is the length of a binary-encoded solution. Each element of the vector $\vec{M}^{0i}$ denotes the mutation probability for a corresponding gene locus that has the allele 0, while each element of $\vec{M}^{1i}$ denotes the mutation probability for a corresponding gene locus that has the allele 1. Below, we describe the calculation of the mutation probability vectors for each sub-population.

In the statistics procedure, the mutation probabilities for each gene locus are updated using the feedback information for each current sub-population. Each vector contains a set of elements, which is associated to the probabilities of mutating a particular allele at each gene locus. Initially, each element of the vectors is set to an initial value within its boundaries.

For each generation, vectors are updated based on the fitness and distribution of solutions in a specific sub-population. It means that each sub-population alters its own vectors independently. The estimation of both vectors are updated using Eqs. (6.2) and (6.3).

$$M_j^{0i}(t+1) = \begin{cases} M_j^{0i}(t) + \gamma, & \text{if } \overline{G}_{jt}^{1i} > \overline{P}_t^i \\ M_j^{0i}(t) - \gamma, & \text{otherwise} \end{cases} \tag{6.2}$$

$$M_j^{1i}(t+1) = \begin{cases} M_j^{1i}(t) - \gamma, & \text{if } \overline{G}_{jt}^{1i} > \overline{P}_t^i \\ M_j^{1i}(t) + \gamma, & \text{otherwise} \end{cases} \tag{6.3}$$

where γ is the updated value of mutation probabilities of the vectors, $\overline{G}_{jt}^{1i}$ is the average fitness of individuals with allele "1" for locus j within sub-population i at generation t, and $\overline{P}_t^i$ is the average fitness of sub-population i at iteration t. The above update mechanism is used for each locus separately.

In summary, each gene locus j has two different mutation probabilities: M_j^{1i} is used for those individuals that have the value of 1 at gene locus j, and M_j^{0i} is used for those individuals that have the value of 0 at gene locus j. Initially, all mutation probabilities are assigned to a value, e.g, 0.02. The probabilities of M_j^{0i} and M_j^{1i}

are then automatically updated based on the feedback information from the search space, according to the relative success or failure of those chromosomes having a "1" or "0" at gene locus j for each generation. The new mutation probability for each locus j at generation $t+1$ is updated using Eqs. (6.2) and (6.3) in the case of a maximization problem.

6.3.2.3 Mutation(p_t^i)

This mutation procedure is performed on each sub-population i every iteration. It is distinctly employed on each sub-population using the associated vectors. Suppose, an offspring ($\acute{x} = \{\acute{x}_1, ..., \acute{x}_l\}$) is generated by mutating the parent $x = \{x_1, ..., x_l\}$ according to the corresponding vectors, which is one member of a sub-population. The mutation operation is performed on the whole solution gene by gene, which is mentioned in the Algorithm 15, where $Rnd(0,1)$ is a uniform random number generated within the range $[0,1]$.

Algorithm 15 Adaptive mutation

```
1:  for $j := 0$ to l do
2:     if $x[j] = 0$ then
3:        if $Rnd(0,1) < M_j^{0i}$ then
4:           $\acute{x}[j] := 1$;
5:        end if
6:     else
7:        if $Rnd(0,1) < M_j^{1i}$ then
8:           $\acute{x}[j] := 0$;
9:        end if
10:    end if
11: end for
```

Algorithm 16 Overlapping check of sub-populations

```
 1: $numRemovedInds := 0$;
 2: for $i := 0$ to $pop\_lst.size()$ do
 3:     $b1$ is the best solution of $pop\_lst_i$;
 4:     for $j := i + 1$ to $pop\_lst.size() - 1$ do
 5:         $b2$ is the best solution of $pop\_lst_j$;
 6:         if $(distance(b1, b2) < min\_Dist)$ then
 7:             if ($b1$ is worse than $b2$) then
 8:                 $numRemovedInds := numRemovedInds + pop\_lst_i.size()$;
 9:                 delete $pop\_lst_i$;
10:             else
11:                 $numRemovedInds := numRemovedInds + pop\_lst_j.size()$;
12:                 delete $pop\_lst_j$ from $pop\_lst$;
13:             end if
14:         end if
15:     end for
16: end for
```

6.3.3 Overlapping and Convergence Check

After selection, crossover, and mutation operations, all the sub-populations are checked regarding overlapping and convergence. Normally, the overlapping check between two sub-populations can be carried out by calculating the distance of the best solutions of the two sub-populations. If the distance is less than a threshold value min_Dist, then the two sub-populations are considered as overlapped. In this case, the sub-population, of which the fitness of the best solution is worse than that of another sub-population, is removed from the list pop_lst, and the number of individuals in the removed sub-population is recorded and accumulated into the variable $numRemovedInds$. Algorithm 16 shows the overlapping check of sub-populations.

After overlapping check, all sub-populations will undergo the convergence check process, which is performed to see whether a sub-population has converged or not. If the number of similar solutions of a sub-population is greater than a threshold value Lmt_conv (The threshold value is set to 0.95 times the size of the sub-population), the sub-population is considered to be converged. If a sub-population is converged,

Algorithm 17 Convergence check of sub-populations

```
1: for each sub-population $s \in pop\_lst$ do
2:     if $totSimilInds > (0.95 * s.size())$ then
3:         $numRemovedInds := numRemovedInds + s.size();$
4:         Delete $s$ from $pop\_lst$;
5:     end if
6: end for
```

then it is removed from *pop_lst*. The convergence check of sub-populations is shown in Algorithm 17.

After the overlapping and convergence checks, we check whether the number of individuals that are still active in the whole population is less than a predefined minimum threshold *min_Inds*. If so, the following actions (see Lines 15-19 in Algorithm 14) will be taken. First, a temporary population of *numRemovedInds* random individuals is created; Then, the temporary population is partitioned into a list *new_pop_lst* of sub-populations using one of the SWAM and PWAM partition methods; Finally, this list of sub-populations is merged with the current list *pop_lst* of sub-populations.

In summary, the proposed GA with the multi-population with adaptive mutation approach has the following characteristics:

1. It maintains two mutation probability vectors for each sub-population. Each gene locus has two different mutation probabilities and different loci have different mutation probabilities.

2. The mutation probabilities for each locus are updated iteration by iteration according to the current state of evolution.

3. The multi-population method is expected to locate multiple optima of multi-modal optimization problems, and the adaptive mutation scheme aims to speed up the convergence toward the optima covered by a sub-population.

6.4 Experimental Study

In this section, three different groups of experiments were conducted on well-known decision problems, which have been evaluated in the area of maximum satisfiability (MAX-SAT) problems. Boolean satisfiability problems involve finding an assignment of variables that maximizes the number of satisfied clauses of given constraints. The general form of these constraints is presented in conjunctive normal form (CNF) or product-of-sum form. The MAX-SAT problem is also a well-known NP-hard optimization problem. In addition, boolean satisfiability expression has been applied to introduce an approach for generating problems with controllable degree of "multi-modality" and "epistasis". Spears in [94] introduced a scheme which creates epistasis problems of different sizes by generating specific boolean expressions which always have only one solution. These problems are similar to the multi-modal problems, but having only one solution makes the problem harder to solve.

In this section, we first introduce the mapping between GAs and MAX-SAT problems [94] for generating constraints with controllable degree of multi-modality and epistasis, and then investigate the experimental results.

6.4.1 Boolean Satisfiability and Genetic Algorithms

In order to employ GAs to solve any particular problem, we need to consider two critical components: 1) specifying the appropriate representation for the solution space, i.e., the search space; and 2) defining an external evaluation function which determines the quality of individuals, i.e., the fitness of individuals. MAX-SAT has a simple string representation which is highly compatible for GAs. Each individual is a binary string of length l, where the i-th bit determines the truth value of the i-th boolean variable shown in the boolean expression. A candidate solution is evaluated

TABLE 6.1: Fitness f for assignments to $a \wedge (a \vee \bar{b})$, taken from [86]

a	b	Fitness f
0	0	avg(0,max(0,(1-0))) = 0.5
0	1	avg(0,max(0,(1-1))) = 0.0
1	0	avg(1,max(1,(1-0))) = 1.0
1	1	avg(1,max(1,(1-1))) = 1.0

by the fitness function, which delegates a fitness value 1 to the individual (string) by whose boolean values the boolean expression is evaluated to 1, and 0 to other solutions.

In [94], Spear proposed another scheme that assigns a fitness to an individual's sub-expressions in the whole expression and aggregate them in some way to produce a complete individual fitness value. According to this scheme, a general and natural way is to specify the value of TRUE to be 1 and the value of FALSE to be 0 by Smith [86]. The fitness value $f(e_i)$ of a simple expressions e_i is defined as follows:

$$f(\bar{e}) = 1 - f(e)$$

$$f(e_1 \vee e_2 \vee ... \vee e_n) = max(f(e_1), f(e_2), ..., f(e_n))$$

$$f(e_1 \wedge e_2 \wedge ... \wedge e_n) = avg(f(e_1), f(e_2), ..., f(e_n))$$

where *avg* represents the average fitness of its parameters. In [86], the author suggested an evaluation function, which gives reward to an intermediate solution. For example, the fitness values of solutions to the boolean expression $(a \wedge (a \vee \bar{b}))$ are determined according to Table 6.1. As shown in Table 6.1, half of the solutions are correct (lines 3 and 4) and are assigned a fitness 1. For the other half of incorrect solutions (lines 1 and 2), one of them which satisfies half of the conjunction (line 1) is achieved a better fitness value.

6.4.2 Generating Multi-Modal Problems

In order to explain the above consideration regarding the mapping between GAs and boolean satisfiability problems. The mapping should have the ability to express any satisfiability problem into a function optimization problem. Spear [94] proposed boolean satisfiability expressions, which can also be applied to control the difficulty of the problem under study by modifying the degree of multi-modality and epistasis of the search space.

The MAX-SAT has been used to create multi-modal problems, which is a straightforward process. Here, we are going to consider the example proposed by [94]. In this example, an expression is based on 30 boolean variables, which may contain varying number of peaks.

A uni-modal problem can be defined using an expression as follows:

$$1Peak \equiv (x_1 \wedge x_2 \wedge ... \wedge x_{30})$$

This expression is completely true only when all variables are true. In addition, the fitness of a solution in the search space is simply the number of true variables (divided by 30) in the expression. Thus, we can say that there is a unique peak in the fitness landscape.

Now, a bi-modal problem can be determined with slightly modification to the one-peak problem as follows:

$$2Peak \equiv 1Peak \vee (\bar{x}_1 \wedge \bar{x}_2 \wedge ... \wedge \bar{x}_{30})$$

The above problem consists of two peaks. Before creating a new peak, all variables are true of the previous peak, then the new peak would occur where all variables are false. The fitness values of both peaks are 1.0 and both are global optima in this context. It is possible to consider the problem to be more challenging.

$$2Peak \equiv 1Peak \vee (x_1 \wedge \bar{x}_1 \wedge \bar{x}_2 \wedge ... \wedge \bar{x}_{30})$$

Since it is impossible for a single boolean variable x_1 to be both true and false at the same time, the second disjunct cannot be true (satisfied). However, if the rest of the 29 variables are 0, the disjunct will have a fitness value a little less than 1.0. It is often pertained as a false peak. It is very difficult to locate the global optimum due to the increasing number of false peaks. This way, the problem becomes more challenging for GAs to solve. The following expressions are more suitable examples of the problems with many number of false peaks:

$$3Peak \equiv 2Peak \vee (x_1 \wedge \bar{x}_1 \wedge \bar{x}_2 \wedge ... \wedge \bar{x}_{15} \wedge x_{16} \wedge ... \wedge \bar{x}_{30})$$

$$4Peak \equiv 3Peak \vee (x_1 \wedge \bar{x}_1 \wedge x_2 \wedge ... \wedge x_{15} \wedge \bar{x}_{16} \wedge ... \wedge x_{30})$$

$$5Peak \equiv 4Peak \vee (x_1 \wedge \bar{x}_1 \wedge x_2 \wedge \bar{x}_3 \wedge x_4 \wedge \bar{x}_5 ... \wedge \bar{x}_{29} \wedge ... \wedge x_{30})$$

$$6Peak \equiv 5Peak \vee (x_1 \wedge \bar{x}_1 \wedge \bar{x}_2 \wedge x_3 \wedge \bar{x}_4 \wedge x_5 ... \wedge x_{29} \wedge ... \wedge \bar{x}_{30})$$

6.4.3 Peer Algorithms for Comparing the Proposed GA

The performance of proposed algorithm is compared with some existing GAs, which are taken from the literature. These algorithms were tested on different multi-modal boolean satisfiability problems, which are widely applied in the literature. Initially, our algorithm has been compared to standard GAs with fixed mutation and crossover probabilities. In addition, the proposed technique is compared with some other parameter control approaches, such as adaptive [96], self-adaptive [11] and SSRGA [106] schemes. These compared approaches are briefly described as follows.

The simple GA (SGA) is a standard binary-encoded GA with each solution represented as a bit string of length l. Mutation and crossover operators are used in the SGA as variation operators. The probabilities of these variation operators are initially assigned in our experiments and are then fixed during the whole evolution process.

Srinivas and Patnaik introduced an adaptive genetic algorithm (AGA) [96], which is an efficient adaptive algorithm for multi-modal optimization problems. The main motivation of this approach is to maintain the diversity in the population and retain the convergence capacity of the GA by using adaptive probabilities of crossover and mutation. The rates of mutation and crossover are updated depending on the fitness values of obtained solutions. These adaptive mutation and crossover operators are helpful to preserve good solutions of the population. The mutation and crossover rates are updated according to the following equations.

$$p_c = \begin{cases} k_1(f_{max} - \acute{f})/(f_{max} - \bar{f}), & \text{if } \acute{f} \geq \bar{f} \\ k_3, & \text{otherwise} \end{cases} \tag{6.4}$$

$$p_m = \begin{cases} k_2(f_{max} - f)/(f_{max} - \bar{f}), & \text{if } f \geq \bar{f} \\ k_4, & \text{otherwise} \end{cases} \tag{6.5}$$

where f_{max} and $\bar{f}$ represent the maximum fitness and the average fitness of the population, respectively, $\acute{f}$ is the larger of the fitness values of the individuals to be crossed and f is the fitness of the solution which is being mutated. k_1, k_2, k_3, $k_4 \in$ [0,1] are predefined constants. These predefined constant values are taken from [96] so that the same values are considered in our experiments.

A self-adaptive GA (SAGA) was developed by Back and Schutz [11], which uses a single mutation rate separately per solution. For example, an individual consists of a binary string of length l and an associated mutation rate $\mu \in [0, 1]$. The new mutation probability is adjusted according to the following equation.

$$\acute{\mu} = 1 + \frac{1-\mu}{\mu} exp(\gamma N(0,1)) \tag{6.6}$$

where $\gamma = 0.22$ is the learning rate and $N(0, 1)$ is the uniform distributed value with mean 0.0 and standard deviation 1.0. A new solution is generated through bit-wise mutation of l bits applying the mutated mutation rate value $\acute{\mu}$. The mutation rate is not less than $1/l$.

Vafaee and Peter recently proposed a site-specific rate GA (SSRGA) [106], which introduces a mutation scheme to determine different mutation rates for different sites of individuals. The main motivation of this approach is to face both explorative and exploitative responsibilities of variation operators. This scheme starts from partitioning the population into a number of sub-populations based on the fitness and distribution of solutions contained in the search space, as discussed in Section 6.2.2.2. Then, the result of this procedure is a probability vector (SSR vector) with each element associated to the rate of mutating a specific allele at each locus of every solution of a sub-population.

6.4.4 Experimental Setting

In order to investigate our proposed algorithms in comparison with four other algorithms on problems with different levels of difficulty, we modified the length l of individuals and the population size n to change the difficulty of problems. Generally, the problem becomes more difficult if the length of individuals increases.

We have considered different levels of problem difficulty by setting the parameter pair (l, n) to (20, 30), (30, 50), (50, 50), and (60, 70), respectively. These parameter settings were used to multi-modal landscape with different degrees of multi-modality by assigning the number of peaks to the values in 1, 5, 10. The number of peaks (or degree of multi-modality) can be controlled by the experimenter. Each peak is determined by a randomly created solution of length l, which identifies the specific position of the peak in the search space. When a solution is to be evaluated, we first identify the nearest peak (in terms of Hamming space) to the solution. Then, the fitness of the solution is calculated by dividing the common number of bits between the solution and the nearest peak by the total length l of a solution. Formally, the fitness of an arbitrary string j, denoted by f_j, is calculated by the following equation:

$$f_j = \frac{1}{l}\, max_{i=1}^{p}\{l - HammingDistance(j, peak_i)\} \tag{6.7}$$

where p is the total number of peaks in the fitness landscape.

The default setting of parameters applied in the experiments can be observed in the original papers [11, 96, 106]. To compare the performance of the multi-population with adaptive mutation approaches and other algorithms described in Section 6.4.3, all algorithms were allowed the same population size, same length of individuals, and same maximum number of generations for each run.

The performance of investigated approaches depends on the operators and parameters applied. In this study, the key parameters are considered for sensitivity analysis, such as the population size, the length of individuals, and the radius (min_dist). It is not always a good idea to locate multiple optima by using a small population size. Increasing the population size, however, reduces the probability of being trapped into local optimum and increasing the probability of locating good solutions during the evolution process. A solution consists of a l-bit binary string. If the length of string increases, the problem becomes more challenging since the state space is exponentially expanded according to the length of solutions.

The parameter r (*min_dist*) is used to determine the species and species seed in the SPSO [70] and to identify the representative individual from the population in PWAM [106]. The radius (r) is the key parameter to both approaches. If the radius is too small, there is a potential problem that few isolated solutions species may tarp into local optimum due to very small size of radius. In this context, due to the loss of diversity, the algorithms can not progress further during the process. If the radius is too large, there may be more than one optimum within the radius. It is very difficult to choose a suitable value of the radius. In this study, we have considered different values of the radius to investigate the performance of algorithms in order to determine that which parameter setting gives the great performance among the compared algorithms.

6.4.5 Experimental Results and Analysis

In order to investigate that the multi-population with adaptive mutation scheme can improve the performance of GAs regarding the solution quality and convergence rate for multi-modal problems, all algorithms were run independently 50 times on multi-modal problems with different degrees of multi-modalities. The parameter r is only associated with the top three algorithms in these tables because multi-population schemes are used in these algorithms. The parameter r is considered as a threshold value (i.e., the minimum distance between two solutions) in the PWAM and SSRGA algorithms. In the SWAM algorithm, species and species seed are constructed with the help of the parameter r.

The experimental results of algorithms on different problems with one peak, five peaks and ten peaks, are shown in Tables 6.2, 6.3 and 6.4, respectively, which give detailed results of the average and standard deviation of the best fitness values found during the evolution process. Both values are computed when the maximum number of iterations is completed or the global optimum is found. The smaller standard deviation means that an algorithm produces reliable solutions.

6.4.5.1 Effect of Varying the Length of Solutions and Population Size

The algorithms have been investigated on different problems with different levels of difficulty in terms of the combination of the length of individuals and the population size. The problem would become more difficult if the length of solutions is increased. The increasing size of population leads to less chance of stagnation of the EA on local optima and more chance to locate the better solution at the termination of the optimization process [94]. The proposed algorithms have been tested on different lengths of solution and population sizes.

6.4.5.2 Effect of Varying the Number of Peaks

The performance of PWAM and SWAM is equal on the problems with a specific setting of (l, n) and different number of peaks. The efficiency of $SSRGA$ slightly declines as the number of peaks increases. However, the situation of the rest of compared algorithms is reversed.

6.4.5.3 Effect of Varying the Parameter r or min_dist

Different settings of the parameter r, e.g., $0.5*l$, $0.6*l$, and $0.7*l$, are applied within the algorithms PWAM, SWAM, and SSRGA, where the performance of PWAM and SWAM is observed to be identical on these values of r. The performance of PWAM and SWAM may be affected by different values of r other than these values. From

Table 6.2: Comparison results of algorithms with different parameter settings on different problems with one peak

(l, n)		(20, 30)			(30, 50)			(50, 50)			(60, 70)		
r or *min_dist*		0.5*l	0.6*l	0.7*l	0.5*l	0.6*l	0.7*l	0.5*l	0.6*l	0.7*l	0.5*l	0.6*l	0.7*l
PWAM	avg	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	std	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SWAM	avg	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	std	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SSRGA	avg	1.0	1.0	1.0	0.998	1.0	0.998	0.922	0.916	0.921	0.891	0.893	0.897
	std	0.0	0.0	0.0	0.007	0.0	0.006	0.016	0.019	0.022	0.016	0.014	0.016
SAGA	avg		0.877			0.824			0.750			0.740	
	std		0.033			0.024			0.022			0.020	
AGA	avg		0.93			0.872			0.794			0.774	
	std		0.026			0.023			0.020			0.015	
SGA	avg		0.897			0.876			0.817			0.810	
	std		0.051			0.047			0.041			0.034	

Table 6.3: Comparison results of algorithms with different parameter settings on different problems with five peaks

(l, n)		(20, 30)			(30, 50)			(50, 50)			(60, 70)		
r or *min_dist*		0.5*l	0.6*l	0.7*l	0.5*l	0.6*l	0.7*l	0.5*l	0.6*l	0.7*l	0.5*l	0.6*l	0.7*l
PWAM	avg	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	std	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SWAM	avg	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	std	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SSRGA	avg	1.0	1.0	1.0	0.991	0.995	0.944	0.906	0.906	0.911	0.879	0.874	0.878
	std	0.0	0.0	0.0	0.014	0.011	0.012	0.017	0.017	0.016	0.026	0.026	0.024
SAGA	avg		0.909			0.850			0.779			0.760	
	std		0.022			0.021			0.018			0.012	
AGA	avg		0.943			0.878			0.802			0.779	
	std		0.026			0.022			0.017			0.014	
SGA	avg		0.905			0.886			0.806			0.807	
	std		0.038			0.039			0.034			0.028	

Table 6.4: Comparison results of algorithms with different parameter settings on different problems with ten peaks

(l, n)		(20, 30)			(30, 50)			(50, 50)			(60, 70)		
r or min_dist		0.5*l	0.6*l	0.7*l	0.5*l	0.6*l	0.7*l	0.5*l	0.6*l	0.7*l	0.5*l	0.6*l	0.7*l
PWAM	avg	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	std	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SWAM	avg	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	std	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SSRGA	avg	1.0	1.0	1.0	0.987	0.988	0.987	0.876	0.882	0.884	0.831	0.837	0.833
	std	0.0	0.0	0.0	0.017	0.015	0.016	0.040	0.027	0.030	0.031	0.030	0.034
SAGA	avg		0.923			0.866			0.788			0.774	
	std		0.030			0.021			0.018			0.021	
AGA	avg		0.952			0.890			0.810			0.791	
	std		0.030			0.021			0.020			0.019	
SGA	avg		0.920			0.893			0.820			0.805	
	std		0.036			0.032			0.0340			0.027	

Tables 6.2, 6.3, and 6.4, it is clear that different configurations of the parameter r affects the performance of the SSRGA algorithm.

6.4.5.4 Comparison Regarding the t-Test and Performance Results

The statistical test of comparing algorithms is carried out applying a two-tailed t-test with a 98 degree of freedom at a 0.05 level of significance. Table 6.5 presents the t-test results of comparing algorithms, where the result is shown as "$s+$", "$s-$", "+", "−", or "$\sim$" if the first algorithm in a pair is significantly better than, significantly worse than, insignificantly better than, insignificantly worse than, or equivalent to the second algorithm, respectively. The t-test results of comparing algorithms are shown on the problem with 10 peaks and *mini_dist* is chosen $0.6 * l$ with different levels of problem settings. From Table 6.5, it can be observed that both PWAM and SWAM approaches are statistically better than other compared schemes to locate the optimum value in the fitness landscape. Here, we present only the statistical results of compared techniques with different level of difficulty of different problems with 10 peaks.

According to the parameter setting of different level of multi-modal problems are divided into three categories such as simple problems, bit harder problems, and difficult problems. The performance of all algorithms varies on different categories. From Figures 6.1 and 6.2, it can be seen that the convergence speed of involved algorithms, which were tested on different level of difficulty in simple multi-modal problems.

In Figures 6.1 and 6.2, PWAM, SWAM, and SSRGA have a higher convergence rate than self-adaptive GA, adaptive GA, and Standard GA on simple problems. The overall convergence speed of PWAM and SWAM is initially better than SSRGA on

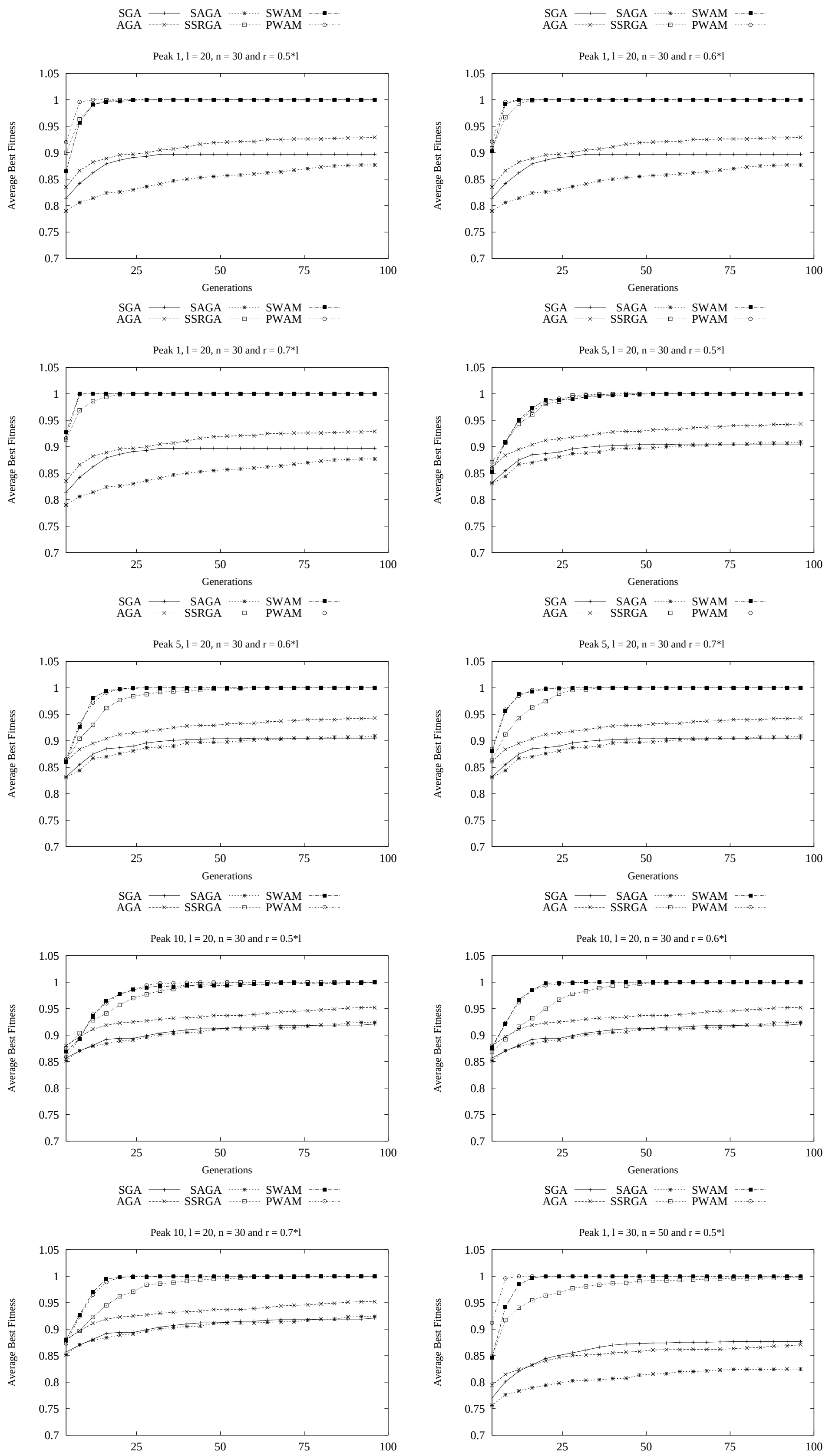


FIGURE 6.1: Evolutionary process of algorithms on simple problems.

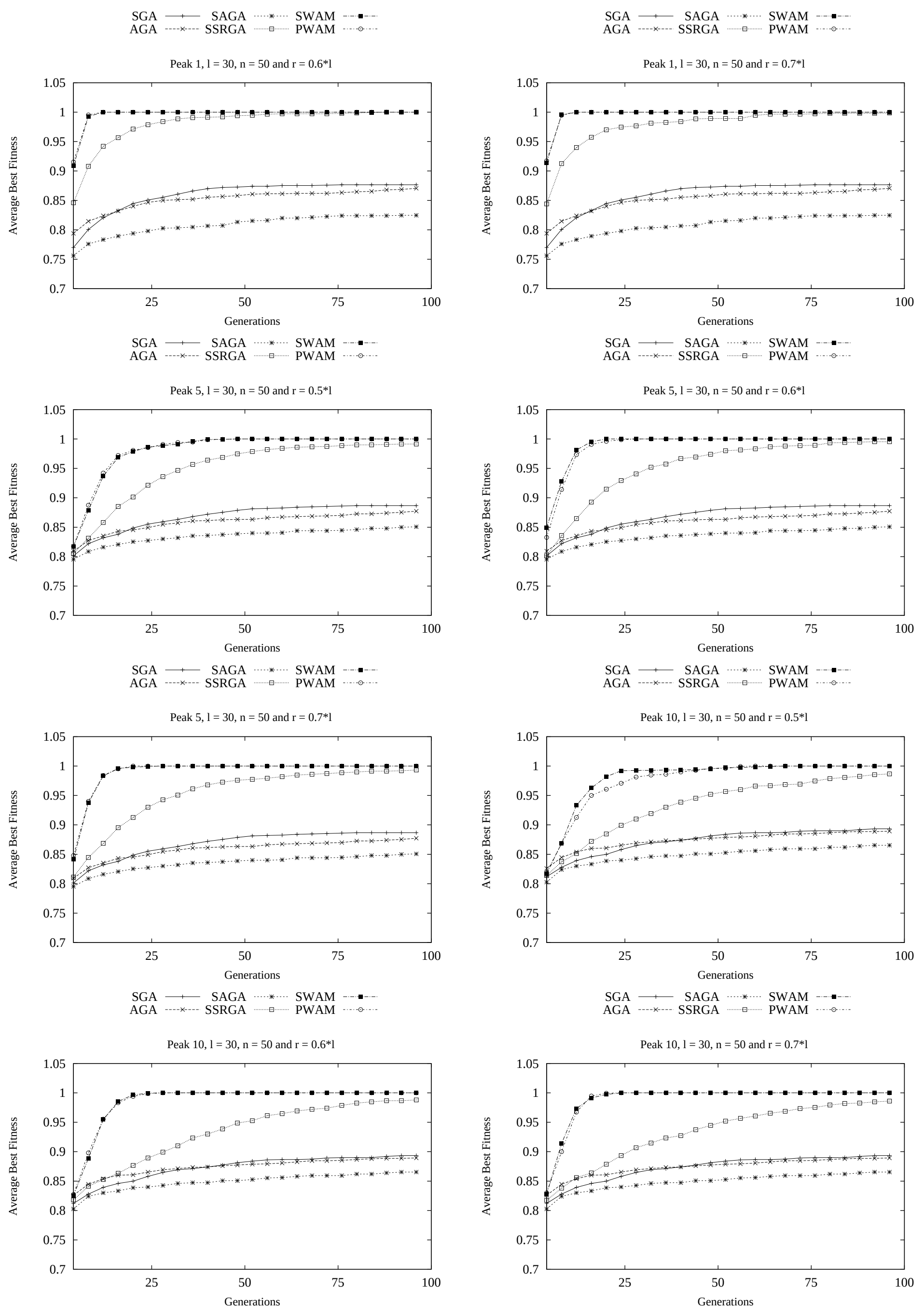


Figure 6.2: Evolutionary process of algorithms on simple problems.

Table 6.5: The t-test results of comparing algorithms on different problems with different levels of difficulty

Problems	$(20, 30)$	$(30, 50)$	$(50, 50)$	$(60, 70)$
PWAM-SWAM	$\sim$	$\sim$	$\sim$	$\sim$
PWAM-SSRGA	$\sim$	$\sim$	$s+$	$s+$
PWAM-SAGA	$s+$	$s+$	$s+$	$s+$
PWAM-AGA	$s+$	$s+$	$s+$	$s+$
PWAM-SGA	$s+$	$s+$	$s+$	$s+$
SWAM-SSRA	$\sim$	$\sim$	$s+$	$s+$
SWAM-SAGA	$s+$	$s+$	$s+$	$s+$
SWAM-AGA	$s+$	$s+$	$s+$	$s+$
SWAM-SGA	$s+$	$s+$	$s+$	$s+$
SSRGA-SAGA	$s+$	$s+$	$s+$	$s+$
SSRGA-AGA	$s+$	$s+$	$s+$	$s+$
SSRGA-SGA	$s+$	$s+$	$s+$	$s+$
SAGA-AGA	$s+$	$s-$	$s+$	$s-$
SAGA-SGA	$s+$	$s+$	$s-$	$s-$
AGA-SGA	$s+$	$s+$	$-$	$-$

simple problems but with time going the performance of all three algorithms are same except last two rows in Figures 6.1 and 6.2.

Figure 6.3 presents the results of the evolutionary process of compared algorithms on a bit harder problems. It can be clearly seen that the convergence speed of PWAM and SWAM is higher than other approaches. But, it is interesting to observe that different scheme have different convergence rate on different (bit harder) problems with different peaks. On a few problems, PWAM is initially better than SWAM regarding the convergence speed.

From Figure 6.4, it can be seen that the convergence rate of PWAM is faster than SWAM from the beginning to generation 50 on the first column, afterwards both schemes have the same convergence curve. In this figure, the convergence curves are quiet clear among the compared algorithms.

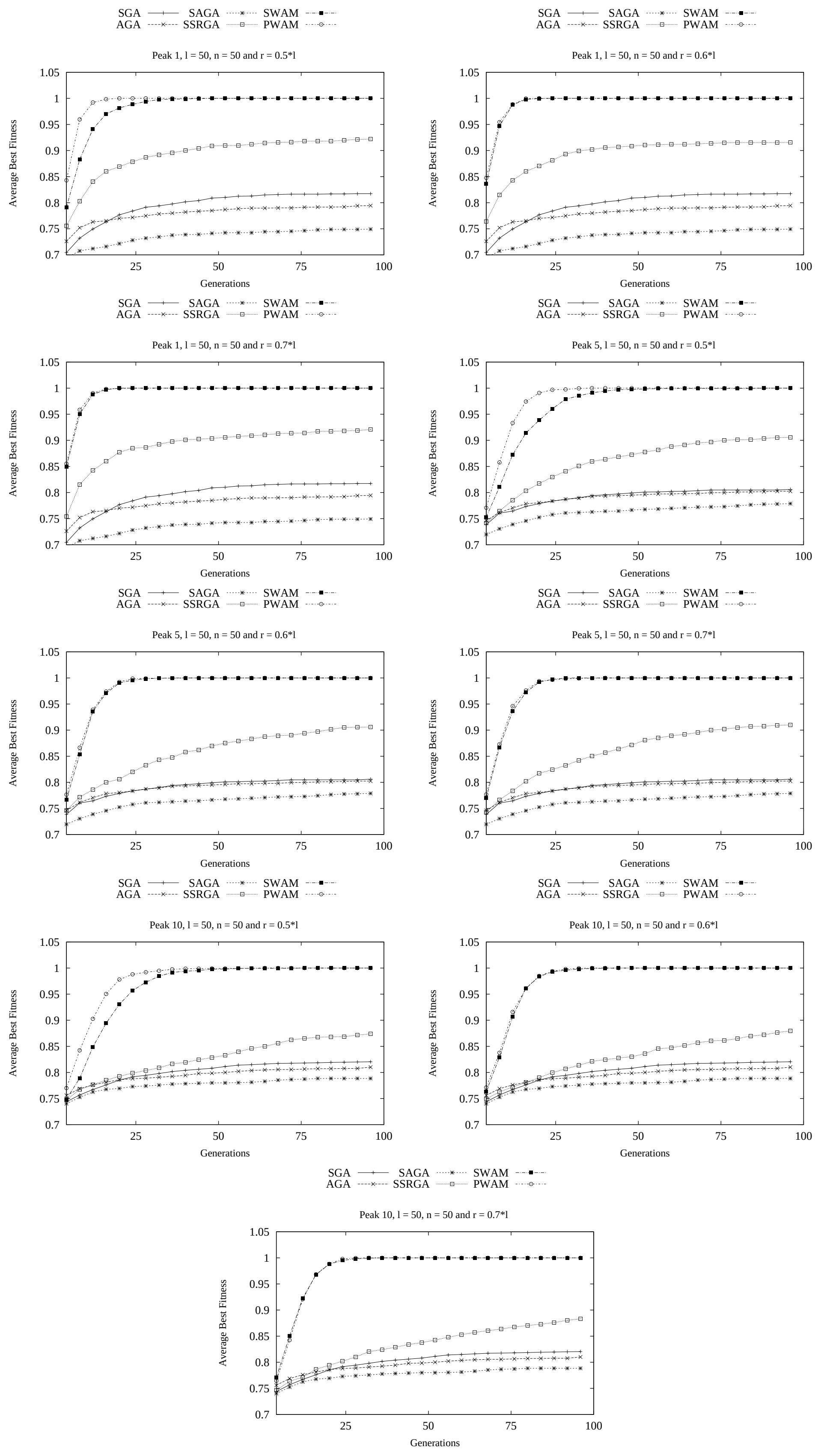


FIGURE 6.3: Evolutionary process of algorithms on bit harder problems.

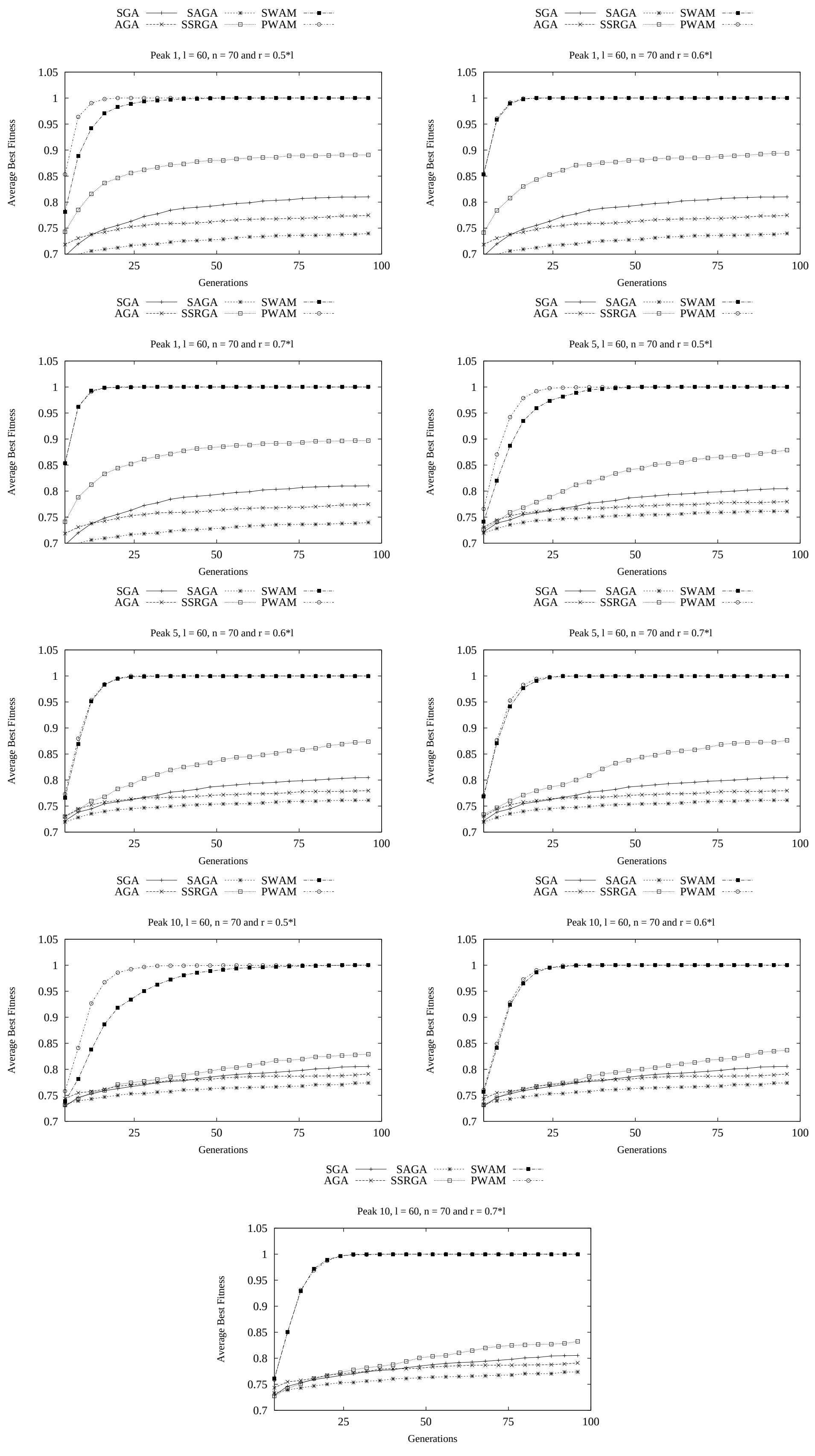


FIGURE 6.4: Evolutionary process of algorithms on difficult problems.

Let's analyze the convergence rate of the six compared algorithms. A very interesting thing is that the convergence speed of both PWAM and SWAM approaches is initially faster than other compared algorithms even on simple problems where SSRGA is going to compete to PWAM and SWAM. It can also be seen from figures that the proposed approach has two properties. The first is the quick convergence rate against other peer algorithms except on simple problems; PWAM and SWAM have quicker convergence speed than other compared approaches on different levels of problem difficulty by setting the parameters. The second property of suggested scheme is the ability to explore the prominent area by avoiding the premature convergence in the fitness landscape on multi-modality problems. The proposed technique reduces the probability of being trapped into local optima and increases the probability of locating the better results than other algorithms.

6.5 Chapter Summary

From the beginning, the community of GAs has done a lot of work to improve the performance and locate multiple optima in multi-modal optimization problems. Exploration and exploitation are two main characteristics that affect the performance of GAs for multi-modal optimization problems. Researchers have applied multi-population schemes to increase the efficiency of GAs in this context.

This chapter proposed a new approach for GAs to address multi-modal optimization problems, which is the combination of multi-population and adaptive mutation schemes. In the adaptive mutation scheme, two different mutation probability vectors are used for each sub-population. Each mutation probability vector contains elements associated to the probabilities of possessing a specific allele at each site. These vectors are updated during the evolution process independently for each sub-population. The suggested mutation scheme has been combined with two different multi-population techniques for GAs to address multi-modal optimization problems in this chapter.

In order to test the performance of our proposed GA, experimental studies were conducted on different benchmark problems to compare the results of our proposed GA with the results of a standard GA and some other approaches which belong to parameter control approaches. From the experimental results, it can be seen that our proposed approach greatly improves the performance of GAs regarding the location of multiple optima in the fitness landscape.

In the future, we will use the proposed GA with the multi-population with adaptive mutation scheme on large and complex real-world problems because GAs have a lot of applications in different area of real-world problems, such as science, engineering, business, and social science. In addition, it would be interesting to combine the clustering partition technique with adaptive mutation operators to improve the performance of GAs. Finally, we will conduct further experiments on epistatic [94] and deceptive [39] benchmark problems, which are similar to the multi-modal problems tested in this chapter.

Chapter 7

Conclusions and Future Work

Research is a never ending iterative process. The community of researchers keep testing their hypothesis, extending previous successful ideas, and investigate what other researchers have done in the same field. My work in this thesis is not an exception.

In general, the performance of evolutionary algorithms (EAs) depends on several factors, such as the choice of the population size, genetic operators, and many more. Selecting the right parameters and operators for EAs is a very hard task. There are two main points which motivate us to pursue the adaptation of genetic operators and relevant parameter values for EAs.

1. Parameter tuning: It is time consuming and difficult process, even if parameters are optimised one by one, regardless of their interactions. Therefore, reliable adaptation of parameter values and operators can improve the performance of EAs for optimization problems.
2. It is likely that for many optimization problems, the optimal parameter values or genetic operators may vary during the whole evolutionary process.

In the heuristic context, there is no single algorithm that will perform the best than all other algorithms. As Wolpert and Macready stated, "No search algorithm is superior on ALL problems" [115]. Due to this "No Free Lunch" theorem, one set of parameters can not be suitable for all problems.

In this thesis, we investigate adaptive mutation operators for EAs in order to tackle the problem of diversity loss and premature convergence to a sub-optimal solution for solving global optimization problems. From experimental results, it can be observed that the proposed ideas greatly improve the performance of EAs in the context of global optimization problems.

This chapter summarizes the technical contributions, describes conclusions based on the empirical results in this thesis, and discusses possible future research relevant to the work carried out in this thesis.

7.1 Technical Contributions

In this thesis, the following technical contributions have been made for the domain of EAs for global optimization problems.

- A novel approach that simultaneously uses several mutation operators for EAs to solve global optimization problems was proposed by considering various challenging issues (discussed in Section 1.1 of Chapter 1). This is one of the contributions in this thesis. This technique significantly improves the performance of the particle swarm optimization (PSO) algorithm for global optimization problems.

- In this thesis, we investigated several adaptive mutation operators, including population level adaptive mutation and gene level adaptive mutation operators, for genetic algorithms (GAs) and PSO for function optimization, which is the second contribution in this thesis. These adaptive algorithms were tested on different benchmark problems.

- The third contribution is the directed mutation (DM) idea for GAs to explore promising areas in the search space. There are some challenging issues in real coded mutation operators when mutation is used, e.g., how to determine the optimum mutation step size and how to get the suitable search direction towards the global optimum. The DM technique was proposed to address these issues in EAs.

- The multi-population with adaptive mutation idea is the fourth contribution in this thesis. The loss of diversity and premature convergence are the major drawbacks in EAs for multi-modal optimization problems. These issues are discussed in the thesis (Section 6.1 in Chapter 6). The idea of combining multi-population with adaptive mutation is used to enhance the performance of GAs on multi-modal optimization problems.

  The major technical contributions accomplished in this thesis towards the research domain of global optimization and multi-modal optimization are summarised as follows.

7.1.1 Adaptive Techniques Developed for Global Optimization

1. A new PSO algorithm based on adaptive mutation operator using self-choosing mutation operator is introduced for global optimization problems. Adaptive

mutation is applied to diversify the search direction and expedite the convergence speed of the algorithm to global optima. Three mutation operators are employed in this scheme. These mutation operators have different properties, which are used to guide the individual towards the global optima, explore new promising areas in the search space, and avoid the stagnation of PSO to local optima, respectively.

In order to find the potential solution for automatically choosing the appropriate mutation operator at the different level of the evolutionary process, an adaptive selection approach based on the assumption that the most suitable operator applied in the latest generations may also be successful in the coming various iterations is used. The selection ratios of all available mutation operators are equally initialized to 1/3 and are updated according to their performance. If the performance of an operator is relatively higher, then its selection ratio will increase. Gradually, the most suitable operator will be selected automatically for an individual and that operator will dominate almost all mutation behaviour according to its local fitness landscape at the corresponding evolutionary stage. By introducing the adaptive mechanism, PSO significantly improves its global search capability without losing its convergence property. However, different mutation operators give different performance on different test problems. The adaptive mutation operator exhibits balanced performance on all test problems.

2. We review several schemes that have been developed to modify the mutation probabilities based on statistical information of the population during the whole process. This approach provides a comparative analysis of various population level and gene level adaptive mutation operators for EAs for function optimization. The experimental results were presented and analyzed regarding the comparison of these operators on different benchmark optimization

problems.

3. A new DM approach was suggested to explore the promising areas in the search space. In this approach, the fitness and some other factors are considered to guide the individuals towards global optima. Different benchmark optimization problems were used as the test bed to verify the effectiveness of the DM approach. The idea of DM is comprised of fitness and percentage of individuals in each interval by using the statistical information from the current population. This information is used to shift the individuals around the search space via the DM. DM also has the capability to guide the individuals towards unseen areas of the search space by applying the feedback information. Some sort of neighbouring individuals to be mutated regarding the fitness and their distribution of those individuals, which exist in a specific interval. It is a modified version of the standard mutation.

   To show the effectiveness of the DM strategy, experiments were conducted to test the performance of traditional mutation GAs and proposed DM in two aspects: one is the comparison of the performance on different function optimization problems in terms of the *t*-test results; the other is the comparison of the performance with directed variation according to the same parameter setting. Based on the experimental analysis, it is clear that the DM approach has the property to guide the individuals towards the un-explored area of the search space and expedites individuals of the population towards local or global optima.

4. The multi-population with adaptive mutation approach is used to find multiple peaks and accelerate the search to global optima in the case of multi-modal optimization problems. This idea is the combination of multiple population and adaptive mutation schemes. A multi-population technique divides the whole search space into a number of sub-populations in order to maintain

these sub-populations on different promising peaks of the fitness landscape. These sub-populations are automatically created in the search space. For each sub-population, an adaptive mutation mechanism is considered to help GAs jump out of local optima and expediting the GAs towards promising regions in the search space.

In order to investigate the effect of the multi-population with adaptive mutation strategy, experiments were conducted to test the performance on different multi-modal Boolean satisfiability problems, which are popular problems in the literature. The experimental results of the proposed approach were also compared to standard GAs with random mutation probability and several other parameter control techniques such as adaptive, self-adaptive, and SS-RGA schemes on different multi-modal Boolean satisfiability problems.

7.2 Conclusions

It is evident from the investigation demonstrated in this thesis that the performance of EAs for optimization problems can be improved by using the fine-grained adaptation of mutation operator probabilities at the population level, individual level, and gene level. This research mainly focuses on the adaptation of mutation operators at the population level and gene level. The main objectives of this study have been to explore alterations to the traditional mutation operator for EAs. The adaptive approaches operate on the population level and gene level to enhance the diversity and avoid the premature convergence of EAs towards local optima.

In order to justify the algorithms which we have developed in this thesis, we conducted the experiments systematically and analysed the experimental results regarding the performance of these algorithms in comparison with several existing

EAs taken from the literature. Here, the major conclusions are given based on the experimental results and relevant analysis carried out in this thesis.

7.2.1 Adaptive Mutation Operators with GAs and PSO

The new PLAM_PSO operator diversifies the search directions and avoids the convergence of PSO to local optima. This strategy simultaneously uses several mutation operators in solving optimization problems. The selection ratio of each mutation operator is updated according to its relative performance. The results show that the adaptive scheme performs better than most algorithms that use a single mutation operator. It is very suitable to use for problems where the appropriate mutation operator is unknown.

We demonstrate various adaptive mutation operators, including the population level and gene level adaptive mutation operators, for GAs and PSO. The performance of several adaptive mutation operators varies on different benchmark problems. From the empirical results, it can be concluded that PLAM_PSO is greatly efficient on all test optimization problems with only some exceptions. GBAM_FAD, GBAM, and PLAM_GA algorithms are more efficient than PLAM_PSO on some functions.

The experimental results show that the gene level adaptive mutation operators are usually more efficient than the population level adaptive mutation operators for GAs. There is one issue with the gene level adaptive mutation operators. These operators take some time to calculate new mutation probabilities for each gene locus at each generation.

7.2.2 GAs with Directed Mutation Operator

The DM operator is an enhancement to the standard mutation for GAs to explore promising solutions in the search space. The statistical information regarding the fitness and distribution of solutions within the search space is achieved by DM. Such information is applied to migrate the individuals within the search space. DM also guides the solutions towards unseen areas of the fitness landscape by using the statistical information.

The effectiveness of GAs with DM on different benchmark optimization problems is presented via experiments. From the experimental results, it has been observed that the DM mechanism obtains good solutions.

7.2.3 Multi-population with Adaptive Mutation Operator

We analyzed that the multi-population with adaptive mutation approach works well on multi-modal optimization problems, which is also very effective to maintain multi-populations on different peaks to locate multiple optima for multi-modal optimization functions. This mechanism is applied to avoid the stagnation of GAs to local optima and expediting the convergence speed of GAs towards the better solutions in the search space. Exploitation and exploration are the two main properties in EAs for optimizing multi-modal functions. The balance between exploitation and exploration can be determined by using multi-population with adaptive mutation mechanism. Experimental results in Chapter 6 show that the suggested multi-population with adaptive mutation technique is effective in helping GAs to locate multiple optima for multi-modal problems.

7.3 Future Work

In this thesis, we presented different adaptive mutation schemes for EAs, which maintain the genetic diversity and also accelerate the convergence speed towards the global optimum. The effectiveness of these approaches are demonstrated for solving global optimization problems. However, there are some ideas that still need to be explored further in the future.

7.3.1 Adaptive Mutation Operators within GAs and PSO

The observation made in Chapter 4 regarding the adaptive mutation scheme for PSO showed that the operator that results in a higher relative performance, which is measured by a combination of the offspring fitness, current success ratio, and previous selection ratio, will have its selection ratio increased. The selection ratio of each operator is modified by using a single mechanism. In the future, we might also be able to find better schemes for updating the selection ratio of each operator. For example, if each operator is modified based on the average fitness improvement of all of the individuals affected by that operator, it may be possible to enhance the performance of GAs with adaptive mutation for global optimization problems.

In addition, we will investigate the performance of individual level adaptive mutation operators in comparison with population level and gene level adaptive mutation operators on different function optimization problems, and study how to reduce the complexity of gene level adaptive mutation operators in the future.

7.3.2 GAs with Directed Mutation Operator

The current research of DM has determined two objectives for real-coded GAs. The first one is to maintain the diversity of the population and the second is to accelerate the convergence speed for finding the potential individuals. There are still some issues to be considered for further research, e.g., how to adaptively adjust the number of intervals. This is an issue that needs to be explored further.

Adaptive tuning depends on not only the average fitness of intervals and the distribution of individuals in the population, but also automatically modifying the number of intervals, population size, selection schemes, and the number of operators (crossover or mutation). This could also be experimentally tested further for enhancing the optimization.

The basic idea of DM can be applied with the multi-population approach for maintaining the diversity of the population and enhancing the performance of the algorithm. For this, the fitness function can also be modified. The aim of this mechanism is to hold sub-populations at different peaks. It may be helpful to find the good solution or global optimum within a reasonable time. In addition, the performance of DM can be compared with some state-of-the-art algorithms, e.g., the covariance matrix adaptation evolution strategy (CMA-ES) [45].

7.3.3 Multi-population with Adaptive Mutation Operator

The multi-population with adaptive mutation approach has been used in GAs, which greatly improves the performance of GAs for multi-modal optimization problems. But, still some problems related to the divergence of algorithms and the premature convergence can be considered for further research. It is difficult for GAs to jump out form local optimum if the mutation probability and population variance are decreased too fast. A lot of work needs to be done to solve this problem.

Multi-population with adaptive mutation may not be useful in the situation when each sub-population contains a small number of individuals or a single solution. In this case, since the update mechanism of adaptive mutation is based on the feedback information of each current sub-population in the search space, there is no enough information achieved from the search space. This could result in almost similar mutation probabilities in a sub-population and the algorithm can be stuck into local optimum. This problem could be solved by setting the population properly or maintaining the population adaptively. This needs further experimentation.

When a sub-population becomes converged or overlapped in the proposed multi-population with adaptive mutation scheme, we just delete the converged or overlapped sub-population from the population. Hence, the entire population becomes smaller and smaller. But, if the size of the population is less than a predetermined threshold $MniInds$, then sub-populations are created after regenerating the corresponding number of randomly created individuals (these solutions may not work well since they may be attracted towards existing sub-populations). Finally, we merge all the sub-populations as a whole population. It is worth examining how to effectively generate individuals in the process of re-initialization to maintain the entire population size at the optimal value.

Bibliography

[1] http://www.biology.ed.ac.uk/research/groups/jdeacon/statistics/tress4a.html Student' t-test, 2011.

[2] http://www.statisticallysignificantconsulting.com/Statistics101.htm, Statistics Tutorial, 2011.

[3] http://www.experiment-resources.com/independent-one-sample-t-test.html, INDEPENDENT ONE-SAMPLE T-TEST, 2011.

[4] A. O. Adewumi, M. M. Ali, and J. O. Ayeni. A multi-level genetic algorithm for a multi-stage space allocation problem. *Mathematical and Computer Modelling*, 51: 109–126, 2010.

[5] P. J. Angeline. Adaptive and self-adaptive evolutionary computations. In *Computational Intelligence: A Dynamic Systems Perspective*, pages 152–163. IEEE Press, 1995.

[6] J. Arabas, Z. Michalewicz, and J. J. Mulawka. Gavaps - a genetic algorithm with varying population size. In *Proceedings of the 1994 International Conference on Evolutionary Computation*, pages 73–78, 1994.

[7] T. Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In R. Männer and B. Manderick, editors, *Proceedings of*

the 2nd International Conference on Parallel Problem Solving from Nature, pages 87–96, North-Holland, Amsterdam, 1992.

[8] T. Bäck. Self-adaptation in genetic algorithms. In *Proceedings of the 1st European Conference on Artificial Life*, pages 263–271. MIT Press, 1992.

[9] T. Bäck. Optimal mutation rates in genetic search. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 2–8. Morgan Kaufmann, 1993.

[10] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., Bristol, UK, 1st edition, 1997.

[11] T. Bäck and M. Schütz. Intelligent mutation rate control in canonical genetic algorithms. In *Proceedings of the 9th International Symposium on Foundations of Intelligent Systems*, pages 158–167, 1996.

[12] R. Bai. *An Investigation of Novel Approaches for Optimising Retail Shelf Space Allocation*. PhD thesis, School of Computer Science And Information Technology, The University of Nottingham, Nottingham, UK, September 2005. Online avialable http://etheses.nottingham.ac.uk/153/.

[13] J. Balczar, J. Diaz, and J. Gabarro. On Non- uniform Polynomial Space. In *Proceedings of the 1988 Conference on Structure in Complexity Theory*, pages 35–50, Berkeley, California, USA, 1988.

[14] S. Berlik and B. Reusch. Foundations of directed mutation. In *Proceeding of the 2006 International Conference on Integrated Intelligent Systems for Engineering Design*, pages 3–22, Amsterdam, The Netherlands, 2006. IOS Press.

[15] A. Berry and P. vamplew. Pod can mutate: A simple dynamic directed mutation approach for genetic algorithms. In *Proceedings of the 2nd International*

Conference on Artificial Intelligence in Science and Technology, pages 200–205, 2004.

[16] H. Beyer. *The Theory of Evolution Strategies*. Natural computing series. Springer, Berlin, 2001.

[17] L. B. Booker. Recombination distributions for genetic algorithms. In *Proceedings of the 2nd Workshop on Foundations of Genetic Algorithms*, pages 29–44, Vail, Colorado, USA, 1992. Morgan Kaufmann.

[18] R. Caruana and J. D. Schaffer. Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In *Proceedings of the 5th International Conference on Machine Learning*, pages 153–161, 1998.

[19] A. Charkaev. Course note on methods of optimization. online available at. Department of Mathematics, College of Science, The University of Utah, USA. `http://www.math.utah.edu/~cherk/teach/opt/2009.html`.

[20] J. W. Chinneck. *Practical Optimization: A Gentle Introduction*. Systems and Computer Engineering, Carleton University Ottawa, Canada, 2009. Online avialable `http://www.sce.carleton.ca/faculty/chinneck/po.html`.

[21] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 46–93, PWS Publishing Co., Boston, MA, USA, 1997.

[22] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In *Proceeding of the 1991 European Conference on Artificial Life*, pages 134–142, 1991.

[23] D. Corne, P. Ross, P. Ross, and H. lan Fang. Fast practical evolutionary timetabling. In *Proceedings of the 1994 Artificial Intelligence and the Simulation of Behaviour Workshop on Evolutioanry Computation*, pages 251–263, Springer Verlag, 1994.

[24] L. DaCosta, A. Fialho, M. Schoenauer, and M. Sebag. Adaptive operator selection with dynamic multi-armed bandits. In *Proceedings of the 2008 Genetic and Evolutionary Computation Conference, GECCO'08*, pages 913–920, New York, NY, USA, 2008. ACM.

[25] L. Davis. Adapting operator probabilities in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 61–69, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[26] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[27] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the 6th International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[28] A. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124 –141, jul 1999.

[29] A. E. Eiben, E. Marchiori, and V. A. Valk. Evolutionary algorithms with on-the-fly population size adjustment. In *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature, PPSN VIII*, LNCS 3242, pages 41–50, 2004. Springer.

[30] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith. Parameter control in evolutionary algorithms. In F. G. Lobo, C. F. Lima, and Z. Michalewicz,

editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 19–46. Springer, 2007.

[31] L. J. Eshelman and J. D. Schaffer. Productive recombination and propagating and preserving schemata. In *Proceedings of the 3rd Workshop on Foundations Of Genetic Algorithms*, pages 299–313, 1994.

[32] T. Fogarty. Varying the probability of mutation in the genetic algorithm. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 104–109, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[33] D. B. Fogel. *System Identification through Simulated Evolution: A Machine Learning Approach to Modeling.* Ginn Press, 1991.

[34] D. B. Fogel. *Evolving Artificial Intelligence.* PhD thesis, 1992.

[35] D. B. Fogel. Applying evolutionary programming to selected traveling salesman problems. *Cybernetics and Systems*, 24: pages 27–36, January 1993.

[36] L. G. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution.* Wiley, Chichester, UK, 1966.

[37] J.-P. Gagliardi, A. Ruiz, and J. Renaud. Space allocation and stock replenishment synchronization in a distribution center. *International Journal of Production Economics*, 115(1): pages 19–27, April 2008.

[38] D. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the 2nd International Conference on Genetic Algorithms and their Application*, pages 41–49, Hillsdale, NJ, USA, 1987.

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

[40] D. E. Goldberg. Sizing populations for serial and parallel genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 70–79, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[41] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Proceeding of the 1st Workshop on Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.

[42] J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16: 122–128, January 1986.

[43] C. Grosan and M. Oltean. Adaptive representation for single objective optimization. *Soft Comput.*, 9: 594–605, August 2005.

[44] P. Gwozdz and E. Szlachcic. An adaptive selection evolutionary algorithm for the capacitated vehicle routing problem. In *Proceedings of the 2nd International Symposium on Logistics and Industrial Informatics (LINDI 2009)*, pages 1–6, 2009.

[45] N. Hansen and A. Ostermeier. Convergence properties of evolution strategies with the derandomized covariance matrix adaptation: The $(\mu/\mu_I, \lambda)$-CMA-ES. In *Proceedings of the 5th European Congress on Intelligent Techniques and Soft Computing*, pages 650–654, 1997.

[46] A.-R. Hedar and M. Fukushima. Directed evolutionary programming: Towards an improved performance of evolutionary programming. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 1521–1528, 2006.

[47] J. Hesser and R. Männer. Towards an optimal mutation probability for genetic algorithms. In *Proceedings of the 1st International Conference on Parallel Problem Solving from Nature, PPSN I*, pages 23–32. Springer-Verlag, 1990.

[48] R. Hinterding, Z. Michalewicz, and T. C. Peachey. Self-adaptive genetic algorithm for numeric functions. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, PPSN IV, pages 420–429. Springer-Verlag, 1996.

[49] J. H. Holland. *Adaptation in natural and artificial systems.* PhD thesis, Ann Arbor, MI, USA, 1975.

[50] J. H. Holland. *Adaptation in natural and artificial systems.* MIT Press, Cambridge, MA, USA, 1992.

[51] T.-P. Hong, H.-S. Wang, and W.-C. Chen. Simultaneously applying multiple mutation operators in genetic algorithms. *Journal of Heuristics*, 6: 439–455, September 2000.

[52] X. Hu. PSO Tutorial. Self-Published. Online avialable: `http://www.swarmintelligence.org/tutorials.php`.

[53] C. Igel and M. Toussaint. Neutrality and self-adaptation. *Natural Computing*, 2(2): 117–132, 2003.

[54] K. A. D. Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[55] B. A. Julstrom. What have you done for me lately? adapting operator probabilities in a steady-state genetic algorithm. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 81–87, San Francisco, CA, USA, 1995.

[56] D. Karaboga. An idea based on honey bee swarm for numerical optimization. Technical report, Erciyes University, Engineering Faculty, Computer Engineering Department. Online avialable `http://mf.erciyes.edu.tr/abc/pub/tr06_2005.pdf`.

[57] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings 1995 IEEE International Conference on Neural Networks*, pages 1942 –1948, 1995.

[58] J. R. Koza. Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems. Technical report, Stanford, CA, USA, Tech: Report, 1990. Online avialable `http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.9961`.

[59] A. B. Levy. *The Basics of Practical Optimization.* Society for Industrial and Applied Mathematics, Market Street Philadelphia, USA, 2009. online available `http://coep.ufrj.br/~nero/Levy%20-%20The%20Basics%20of%20Practical%20Optimization.pdf`.

[60] C. Li, S. Yang, and I. Korejo. An adaptive mutation operator for particle swarm optimization. In *Proceedings of the 2008 UK Workshop on Computational Intelligence*, pages 165–170, 2008.

[61] F. Lobo and C. F. Lima. Adaptive population sizing schemes in genetic algorithms. In F. G. Lobo, C. F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 185–204. Springer, 2007.

[62] F. G. Lobo and C. F. Lima. Revisiting evolutionary algorithms with on-the-fly population size adjustment. In *Proceeding of the 2006 Genetic and Evolutionary Computation Conference, GECCO'06*, pages 1241–1248, 2006.

[63] S. Meyer-Nieberg and H.-G. Beyer. Self-adaptation in evolutionary algorithms. In F. G. Lobo, C. F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 47–75. Springer, 2007.

[64] B. L. Miller and D. E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4: 113–131, June 1996.

[65] H. Mühlenbein. How genetic algorithms really work i: Mutation and hillclimbing. In *Proceedings of the 2nd Conference on Parallel Problem Solving from Nature*, pages 15–25, 1992.

[66] G. Ochoa. Setting the mutation rate: Scope and limitations of the 1/l heuristic. In *Proceedings of the 2002 Genetic and Evolutionary Computation Conference, GECCO'02*, pages 495–502, 2002.

[67] G. Ochoa. Error thresholds in genetic algorithms. *Evolutionary Computation*, 14(2): 157–182, 2006.

[68] G. Ochoa, I. Harvey, and H. Buxton. Error thresholds and their relation to optimal mutation rates. In *Proceedings of the 5th European Conference on Artificial Life, ECAL '99*, pages 54–63, London, UK, 1999. Springer-Verlag.

[69] M. Oltean, C. Grosan, A. Abraham, and M. Köppen. Multiobjective optimization using adaptive pareto archived evolution strategy. In *Proceedings of the 5th International Conference on Intelligent System Design and Application*, pages 558–563, 2005.

[70] D. Parrott and X. Li. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Transactions on Evolutionary Computation*, 10(4): 440–458, August 2006.

[71] D. T. Pham and M. Casttellani. Adaptive selection routine for evolutionary algorithms. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 224(6): 623–633, September 2010.

[72] I. Rechenberg. Evolutions strategie - optimierung technischer systeme nach prinzipien der biologischen evolution, stuttgart: Frommann-holzboog. 1973.

[73] R. Ros Benchmarking the NEWUOA on the BBOB-2009 noisy testbed. In *Proceedings of the 2009 Genetic and Evolutionary Computation Conference, GECCO '09*, pages 2429–2434, Montreal, CA, USA, 2009.

[74] R. Rosenberg. Simulation of genetic populations with biochemical properties. Technical report, The Univresity of Michigan, College of Literature, Science, and The Department of Communication Science, 1967. online available http://deepblue.lib.umich.edu/handle/2027.42/7321.

[75] R. Salomon. Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions - a survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, 39: 263–278, 1995.

[76] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 51–60, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[77] J. D. Schaffer and A. Morishima. An adaptive crossover distribution mechanism for genetic algorithms. In *Proceedings of the 2nd International Conference on Genetic Algorithms and their Application*, pages 36–40, 1987.

[78] D. Schlierkamp-voosen and H. Ḿuhlenbein. Strategy adaptation by competing subpopulations. In *Proceedings of the 3rd International Conference on Parallel Problem Solving from Nature, PPSN III*, pages 199–208. Springer-Verlag, 1994.

[79] H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA, 1981.

[80] H. P. Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[81] M. Serpell and J. Smith. Self-adaptation of mutation operator and probability for permutation representations in genetic algorithms. *Evolutionary Computation*, 18: 491–514, September 2010.

[82] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pages 69–73, 1998.

[83] P. Siarry, A. Pétrowski, and M. Bessaou. A multipopulation genetic algorithm aimed at multimodal optimization. *Advances in Engineering Software*, 33: 207–213, April 2002.

[84] G. E. Sima Uyar, Sanem Sariel. A gene based adaptive mutation strategy for genetic algorithms. In *Proceedings of the 2004 International Conference on Genetic and evolutionary computation*, pages 271–281, 2004.

[85] S. N. Sivanandam and S. N. Deepa. *Introduction to Genetic Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2007.

[86] G. H. Smith. Adaptive genetic algorithms and the boolean satisfiability problem. Technical report, University of Pittsburgh, Pittsburgh, PA, 1979.

[87] J. Smith and T. C. Fogarty. An adaptive poly-parental recombination strategy. In *Selected Papers from AISB Workshop on Evolutionary Computing*, pages 48–61, London, UK, 1995. Springer-Verlag.

[88] J. Smith and T. C. Fogarty. Adaptively parameterised evolutionary systems: Self-adaptive recombination and mutation in a genetic algorithm. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature, PPSN IV*, pages 441–450, London, UK, 1996. Springer-Verlag.

[89] J. Smith and T. C. Fogarty. Recombination strategy adaptation via evolution of gene linkage. In *Proceedings of the 1996 International Conference on Evolutionary Computation*, pages 826–831, 1996.

[90] J. Smith and T. C. Fogarty. Self adaptation of mutation rates in a steady state genetic algorithm. In *Proceedings of the 1996 International Conference on Evolutionary Computation*, pages 318–323, 1996.

[91] J. Smith and T. C. Fogarty. Operator and parameter adaptation in genetic algorithms. *Soft Computing*, 1(2): 81–87, 1997.

[92] J. E. Smith. *Self-Adaptation in Evolutionary Algorithms for Combinatorial Optimisation*, volume 136 of *Studies in Computational Intelligence*. Springer, pages 31-51, 2008.

[93] W. M. Spears. Adapting crossover in evolutionary algorithms. In *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 367–384, 1995. MIT Press.

[94] W. M. Spears. *Evolutionary Algorithms: The Role of Mutation and Recombination.* Natural Computing Series. Springer, Secaucus, NJ, USA, 2000.

[95] W. M. Spears, K. A. D. Jong, T. Bäck, D. B. Fogel, and H. de Garis. An overview of evolutionary computation. In *Proceedings of the European Conference on Machine Learning*, pages 442–459, London, UK, 1993. Springer-Verlag.

[96] M. Srinivas and L. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 24(4): 656–667, April 1994.

[97] P. Suganthan, N. Hansen, J. Liang, K. Deb, Y. Chen, A. Auger, and S. Tiwari. Problem definitions and evoluation criteria for the cec 2005 speicial session on real-parameter optimization. Technical report, Nanyang Technical University, Singapore and Indian Institute of Technology, Kanpur, India, 2005.

[98] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998. `http://www.cs.ualberta.ca/%7Esutton/book/ebook/the-book.html`.

[99] L. Temby, P. Vamplew, and A. Berry. Accelerating real valued genetic algorithms using mutation-with-momentum. In *Proceedings of the 2005 Australian Conference on Artificial Intelligence*, pages 1108–1111, 2005.

[100] M. Thathachar and P. Sastry. A class of rapidly converging algorithms for learning automata. *IEEE Transactions on Systems, Man and Cybernetics*, 15: 168–175, 1985.

[101] D. Thierens. An adaptive pursuit strategy for allocating operator probabilities. In *Proceedings of the 2005 Genetic and Evolutionary Computation Conference, GECCO '05*, pages 1539–1546, New York, NY, USA, 2005. ACM.

[102] D. Thierens. Adaptive strategies for operator allocation. In F. G. Lobo, C. F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary*

Algorithms, volume 54 of *Studies in Computational Intelligence*, pages 77–90. Springer, 2007.

[103] A. Tuson and P. Ross. Adapting operator settings in genetic algorithms. *Evolutionary Computation*, 6: 161–184, June 1998.

[104] R. K. Ursem. Multinational evolutionary algorithms. In *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, pages 1633–1640. IEEE Press, 1999.

[105] F. Vafaee and P. C. Nelson. A genetic algorithm that incorporates an adaptive mutation based on an evolutionary model. In *Proceedings of the 2009 International Conference on Machine Learning and Applications*, pages 101–107, Washington, DC, USA, 2009. IEEE Computer Society.

[106] F. Vafaee and P. C. Nelson. An explorative and exploitative mutation scheme. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

[107] F. Vafaee, P. C. Nelson, C. Zhou, and W. Xiao. Dynamic adaptation of genetic operators' probabilities. In *Proceedings of the Nature Inspired Cooperative Strategies for Optimization*, pages 159–168. 2007.

[108] F. Vafaee, W. Xiao, P. C. Nelson, and C. Zhou. Adaptively evolving probabilities of genetic operators. In *Proceedings of the 7th International Conference on Machine Learning and Applications*, pages 292–299. IEEE Computer Society, 2008.

[109] F. van den Bergh. *An analysis of particle swarm optimizers.* PhD thesis, Department of Computer Science, University of Pretoria, South Africa, 2002.

[110] N. Wagner and Z. Michalewicz. Parameter adaptation for GP forecasting applications. In F. G. Lobo, C. F. Lima, and Z. Michalewicz, editors, *Parameter*

Setting in Evolutionary Algorithms, volume 54 of *Studies in Computational Intelligence*, pages 295–309. Springer, 2007.

[111] T. Weise. *Global Optimization Algorithms – Theory and Application*. Self-Published, second edition, 2009. Online available http://www.it-weise.de/projects/book.pdf.

[112] T. Weise, M. Zapf, R. Chiong, and A. J. Nebro. *Why Is Optimization Difficult?*, volume 193 of *Studies in Computational Intelligence*. Springer, 2009.

[113] T. White and F. Oppacher. Adaptive crossover using automata. In *Proceedings of the 3rd International Conference on Parallel Problem Solving from Nature*, pages 229–238, 1994.

[114] D. Whitley and T. Starkweather. Genitor ii.: a distributed genetic algorithm. *Journal of Experimental and Theoretical Artificial Intelligence*, 2: 189–214, October 1990.

[115] D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1): 670–82, April 1997.

[116] S. Yang. Adaptive non-uniform crossover based on statistics for genetic algorithms. In *Proceedings of the 2002 Genetic and Evolutionary Computation Conference, GECCO '02*, pages 650–657, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[117] S. Yang. Adaptive non-uniform mutation based on statistics for genetic algorithms. In *Proceedings of the 2002 Genetic and Evolutionary Computation Conference, GECCO'02*, Part II, pages 490–495, Berlin, Heidelberg, 2002. Springer-Verlag.

[118] S. Yang. Statistics-based adaptive non-uniform mutation for genetic algorithms. In *Proceedings of the 2003 Genetic and Evolutionary Computation*

Conference, GECCO'03, Part II, pages 1618–1619, Berlin, Heidelberg, 2003. Springer-Verlag.

[119] S. Yang and C. Li. A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 14(6): 959–974, December 2010.

[120] S. Yang and S. Uyar. Adaptive mutation with fitness and allele distribution correlation for genetic algorithms. In *Proceedings of the 2006 ACM symposium on Applied Computing, SAC '06*, pages 940–944, New York, NY, USA, 2006. ACM.

[121] J. Yao, N. Kharma, and P. Grogono. A multi-population genetic algorithm for robust and fast ellipse detection. *Pattern Analysis and Applications*, 8: 149–162, September 2005.

[122] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2): 82–102, 1999.

[123] P.-Y. Yin, S.-S. Yu, P.-P. Wang, and Y.-T. Wang. Task allocation for maximizing reliability of a distributed system using hybrid particle swarm optimization. *Journal of Systems and Software*, 80: 724–735, May 2007.

[124] T.-L. Yu, K. Sastry, and D. E. Goldberg. Population sizing to go: Online adaptation using noise and substructural measurements. In F. G. Lobo, C. F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 205–223. Springer, 2007.

[125] Q. Zhou and Y. Liu. Directed variation in evolutionary strategies. *IEEE Transactions on Evolutionary Computation*, 7(4): 356–366, 2003.