An Integrated Evolutionary System for Solving Optimization Problems

Abu Saleh Shah Muhammad Barkat Ullah

Bachelor of Science in Computer Science and Engineering
Khulna University, Bangladesh

A thesis submitted in partial fulfillment of the requirements

for the degree of Doctor of Philosophy at the

School of Engineering and Information Technology

University of New South Wales

Australian Defence Force Academy

Abstract

Many real-world decision processes require solving optimization problems which may involve different types of constraints such as inequality and equality constraints. The hurdles in solving these Constrained Optimization Problems (COPs) arise from the challenge of searching a huge variable space in order to locate feasible points with acceptable solution quality. Over the last decades Evolutionary Algorithms (EAs) have brought a tremendous advancement in the area of computer science and optimization with their ability to solve various problems. However, EAs have inherent difficulty in dealing with constraints when solving COPs.

This thesis presents a new Agent-based Memetic Algorithm (AMA) for solving COPs, where the agents have the ability to independently select a suitable Life Span Learning Process (LSLP) from a set of LSLPs. Each agent represents a candidate solution of the optimization problem and tries to improve its solution through co-operation with other agents. Evolutionary operators consist of only crossover and one of the self-adaptively selected LSLPs. The performance of the proposed algorithm is tested on benchmark problems, and the experimental results show convincing performance.

The quality of individuals in the initial population influences the performance of evolutionary algorithms, especially when the feasible region of the constrained optimization problems is very tiny in comparison to the entire search space. This thesis proposes a method that improves the quality of randomly generated initial solutions by sacrificing very little in diversity of the population. The proposed Search Space Reduction Technique (SSRT) is tested using five different existing EAs, including AMA, by solving a number of state-of-the-art test problems and a real world case problem. The experimental results show SSRT improves the solution quality, and speeds up the performance of the algorithms.

The handling of equality constraints has long been a difficult issue for evolutionary

optimization methods, although several methods are available in the literature for handling functional constraints. In any optimization problems with equality constraints, to satisfy the condition of feasibility and optimality the solution points must lie on each and every equality constraint. This reduces the size of the feasible space and makes it difficult for EAs to locate feasible and optimal solutions. A new Equality Constraint Handling Technique (ECHT) is presented in this thesis, to enhance the performance of AMA in solving constrained optimization problems with equality constraints. The basic concept is to reach a point on the equality constraint from its current position by the selected individual solution and then explore on the constraint landscape. The technique is used as an agent learning process in AMA. The experimental results confirm the improved performance of the proposed algorithm.

This thesis also proposes a Modified Genetic Algorithm (MGA) for solving COPs with equality constraints. After achieving inspiring performance in AMA when dealing with equality constraints, the new technique is used in the design of MGA. The experimental results show that the proposed algorithm overcomes the limitations of GA in solving COPs with equality constraints, and provides good quality solutions.

Keywords

Agent-based Evolutionary Algorithms.

Agent-based Memetic Algorithm.

Constrained Optimization Problems.

Constraint Handling Technique.

Evolutionary Algorithms.

Genetic Algorithms.

Multi-agent Systems.

Memetic Algorithms.

Acknowledgement

No words of gratitude are enough to thank my supervisor Associate Professor Ruhul A. Sarker (School of Engineering and Information Technology, UNSW@ADFA), whose untiring support, motivation and supervision made this thesis, see the light of the day. He always gives me timely and constructive feedback and thoughtful suggestions for my research ideas, paper and thesis drafts. I am particularly thankful for his patience and encouragement. His guidance has been indispensable in my academic, professional, and personal development. I feel lucky and honored to work with him.

I am indebted to Dr. Chris Lokan (School of Engineering and Information Technology, UNSW@ADFA) my co-supervisor. His critical advice and comments have taught me the balance between careful research and exploratory investigation. I must give him very special thanks for his deep concentration and enormous effort on my thesis writing.

I also like to thank my external co-supervisors, Dr. David Cornforth (CSIRO Energy Technology, Australia) for his guidance, enormous encouragements throughout my PhD candidature.

Thanks to the School of Engineering and Information Technology, all its administrative staff and IT support group who provided me with all the required facilities and took care of my needs during my stay at the school.

It is my honor to thank the University of New South Wales at the Australian Defence Force Academy (UNSW@ADFA) for providing me the University College Postgraduate Research Scholarship (UCPRS) to carry out this research.

I am also thankful to UNSW for awarding me Postgraduate Research Student Support Scholarship (for the year 2007 and 2008), ACM/GECCO for ACM / GECCO Student Travel Grant 2008, IEEE Computational Intelligence for Student Travel Grant

2007, and School of Engineering and Information Technology, UNSW@ADFA for sponsoring my different conferences expanses.

I am also thankful to Prof. Kalyanmoy Deb (IIT, Kanpur) and Dr. Tapabrata Ray (UNSW@ADFA) for their guidance, suggestions and providing their algorithms which helped a lot during the initial experimentations.

All praises are due to Allah the almighty God, the most Beneficent, and the most Merciful. I thank Him for bestowing His Blessings upon me to accomplish this task.

I would like to express gratitude to all my fellows in the campus. Particularly, I would like to thank Mr. S. M. Kamrul Hasan, Mr. Ehab Zaky Elfeky, Mr. Ziauddin Ursani, Mr. Mizanur Rahman, Mr. Shafiul Azam, and Mrs. Nafisa Tarannum for sharing their time and extending their friendship to me.

I express my gratitude to all of my teachers who showed me divine light of knowledge. Special thanks to my school teacher Mr. Israil who helped me to believe in myself.

I owe a debt of gratitude to my wife Nusrat Jahan. Her support and encouragement have been unfailing through the highs and lows of last two years. I thank my son Raghib Barkat for being born two weeks earlier than his due date of confinement, which allow me some more days to finalize the thesis.

Finally I would like to thank my parents (Abba and Amma) and my elder brother (Vaia) for their unwavering support, guidance, encouragement, and emphasis on the value of education that allowed me to get where I am today. I am forever grateful to them.

Dedication

To My Family

Abba (My Father Md. Hafizur Rahman)

Amma (My Mother Rumisa Begum)

Vaia (My Brother Maj. Saleh)

Vabi (My Sister in law Mouri)

My Wife (Nusrat Jahan Jhumu)

My Son (Raghib Barkat)

Originality Statement

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

Signed.....

Date.....

Copyright Statement

Authenticity Statement

I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis. No emendation of content has occurred and if there are any minor variations in formatting, they are the result of the conversion to digital format.

Signed.....

Date.....

List of Publications

Journal Articles

1. Barkat Ullah, A. S. S. M., Sarker, R., Cornforth, D. and Lokan, C. (2009). AMA: a new approach for solving constrained real-valued optimization problems. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, **13**(8): 741-762.
2. Barkat Ullah, A. S. S. M., Sarker, R. and Lokan, C (08/2009). Handling Equality Constraints with Agent-based Memetic Algorithms, *Memetic Computing Journal, Springer*, Under Review.
3. Barkat Ullah, A. S. S. M., Sarker, R. and Lokan, C. (10/2009). Evolutionary Optimization: Exploring Larger Search Space!, *Information Sciences*, Under Review.

Book Chapters

4. Barkat Ullah, A. S. S. M., Sarker, R. and Lokan, C. (2009). An Agent Based Evolutionary Approach for Nonlinear Optimization with Equality Constraints. In Sarker, R. and T. Ray, editor, *Agent-Based Evolutionary Search*, Springer. In Press.
5. Barkat Ullah, A. S. S. M., Sarker, R. and Cornforth, D. (2007). An Evolutionary Agent System for Mathematical Programming. In *Advances in Computation and Intelligence*, Lecture Notes in Computer Science, Vol. 4683, pp. 187-196. Springer.

Conference Articles

6. Barkat Ullah, A. S. S. M., Sarker, R. and Lokan, C. (2009). An Agent-based Memetic Algorithm (AMA) for nonlinear optimization with equality constraints. In *IEEE Congress on Evolutionary Computation, 2009. CEC '09*, Norway, pp. 70-77.

7. Barkat Ullah, A. S. S. M., Sarker, R. and Cornforth, D. (2008). Search space reduction technique for constrained optimization with tiny feasible space. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation, GECCO 2008*, Atlanta, GA, USA, ACM, pp. 881-888.

8. Barkat Ullah, A. S. S. M., Elfeky, E. Z., Cornforth, D., Essam, D. L. and Sarker, R. (2008). Improved evolutionary algorithms for solving constrained optimization problems with tiny feasible space. In *IEEE International Conference on Systems, Man and Cybernetics, SMC 2008,* Singapore, pp. 1426-1433.

9. Barkat Ullah, A. S. S. M., Sarker, R., Cornforth, D. and Lokan, C. (2007). An Agent-based Memetic Algorithm (AMA) for Solving Constrained Optimization Problems. In *IEEE Congress on Evolutionary Computation, CEC 2007*, Singapore, pp. 999-1006.

10. Barkat Ullah, A. S. S. M., Sarker, R. and Cornforth, D. (2007). A Combined MA-GA Approach for Solving Constrained Optimization Problems. In *6th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2007*, Australia, pp. 382-387.

Contents

List of Figures

List of Tables

List of Acronyms

AEAs	Agent-based Evolutionary Algorithms
AMA	Agent-based Memetic algorithm
AR	Allowable Range
COMOGA	Constrained Optimization by Multi-Objective Genetic Algorithms
COPs	Constrained Optimization Problems
CV	Constraint Violation
DR	Diversity Reduction
EAs	Evolutionary Algorithms
ECHT	Equality Constraint Handling Technique
EP	Evolutionary Programming
ES	Evolutionary Strategy
FGA	Force-based Genetic Algorithm
GAs	Genetic Algorithms
GP	Goal Programming
HELPR	Hybrid Evolutionary Learning for Pattern Recognition
II	Improvement Index
IP	Integer Programming
JSP	Job-shop scheduling problem
LP	Linear Programming
LS	Local Search
LSLP	Life Span Learning Process
MAs	Memetic Algorithms
MAS	Multi-agent Systems
MGA	Modified Genetic Algorithm
MILP	Mixed Integer Linear Program
MIP	Mixed Integer Program

MOGA	Multi-Objective Genetic Algorithms
NLP	Nonlinear Programming
NPGA	Niched-Pareto Genetic Algorithms
SA	Simulated Annealing
SBX	Simulated Binary Crossover
SGA	Simple Genetic Algorithm
SSRT	Search Space Reduction Technique
TCV	Total Constraint Violation
TS	Tabu Search
TSP	Traveling Salesman Problem
VEGA	Vector Evaluated Genetic Algorithms
VRP	Vehicle Routing Problem

Chapter 1

Introduction

1.1 Overview

Many real world decision processes require solving optimization problems. A good number of these problems can be represented as nonlinear programming models. These models consist of a nonlinear objective function that has to be optimized, while satisfying a number of linear and/or nonlinear constraints. The constraint type could be of equality, inequality or both. Due to their complex nature, many of these Constrained Optimization Problems (COPs) may not contain nice mathematical properties required by traditional solution techniques. Hence, conventional optimization algorithms are often unable to provide even a feasible solution (Sarimveis and Nikolakopoulos, 2005). For example, gradient based optimization techniques are only able to tackle mathematical models where properties such as continuity and convexity exist. Solving COPs with non-standard function properties has become an important research topic in computer science and operations research, due to the presence of high dimensionality, nonlinear parameter interaction, and multimodality of the objective function as well as due to the physical, geometric, and other limitations of different constraints (Liang and Suganthan, 2006).

Over the last few decades, Evolutionary Algorithms (EAs), as well as other bio-inspired heuristics, have proven themselves as efficient optimization techniques (Mezura-Montes, 2009). With the increasing recognition of the potential of EAs, Genetic Algorithms (GAs) (the most well-known branch of EAs) have also been used to solve a broad variety of problems in an extremely diverse array of fields, clearly

showing their power and potential for solving optimization problems (Marczyk, 2004; Sarker et al., 2003).

EAs may be able to overcome the above mentioned drawbacks of conventional optimization methods. However, in their initial versions, EAs were limited to only unconstrained problems (Mezura-Montes, 2009). Therefore, over the last decade, an extensive amount of research has been contributed to design and implement constraint-handling techniques (Coello, 2002). EAs are still criticized as they have insufficient ability to solve highly constrained and multi-modal problems efficiently. There is no guarantee of achieving the optimality, and the stability and efficiency of searches is low in those problems (Takahama and Sakai, 2009).

Recently, many hybridized algorithms have appeared in the literature to enhance the performance of EAs. Examples are memetic algorithms, where local search techniques are incorporated with EAs, and agent-based EAs that incorporate intelligent agent concepts with EAs.

Constrained optimization problems require efficient solution approaches for supporting quality decision making (Sarimveis and Nikolakopoulos, 2005). So, the main focus of this thesis is to design efficient algorithms and techniques to enhance the performance of EAs in solving constrained optimization problems.

1.2 Motivation and Scope of Research

This section briefly discusses the scope of research in solving constrained optimization problems in this thesis, and the motivation for carrying out this research.

The hurdles in solving constrained optimization problems arise from the challenge of searching a huge variable space in order to locate feasible points with an acceptable quality. It becomes even more challenging when the feasible space is very tiny compared to the search space. Over the last few decades, evolutionary algorithms have brought a tremendous advancement in the area of computer science and optimization, with their ability to solve various complex optimization problems. However, EAs have

inherent difficulty in dealing with constraints while solving the COPs. In spite of the presence of many different constraint handling techniques, EAs still suffer in solving COPs as stated by Takahama and Sakai (2009):

> *"While research on constrained optimization using evolutionary algorithms has been actively pursued, it has had to face the problem that the ability to solve multi-modal problems is insufficient, that the ability to solve problems with equality constraints is inadequate, and that the stability and efficiency of searches is low."*

To improve the performance of EAs, different hybridizations of algorithms have been introduced in recent times. Memetic Algorithms (MAs), a hybridized algorithm, can be considered as a marriage between the population-based global search and the heuristic-based Local Search (LS). The concept of MAs is inspired from the model of adaptation in natural systems, where an individual of a population may be improved through self learning along with the evolutionary adaptation of the population (Krasnogor and Smith, 2005; Moscato, 1989). Initially MAs were applied in combinatorial optimization problems (Alkan and Ozcan, 2003; Burke and Smith, 1999; Cheng and Gen, 1996; Merz and Freisleben, 1997; Merz and Freisleben, 2000; Tang *et al.*, 2005) and subsequently in continuous search spaces (Guimaraes *et al.*, 2006; Knowles and Corne, 2000; Molina *et al.*, 2005; Ong and Keane, 2004), and the performances were splendid. One of the critical issues regarding the performance of MAs is the selection of appropriate LS while hybridizing LS with Genetic Algorithms (GAs). If the selection of LS is not appropriate for a particular problem then MAs may not perform well; the performance may even be worse than GAs alone (Davis, 1991; Hart, 1994; Ong and Keane, 2004). Many types of local searches are available in the literature but it is very difficult to know which type is appropriate for a particular problem.

Several intelligent hybridized algorithms, such as agent-based evolutionary algorithms, have appeared in the literature recently, showing enhanced performance in solving optimization problems like unconstrained global optimization problems (Zhong *et al*., 2004), constraint satisfaction problems (Liu *et al*., 2006), and multi-objective

problems (Dobrowolski *et al.*, 2001; Siwik and Kisiel-Dorohinicki, 2006). However, good performance in solving constrained optimization problems with an agent-based evolutionary algorithm, to the best of our knowledge, has not been achieved in the literature. This motivates the design of an Agent-based Memetic Algorithm (AMA), which mitigates the shortcomings of MAs and solves COPs efficiently.

The quality of individuals in the initial population influences the performance of evolutionary algorithms, especially when the feasible region of the constrained optimization problems is very tiny in comparison to the entire search space. To solve problems with tiny feasible space, EAs usually take a long time to find even feasible solutions. With good quality initial solutions, the search operators reach the feasible region quickly and find better solutions. Some algorithms like GENOCOP (Michalewicz, 1994; Michalewicz and Janikow, 1996) assume a feasible starting point (or feasible initial population), which implies that the user or the EA must have a way of generating (in a reasonable time) such a starting point. The homomorphous mapping method of Koziel and Michalewicz (1999) also requires an initial feasible solution. As the initial populations of EAs are randomly generated, they may not be good quality solutions. Careful preprocessing, rather than providing manually a feasible solution, can improve the initial solutions, which not only accelerates the convergence but also finds better solutions. This encourages designing a search space reduction technique for solving COPs with tiny feasible space, by improving the quality of randomly generated initial solutions.

As mentioned above, the handling of equality constraints has long been a difficult issue for evolutionary optimization methods. In any optimization problems with equality constraints, each feasible solution point must lie on each and every equality constraint. The feasible space of the problems with equality constraints becomes too tiny in comparison to the whole search space, which makes it difficult to locate feasible and optimal solutions. Many traditional EAs convert the equality constraints $h_j(\boldsymbol{X}) = 0$ into inequality constraints $-\delta \le h_j(\boldsymbol{X}) \le \delta$ (where δ is a small tolerance value) to increase the feasible space temporarily (Deb, 2000). Still EAs may fail to achieve either feasible or good quality solutions in solving many COPs with equality constraints problems

(Joines and Houck, 1994; Koziel and Michalewicz, 1999; Mezura-Montes and Coello, 2002). This shows the necessity of an efficient technique to handle equality constraints.

In the majority of optimization problems, the usual objective is either maximizing the profit /revenue or minimizing the cost. A small improvement in solutions for such problems would save millions of dollars for a real world problem. This is possible if an effective algorithm and techniques can be developed to solve the COPs efficiently. The research topic considered in this thesis is independent of country or regional boundaries as it can be applied to any real valued constrained optimization problems.

1.3 Objective of the Thesis

The main objective of this research is to develop an integrated evolutionary system for solving constrained optimization problems efficiently. In the trail of the research, we have divided the research objective into three sub-objectives, which are described below along with the steps taken to achieve them.

Objective 1: *Developing an efficient evolutionary algorithm for solving COPs.*

The steps in achieving this objective are:

- Study the conventional optimization and other algorithms for solving COPs;
- Study the evolutionary algorithms and constraint handling techniques for EAs in solving COPs;
- Develop a simple genetic algorithm to solve COPs and analyze its performance;
- Study the prospects of MAS and Agent-based evolutionary algorithms;
- Develop an agent-based memetic algorithm for solving COPs and carry out experimental study;
- Analyze the performance of AMA and compare it with others from the literature; and
- Analyze the effect of different components used in AMA.

Objective 2: *Designing a technique for population-based EAs to enhance performance in solving COPs with tiny feasible region.*

The steps have been completed to achieve this objective are:

- Study and analyze the objective landscapes and the feasible spaces for different COPs;
- Analyze the performance of different EAs in solving COPs with tiny feasible space;
- Design a search space reduction technique (SSRT) to improve the quality of the randomly generated initial population;
- Test the performance of SSRT in solving test problems and real world problems, with a variety of EA-based algorithms including AMA; and
- Analyze the results and set suitable parameters of the better performance of SSRT.

Objective 3: *Designing a new equality constraint handling technique.*

To achieve this objective, the steps have been completed are:

- Analyze the performance of different EAs and identify the hurdles in solving COPs involving equality constraints;
- Develop a new Equality Constraint Handling Technique (ECHT);
- Integrate the ECHT in AMA as a learning process of the agents;
- Investigate the performance of the extended AMA for solving COPs with equality constrained problems;
- Analyze the results and effect of the new learning process;
- Design a modified genetic algorithm (MGA) with ECHT to overcome the limitations of GA in solving COPs with equality constraints; and
- Analyze the performance of MGA and the effects of its different components.

1.4 Contribution to Scientific Knowledge

This section summarizes the unique contributions made in this research. The scientific contributions made in this research, lie in the area of genetic and agent-based evolutionary algorithms for solving constrained optimization problems, as outlined below:

- **Agent-based memetic algorithm for solving COPs.** This thesis introduces a new agent-based memetic algorithm for solving constrained real-valued optimization problems, by tailoring multi-agent concepts into a new memetic algorithm. The rationale of designing the AMA architecture is discussed and analyzed. The agent learning processes and other operators used in AMA are discussed and their performances are analyzed. The proposed AMA architecture, for solving constrained real-valued optimization problems, is new in the literature. It is shown that the performance of the algorithm is very impressive in terms of achieving the optimal solutions.

- **Search space reduction technique for solving COPs with tiny feasible space.** This thesis presents a simple search space reduction technique for population-based evolutionary algorithms in solving constrained optimization problems with tiny feasible region. The idea behind the SSRT is analyzed, and its ability is explored solving a set of benchmark problems. This approach usually improves the performance of the algorithms, in terms of either solution quality or computational time or both, when investigated with AMA, a simple genetic algorithm, and three other existing well-known algorithms. Interestingly, the method is more appreciable for large scale problems with tiny feasible space. This concept of SSRT has not been observed in the literature.

- **A new technique to handle equality constraints.** A new technique to handle equality constraints is proposed in this thesis. The mathematical basis of the technique is discussed and its ability is explored; the use of the method as an

agent learning process is justified. The constraint handling techniques used here do not need any penalty functions or additional parameters. It is shown that a faster convergence and better solutions can be achieved with the new learning process. The idea behind the methodology is undoubtedly new in the literature.

- **A modified genetic algorithm for solving equality constrained optimization problems.** Finally, the ability of simple GA is enhanced by combining it with the proposed equality constraint handling technique. It is shown that the modified algorithm not only performs very well in terms of achieving optimal solutions, but also is robust in handling of both linear and nonlinear equality and inequality constraints.

1.5 Organization of the Thesis

This thesis has nine chapters and is organized as follows:

In chapter 1, an introduction to the thesis is presented. It first provides an overview of the research field, followed by the motivation behind this research. It also presents the objective of the thesis and a list of scientific contributions stemming from this research work. The last section of the chapter presents the organization of the thesis.

Chapter 2 provides a background study and basic fundamentals of the topics covered in this thesis. In the beginning, it provides a brief discussion on optimization problems and their existing solution methodologies. Then the characteristics and applications of genetic algorithms, multi-agent systems, and agent-based evolutionary algorithms are presented.

In chapter 3, a simple genetic algorithm is implemented to investigate the performance of GA in solving constrained optimization problems. A set of state-of-the-art test problems is used to investigate and analyze the performance of the algorithm.

Chapter 4 begins with a brief discussion on agent-based evolutionary algorithms. Then it presents a new agent-based memetic algorithm for solving COPs. The design of

the algorithm and details of the learning processes and other operators are also discussed in this chapter.

Chapter 5 reports detailed experimental studies of the AMA presented in chapter 4. It provides the results on a set of benchmark problems, compares the results with other well-known algorithms, and investigates the effect on its performance of different components of the algorithm.

In chapter 6, a simple method is presented that improves the quality of randomly generated initial solutions, while sacrificing very little in diversity of the population, in solving COPs with tiny feasible space. The performance of the proposed technique is tested using five different EAs, by solving a number of state-of-the-art test problems and a real world case problem.

A new equality constraint handling technique is presented in chapter 7 for solving constrained optimization problems with equality constraints. The technique is used as an agent learning process in AMA. This chapter also provides the details of experimental study, to see the performance of AMA with ECHT on a set of well-known benchmark problems, and to see the effect of the new learning process.

Chapter 8 presents a modified genetic algorithm that incorporates the ECHT presented in chapter 7 with a revised genetic algorithm. The algorithm is tested on a set of standard benchmark problems. Details of the effects of different components of the algorithm, and its performance on the test problems, are also presented in this chapter.

In Chapter 9, the main findings from this thesis are summarized. The chapter concludes the thesis with a discussion of possible future research directions.

Since this thesis discusses different spheres of research in solving COPs such as GA, AMA, SSRT for COPs with tiny feasible space, equality constraint handling technique, it can be presented to the readers in different ways. Figure 1.1 shows the flow chart of this thesis for readers with different interests.

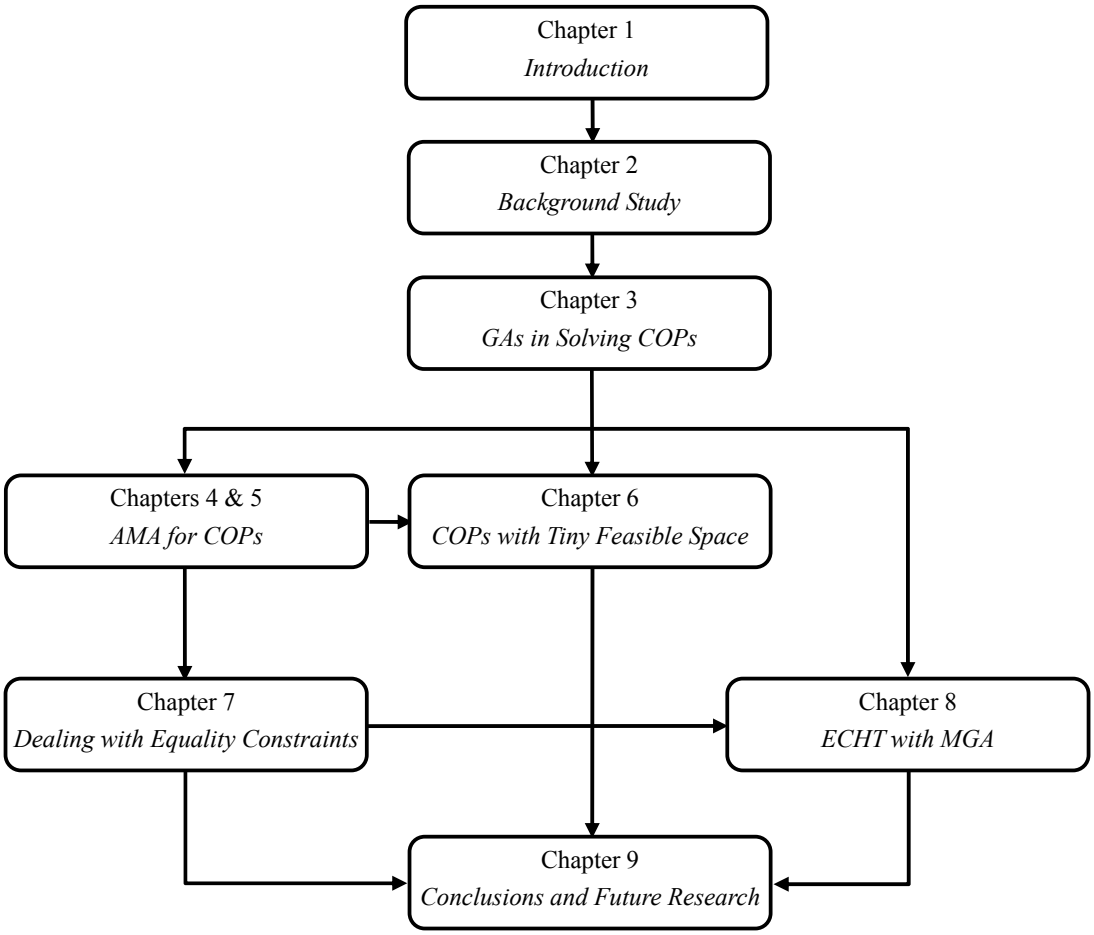


Figure 1.1: Flowchart of the thesis

Chapter 2

Background Study

This chapter provides a background study and basic fundamentals of the topics covered in this thesis. It starts with a brief discussion on optimization problems and their existing solution methodologies. Then the characteristics and applications of genetic algorithms, multi-agent systems, and agent-based evolutionary algorithms are presented.

The next few chapters of this thesis shall provide the literature review specific to those chapters.

2.1 Optimization Problems and their Classification

Many real world decision processes require solving optimization problems. The problems that need to optimize (either maximize or minimize) an objective function of a number of variables, subject to satisfying certain constraints, may be called optimization problems. Optimization problems are of high importance in industry and science. Solving optimization problems has become a challenging research topic in computer science and operations research due to the physical, geometric, and other limitations of different constraints (Liang and Suganthan, 2006).

Any function (either objective or constraint) in optimization problems can be a function of a single variable or a set of variables, depending on the problem characteristics. In general, the optimization problems can be represented as (without loss of generality, minimization is considered here):

Minimize

$$f(\boldsymbol{X}), \quad \boldsymbol{X}=[x_1,x_2,\ldots,x_n]$$

Subject to

$$\begin{aligned} & l_i \le g_i(\boldsymbol{X}) \le u_i, && i = 1,2,\ldots,p \\ & h_j(\boldsymbol{X})=c_j, && j = 1,2,\ldots,q \\ & \underline{x_k} \le x_k \le \overline{x_k} && k = 1,2,\ldots,n \end{aligned} \tag{2.1}$$

Where $\boldsymbol{X} \in R^n$ is a set of n variables of the solution, $f(\boldsymbol{X})$ is the objective function which needs to satisfy p inequality constraints ($g_i(\boldsymbol{X})$ represents i^{th} inequality constraint) and q equality constraints ($h_j(\boldsymbol{X})$ represents j^{th} equality constraint), $\overline{x_k}$ and $\underline{x_k}$ are the upper bound and lower bound of the variable x_k. Not all bounds on constraints ($l_i \le g_i(\boldsymbol{X}) \le u_i$,) may be present in a problem.

Optimization problems can be classified based on the objective functions, constraints, and variables involved. The problem may contain either a single objective function or multiple objective functions, and the objective type may be either maximization or minimization. In case of multiple-objective problems, the objectives usually contradict each other. If the objectives do not contradict then the multiple objectives can be converted easily into a meaningful single objective problem (Sarker and Newton, 2007).

Depending on the presence or absence of functional constraints, the optimization problems can be defined as constrained or unconstrained. The variable bounds are sometimes considered as constraints. The functional constraint types may be of equality (=), or inequality ($\le$ and $\ge$), or a mix of both. Unconstrained optimization problems arise not only from many practical applications, but also from the reformulation of constrained optimization problems required by many optimization algorithms (Turban and Meredith, 1994).

Depending on the nature of the problem, the variables in the model may be real or integer or a mix of both. An optimization problem with integer or discrete variables is termed a combinatorial problem. In combinatorial problems, the feasible space contains either a finite or infinite set of solution points. The feasible set for continuous

optimization problems is usually infinite as the values of variables are real numbers. This thesis considers only continuous optimization problems.

The objective or constraint functions may be either linear, nonlinear, or both. If all the functions are linear in a given problem, it is called a linear optimization problem. If one or more of the functions of the problem involve nonlinearity, it is called a nonlinear programming / nonlinear optimization problem. Many problems in engineering, science, and economics are nonlinear. The solution approaches of nonlinear problems are quite different and more complex than those of linear problems (Sarker and Newton, 2007).

The objective and the constraint functions may have mathematical properties such as convex or nonconvex, differential or nondifferential, and unimodal or multimodal, static or dynamic. The constraints may be treated as either soft or hard constraints. The concept of convexity is fundamental in classical optimization. Many traditional optimization techniques are developed based on the assumption that the function is convex, which generally makes them easier to solve both in theory and practice. The solution approaches in optimization can be classified in to two groups based on the use of derivatives. When using derivative-based techniques differentiability of the function is necessary, that is closely related to the continuity of functions. When a function has only one peak that is a global optimum solution, it is known as a unimodal function. On the other hand a function with more than one peak is considered as a multimodal function. If a function changes over time, it is known as a dynamic function. Hard constraints must be satisfied in the final solution, whereas soft constraints can be violated with a certain penalty or under certain conditions (Bazaraa *et al.*, 1990; Hillier and Lieberman, 2005).

The problems considered in this research are single objective constrained optimization problems. However, this research is not restricted to any particular function properties.

2.2 Applications of Optimization

We can see the applications of optimization everywhere. In fact, both humans and nature optimize many systems of interests. A few examples of optimization are briefly stated in this section.

Manufacturers plan for maximum efficiency in the design and operation of their production processes. Farmers try to minimize the production cost. Investors seek to create portfolios that avoid excessive risk while achieving a high rate of return. Engineers adjust parameters to optimize the performance of their designs. In nature the molecules in an isolated chemical system react with each other until the total potential energy of their electrons is minimized. Rays of light follow paths that minimize their travel time (Nocedal and Wright, 2006). Even the population relocation in response to climate change, considering people's preferences, various costs and planning priorities, is also an optimization problem (Zahir *et al*., 2009).

Some applications of optimization problems are given below:

Production Systems: The production planning problem can be looked at as a system of systems: forecasting, material handling, personnel, purchasing, quality assurance, production, assembly, marketing, design, finance, and other appropriate systems. The design is needed to optimize in such a way that it is easy to produce by using snap-in fasteners; materials easy to form; financial planning provides appropriate working capital; purchases arrive on time and have appropriate quality (Miranker and Lofaso, 1991; Van der Duyn Schouten and Vanneste, 1995; Vatn *et al.*, 1996).

Energy Systems: The oil industry was one of the first users of optimization techniques to help manage their refinery operations (Méndez *et al.*, 2006; Reddy *et al.*, 2004). Electrical and hydro-electric companies use optimization techniques (Alyabysheva *et al.*, 1975; Hanjie and Baldick, 2007) to determine how to efficiently produce power as well as trade power among their partners.

Transportation: Real-time dispatching and delivery truck routing (Handa *et al.*,

2006), healthy package delivery, and international freight including the scheduling and pricing of containers need optimization techniques to minimize cost and enhance performance (Loannou, 2008; Pursula and Niittymäki, 2001).

Airline Optimization: The airline industry was one of the first to apply operations research methods to commercial optimization problems. The combination of advancements in computer hardware and software technologies with clever mathematical algorithms and heuristics has dramatically transformed the ability of operations researchers to solve large scale, sophisticated airline optimization problems over the past 60 years (Snowdon and Paleologo, 2007). Revenue management and pricing, airline network planning, crew scheduling, maintenance planning, spares inventory management, and fuel management all need optimization (Jacobs *et al.*, 2005; Wu, 2006).

Project Management: Project management techniques continue to be a major avenue to accomplishing goals and objectives by optimization (Zarka, 2005) in various organizations ranging from government, business, and industry to academia (Badiru and Pulat, 1994).

Military applications: Military applications use optimization techniques for solving personnel force management, logistics, transportation, war gaming, strategic planning, tactical planning, and many other (Soon, 2003; Weber *et al.*, 2006; White, 1990).

Data mining: With the widespread emergence of very large databases, many different organizations are finding vital help from professionals in extracting the information they really want. The data mining methods, which are growing rapidly, give superior solutions in diverse database-plumbing applications such as: predicting purchasing behavior, segmenting customers, detecting fraud, assessing credit risk, and anticipating customer attrition. Optimization is a powerful and effective tool for these applications (Shi *et al.*, 2008).

E-commerce: E-commerce offers opportunities in business-to-business and

consumer areas of online purchasing, vendor purchasing models, online auctions, and supply procurement. In addition to sellers, bidders are looking for decision support on how to bid intelligently. In all the cases, the optimization process is involved (Kuechler *et al.*, 2001).

Environmental applications: Pollution control (Qiong *et al.*, 2007; Shih *et al.*, 1998), the design of systems to prevent shipping accidents, and population relocation in response to climate change (Zahir *et al.*, 2009) are optimization problems.

2.3 Solution Methodology

The solution approaches for optimization problems can be divided into two major groups: (1) conventional optimization techniques and (2) modern heuristic techniques. Here some popular techniques from both groups are discussed, however the main emphasis will be given to some of the modern heuristic techniques.

2.3.1 Conventional Methods

In the conventional optimization domain, the solution approach for a given type of model is determined by the problem classification e.g. linear programming for linear optimization problems, Integer Programming (IP) when some or all of the decision variables are restricted to integer or discrete values. Some conventional methods are briefly discussed below:

Linear Programming (LP)

Most of the LP methods determine the feasible solution space from the model of the problem and then the optimal point is identified within the feasible space. The graphical method, simplex method and its variants, and the interior point method are basically the most popular approaches to solve linear optimization problems.

The main purpose of the graphical method is to illustrate the concepts of acceptable solutions and search boundary. The method has practical value when solving small

problems with two decision variables and only a few constraints (Turban and Meredith, 1994). The simplex method is an iterative process which moves from one vertex of the feasible space to another, in each iteration, until it reaches the optimal solution. On the other hand, the interior point algorithm moves through the feasible space towards the optimal solution. The method usually obtains a near optimal solution. For large-scale LP models, the interior point method is much more efficient but provides only an approximate solution (Dantzig and Thapa, 2003).

Integer Programming (IP)

Integer programs are mathematical programming models (linear or nonlinear) where some or all of the variables are assumed to be integer or discrete values. If all variables take integer values, then the problem is called a pure IP. On the other hand, if both integer and continuous variables coexist, the problem is called a mixed integer program (MIP) or mixed integer linear program (MILP). It is well known that integer and mixed -integer linear models are difficult to solve. This is due to the fact that the number of alternative solutions increases much faster (usually exponentially) than the size of the problem. That makes the large-scale integer program extremely difficult to solve using the existing algorithms.

Integer programming can be considered as an extension of the general LP problem. IP borrows many concepts and techniques from LP when developing solution approaches. The common solution approaches for solving IP models are complete enumeration, graphical method, rounding the non-integer solution, branch-and-bound, cutting plane, and branch-and-cut method. The details of these methods can be found in (Hillier and Lieberman, 2005; Nemhauser and Wolsey, 1999).

Goal Programming (GP)

The basic idea of goal programming is to establish a specific numeric goal for each of the objectives, formulate a new objective function as the weighted sum of all absolute deviations from the goals, and then seek a solution that minimizes the revised objective function. There are three possible types of goals:

1. A lower, one-sided goal sets a lower limit.
2. An upper, one-sided goal sets an upper limit.
3. A two-sided goal sets a specific target that does not want to miss on either side.

Goal programming problems can be categorized as preemptive and non-preemptive. In non-preemptive goal programming, all the goals are of roughly comparable importance. In preemptive goal programming, there is a hierarchy of priority levels for the goals, so that the goals of primary importance receive first priority attention, those of secondary importance receive second-priority attention, and so on. The non-preemptive goal programming can be solved using the simplex method since the coefficients of the objective function are known numerical values. The preemptive goal programming models are solved either by using a sequential or a streamlined method (Hillier and Lieberman, 2005).

Nonlinear Programming (NLP)

Nonlinear optimization problems can be classified as unconstrained or constrained. The constrained problems can be divided into linearly constrained, quadratic, convex, non-convex, separable, geometric, and fractional programming. No one algorithm can solve all classes of nonlinear models since each of the nonlinear models has its own well defined set of characteristics (Bazaraa *et al.*, 2006).

The unconstrained problem solving approaches are divided into single variable and multivariable problems with and without using derivatives. The one-dimensional search is the backbone of many algorithms for solving a nonlinear programming problem. There are a number of line search procedures described in the literature, for solving unconstrained problems of one variable, with or without using derivatives.

Multidimensional search can also be performed with or without using derivatives. Examples of the multidimensional search without using derivatives are the cyclic coordinate method, the method of Hooke and Jeeves, and Rosenbrock's method. Examples of multidimensional search using derivatives include the steepest decent method and the method of Newton. For more details, see (Bertsekas, 1995).

The penalty function method is a well-known approach for solving constrained nonlinear optimization problems. The approach converts the constrained problem into an equivalent unconstrained problem and then solves the problem using a suitable search algorithm. The constraints are placed into the objective function via a penalty parameter in such a way that the parameter penalizes any violation of the constraints. There are many other existing algorithms for solving nonlinear models such as barrier function, gradient projection, reduced gradient, method of Zoutendijk, and the convex–simplex method. For more details see (Bertsekas, 1995).

Multi-Objective Models

In multi-objective models, it requires making compromises or trade-offs regarding the outcomes of alternate objectives, hence in this type of problem there exists no single optimal solution, rather a set of alternative solutions. There are several methods described in the literature that can be used when solving multi-objective optimization problems such as weighting method (Gass and Saaty, 2006; Zadeh, 1963), ε-constraint method (Haimes *et al.*, 1971), goal attainment, lexicographic ordering (Miettinen, 2001), interactive surrogate worth trade-off method (Chen *et al.*, 2002), Geoffrin–Dyer–Feinberg method (Geoffrion *et al.*, 1972), sequential proxy optimization techniques (Sakawa, 1982), the Tchebycheff method (Steuer and Choo, 1983; Steuer, 1986), reference point method (Wierzbicki, 1982; Wierzbicki and Granat, 1999), satisfying trade-off method, light beam search and the reference direction approach. Further details of these methods can be found in (Eiselt *et al.*, 1987; Miettinen, 1999; Miettinen, 2001)

2.3.2 Heuristic Methods

In optimization problem solving, a heuristic is a rule-of-thumb approach that may not guarantee convergence and optimality. However, in most cases, they work well and produce solutions of acceptable quality. Developing solutions with these methods offers two major advantages: 1) development time is much shorter than when using more traditional approaches, and, 2) the systems are very robust, being relatively insensitive

to noisy and/or missing data.

The use of a heuristic approach in optimization is not new. However, in the past, heuristics were developed based on the concept of either conventional optimization techniques or traditional artificial intelligence techniques. Nowadays, heuristics are also inspired by biology, physics, neuroscience, and other disciplines. The field of heuristics is growing very rapidly. Some of the widely used heuristics are discussed briefly in this section.

Hill Climbing

The main idea of Hill climbing is to continue the search process until the new solution is better than the best solution found so far. It is the greediest heuristic yet encountered. The algorithm is more likely to trap into a local optimum as it represents pure search intensification without any chance for search exploration. So this search process can be very sensitive in regard to the starting point (Russell and Norvig, 2003).

Simulated Annealing (SA)

Based on the "annealing" process in the statistical mechanics, simulated annealing was introduced for solving complicated optimization problems. SA accepts a lower-quality solution in an iteration with some probability depending on a parameter called *temperature*. The algorithm behaves like a random search at high temperature (using a higher probability) and like a greedy hill-climbing at low temperature (with a probability close to zero) (Sait and Youssef, 2000). In the algorithm, a cooling schedule with an initial temperature must be defined by the user, which is not an easy task. The SA algorithm grows exponentially with respect to the size of the problem. Another disadvantage of using the iterative improvement is that it may be trapped in local optima. To avoid this disadvantage, simulated annealing accepts in a limited way neighboring solutions with a cost that is worse than the cost of the current solution. The details of the algorithm can be found in (Gonzalez, 2007; Kirkpatrick *et al.*, 1983; Ravindran, 2007; Van Laarhoven and Aarts, 1987).

Tabu Search (TS)

Tabu search is a meta-heuristic that guides a local heuristic search procedure to explore the solution space beyond local optimality (Gendreau, 2003). Unlike the conventional hill-climbing approach, TS may allow lower-quality solutions in any intermediate iteration. TS also forbids reverse moves to avoid cycling. The forbidden movements are recorded in a data structure called a tabu list. This adaptive memory feature of TS allows the implementation of procedures that are capable of searching the solution space economically and effectively. However, the performance of TS is sensitive to the size of the tabu list in many practical applications. A comprehensive examination of this methodology can be found in (Glover and Laguna, 1997).

Evolutionary Algorithms (EAs)

Evolutionary Algorithms have attracted increasing attention in recent years, as powerful computational techniques for solving many complex real-world problems (Sarker *et al.*, 2003). EAs can be regarded as a metaphor for building, applying, and studying algorithms based on Darwinian principles of natural selection. They are inspired by nature's capability to evolve living beings well adapted to their environment.

Each EA starts with a randomly generated population of individual solutions. Different EAs use different representations (e.g. lists, trees, graphs) for the individuals. A good representation will make a problem easier to solve and a poor representation will do the opposite. At every algorithm generation/iteration a number of reproduction operators are applied to the individuals of the current population to generate the individuals of the population for the next generation. Evolutionary algorithms might use operators called recombination or crossover to recombine two or more individuals to produce new individuals. They also might use mutation operators that cause a self-adaptation of individuals. The main driving force in EAs is the selection of individuals based on their fitness (it may be based on the objective function, the result of a simulation experiment, or some other kind of quality measure). Individuals with higher

fitness have a higher probability to be chosen as members of the population of the next generation (or as parents for the generation of new individuals). This corresponds to the principle of survival of the fittest in natural evolution. It is the capability of nature to adapt itself to a changing environment, which gave the inspiration for EAs.

There has been a variety of different EAs proposed over the years such as Evolution strategies (ES), Evolutionary Programming (EP), genetic algorithms and genetic programming. Each of these algorithms approximates the evolutionary processes in different ways. GAs, the most well known branch of EAs, have been successfully used for numerical optimization, combinatorial optimization, classifier systems, wire routing, scheduling, transportation problem and many other engineering problems (Goldberg, 1989; Michalewicz, 1994). GAs will be discussed in the next section in more detail as some of the research in this thesis is based on GAs.

Memetic Algorithms (MAs)

MAs are population-based meta-heuristic search algorithms, inspired by Neo–Darwinian's principles of natural evolution and Dawkins' notion of a meme defined as a unit of cultural evolution that is capable of performing individual learning (Ong *et al.*, 2009). It can be considered as a marriage between the population-based global search and the heuristic-based local search. Any constructive heuristics may be combined with a population-based algorithm (e.g. GAs) to develop a memetic algorithm. When a local search is combined with GAs (usually known as genetic local search), the algorithm provides a much better performance than GAs alone can do (Hasan *et al.*, 2008). MAs have been successfully applied across a wide range of problem domains such as combinatorial optimization (Alkan and Ozcan, 2003; Burke and Smith, 1999; Cheng and Gen, 1996; Merz and Freisleben, 1997; Merz and Freisleben, 2000; Tang *et al.*, 2005), optimization of non-stationary functions (Vavak *et al.*, 1996), and multi-objective optimization (Hu *et al.*, 2003; Knowles and Corne, 2000; Knowles and Corne, 2001; Knowles and Corne, 2005). They converge to high quality solutions as well as search more efficiently than their conventional counterparts (Tang *et al*., 2007). Some theoretical and empirical investigations on MAs can be found in (Goldberg and

Voessner, 1999; Hart, 1994; Krasnogor, 2002; Krasnogor and Smith, 2005; Merz and Freisleben, 1999; Ong and Keane, 2004; Ong *et al.*, 2009; Tang *et al.*, 2007).

Other Heuristics

There are a number of other heuristics such as immune system, ant colony optimization, particle swarm optimization, cultural algorithms, and cooperative search. A brief description of these algorithms can be found in (Coello *et al.*, 2002; Gonzalez, 2007; Olariu and Zomaya, 2006).

2.4 Genetic Algorithms (GAs)

Genetic Algorithms are stochastic algorithms which simulate both the natural inheritance by genetics and the Darwinian strive for survival (Darwin, 1859). Genetic algorithms were developed by Holland (1975) and his students and colleagues at the University of Michigan in the 1960s and the 1970s. Genetic algorithms were first proposed as adaptive search algorithms, although they have mostly been used as a global optimization algorithm for either combinatorial or numerical problems. GAs are the most widespread variant of EAs (Gonzalez, 2007; Sarker *et al.*, 2003). They start with a randomly generated population (a set of solutions) and then move from one population to another. This process continues until the stopping criteria are met. At each iteration, a new population is generated applying various search operators. Details of the operators are discussed later in this section.

GAs do not require any rich domain knowledge, so they are not difficult to implement. Considerations of convexity/concavity and continuity of functions are not necessary in GAs. However, these attributes of functions are real concerns in most mathematical programming techniques. GAs strike a remarkable balance between exploration and exploitation of the search space, and so are very useful for solving multi-modal problems. This property also helps to improve the solution by skipping

from the local optima. The most favorable point of using GAs is that they provide quick approximate solutions. GAs are more suitable for multi-objective optimization than conventional optimization techniques, because of their capability of simultaneous optimization of conflicting objective functions and generation of a number of alternative solutions in a single run (Sarker *et al.*, 2003). These advantages of GAs over the conventional mathematical programming techniques justify the use of them in solving optimization problems.

2.4.1 Basic Structure

It turns out that there is no rigorous definition of genetic algorithm accepted by all in the evolutionary computation community that differentiates GAs from other evolutionary computation methods (Melanie, 1998). However, most methods called GAs have at least the following elements in common: populations of chromosomes, selection process according to fitness, crossover to produce new offspring, and random mutation of new offspring.

A genetic algorithm explores a number of potential solutions in parallel. It initially creates a population randomly. A population is a set of individuals, each of which has its own genetic content called chromosome. The chromosomes are strings of smaller units termed genes. The different values a gene can take are called the alleles for that gene.

Binary Representation is the most used chromosomes representation, in which each decision variable is represented with more than a gene, where each gene is a binary digit, as shown in Figure 2.1. It needs coding and decoding to map between the genotype and the phenotype if the decision variables are not binary. The structure of a solution vector and representation in any optimization problem depends on the underlying problem (Bäck *et al.*, 2000). Since any arbitrary contiguous region in the search space cannot be represented by this representation and the feasible search space can usually be of any arbitrary shape, we can use *Floating point representation* (also known as real-coded representation). The real-coded GAs overcome the difficulties of

achieving arbitrary precision in decision variables and the Hamming cliff problem (Goldberg, 1989) associated with binary representation of real numbers (Deb, 2000). In floating point representation the decision variables are directly represented in real numbers in only one gene shown in Figure 2.2. This floating point representation is used in this research. There are also combined representation, finite-state representations, Parse trees, or other complex structures used for representation depending on the problem (Bäck *et al.*, 2000).

Variable	x_1			x_2			x_3	
Genotype	0	0	1	1	0	0	0	1	0
Phenotype		1			4			2	

Figure 2.1: Binary representation of chromosome.

Variable	x_1	x_2	x_3	……	x_n
	4.1	2.2	3.1	……	2

Figure 2.2: Floating point representation of cromosome.

In the initial population creation process, the genetic contents of individuals (i.e. chromosomes) are generally produced in a randomized fashion in order to assure diversity in the initial population. Afterwards, in a loop of evolution, individuals and their offspring are transferred to new generations, taking into consideration the quality of their chromosomes, which is called fitness. A better fitness value gives to an individual a better chance to be selected for survival or reproduction. The main feature of GAs is the use of a recombination operator as the primary search tool (Olariu and Zomaya, 2006). The motivation is the assumption that different parts of the optimal solution can be independently discovered, and be later combined to create better solutions. Moreover the mutation is also used, but it is considered as a secondary background operator whose purpose is merely "keeping the pot boiling" by introducing new information in the population (Olariu and Zomaya, 2006). An individual that exists in the current generation may be selected directly, or it may be matched with another individual and the resulting offspring may be transferred to the next generation. This process continues until the termination condition has been reached. Common

terminating conditions are when an upper limit on the number of generations is reached, an upper limit on the number of evaluations of the fitness function is reached, the chance of achieving significant changes in the next generations is excessively low, or allocated budget (computation time/money) is reached, etc (Michalewicz, 1994; Safe *et al.*, 2004).

A general framework of basic genetic algorithm can be summarized as follows:

Pseudo code: Genetic Algorithm.

(1) Set $i = 0$;

(2) Generate the initial population $P(i)$ at random;

(3) REPEAT

(a) Evaluate the fitness of each individual in $P(i)$;

(b) Select parents from $P(i)$ based on their fitness;

(c) Apply reproduction operators (crossover and/or mutation) to the parents and produce generation $P(i+1)$;

(4) UNTIL the terminating conditions is reached.

2.4.2 Operators and Parameters

As discussed earlier the simplest form of genetic algorithm involves three types of genetic operators: selection, crossover, and mutation. These genetic operators are performed on the chromosomes of the current generation to produce child generations that become fitter in the simulated evolution process. The details of these operators are given below:

Selection

The selection scheme determines the probability of an individual to survive or be

selected to reproduce offspring. This operator is designed to improve the average quality of the population by giving individuals of higher fitness a higher probability to be copied to or produce the new individuals in the child generation. The quality of an individual in the current generation is measured by its fitness value through the evaluation of the fitness function; therefore, the selection can focus on more promising regions in the search space. A number of selection schemes, such as roulette wheel selection (also known as the fitness proportional selection) and rank-based selection, as well as tournament selection, have been popularly used in GA (Sarker *et al.*, 2003). In the first scheme individuals are chosen for selection in proportion to their fitness value. In rank-based selection, the population is sorted from best to worst. The fitness assigned to each individual depends only on its position in the ranking; the higher ranked individuals are given higher probabilities to survive. As the name suggests, in tournament selection a random number of individuals are chosen from the population and the best individual from this group is chosen as a parent for reproduction (Elfeky *et al.*, 2008; Liu and Han, 2003; Sarker *et al.*, 2003). The selection operation is a successful artificial emulation of natural selection of the Darwinian survival theory.

Crossover

After the selection operation is completed and the mating pool is formed, the crossover operator may proceed. Crossover is an operation to exchange part of the genes in the chromosomes of two parents in the mating pool to create new individuals for the child generation. It is believed to be the key search operator in the working of a GA as an optimization tool (Goldberg, 1989). In binary representation, one-point, n-point, uniform crossovers are commonly used. In floating point representation, offspring can be generated with different types of crossover such as one-point crossover, n-point crossover, arithmetic crossover (Michalewicz, 1994), geometric crossover (Michalewicz, 1994), heuristic crossover (Wright, 1991), Simulated Binary Crossover (SBX) (Deb and Agrawal, 1995), orthogonal crossover (Leung, 2001), simplex crossover (Renders and Bersini, 1994), and Partially Mapped Crossover (PMX) (Goldberg and R. Lingle, 1985). Some of these crossover operators are briefly discussed

below.

For one-point crossover, a crossover point k is randomly selected at which an exchange of parent chromosomes is made. In this crossover, the solution vectors X_1, X_2 of two parent solution vectors, each of dimension n, are swapped after the k point and produce X_1', X_2' offspring as shown in Figure 2.3. This operator can be extended to a two-point crossover in which two crossover points k_1, k_2 are selected at random and the segments between these two points is exchanged between the parents as shown in Figure 2.4.

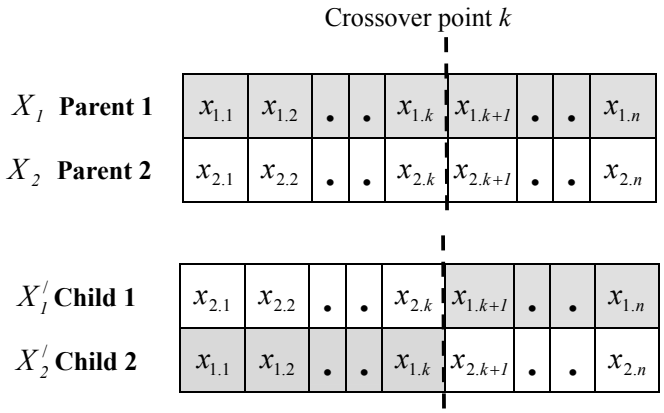


Figure 2.3: One-point crossover.

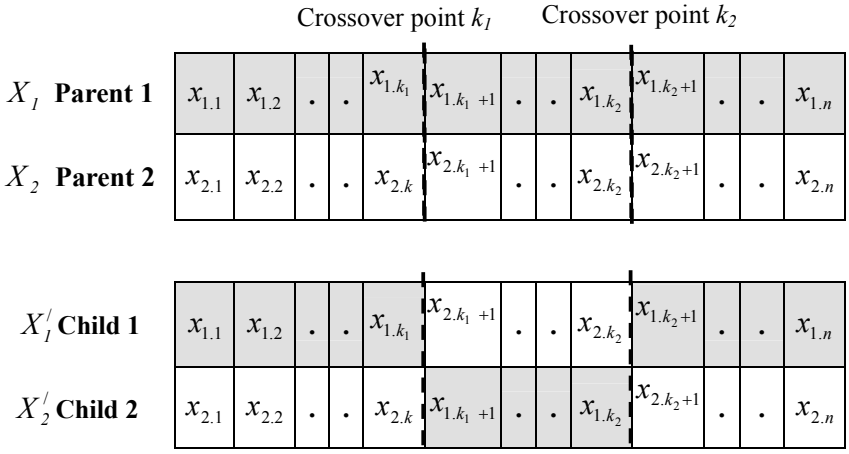


Figure 2.4: Two-point crossover.

Based on the search features of the single-point crossover used in binary-coded genetic algorithms, the simulated binary crossover operator respects the interval schemata processing, in the sense that common interval schemata of the parents are preserved in the offspring (Deb and Agrawal, 1995). The spread of offspring solutions around parent solutions can be controlled using a distribution index. With this operator any arbitrary contiguous region can be searched, provided there is enough diversity maintained among the feasible parent solutions (Deb, 2000). SBX operator performs well in solving problems having multiple optimal solutions with a narrow global basin, and has been used in different applications successfully (Deb, 2000; Deb, 2001; Deb and Agrawal, 1995; Deb *et al.*, 2002; Gupta and Deb, 2005; Srinivas and Deb, 1994; Srinivasan and Rachmawati, 2006).

For handling multiple variables, each variable of the solution vector is chosen with a probability 0.5 in this study and the following SBX operator is applied variable-by-variable. The procedure of computing children solutions variable $x_i^{(c_1)}$ and $x_i^{(c_2)}$ from two parent solution variable $x_i^{(p_1)}$ and $x_i^{(p_2)}$ is as follows:

$$x_i^{(c_1)} = 0.5[(x_i^{(p_1)} + x_i^{(p_2)}) - \overline{\beta}(x_i^{(p_2)} - x_i^{(p_1)})] \tag{2.2}$$

$$x_i^{(c_2)} = 0.5[(x_i^{(p_1)} + x_i^{(p_2)}) + \overline{\beta}(x_i^{(p_2)} - x_i^{(p_1)})] \tag{.2.3}$$

where $\overline{\beta}$ is the ordinate of a probability distribution, which is chosen in such a way that the area under the probability curve from 0 to $\overline{\beta}$ equals a random number u ($0 \leq u \leq 1$).

$$\overline{\beta} = \begin{cases} (2u)^{1/(\eta_c+1)} & \text{if } u \leq 0.5, \\ \left(\dfrac{1}{2(1-u)}\right)^{1/(\eta_c+1)} & \text{otherwise,} \end{cases} \tag{2.4}$$

where η_c is the distribution index for SBX and can take any nonnegative value. A small value of η_c allows solutions far away from parents to be created as children solutions and a large value restricts only near parent solutions to be created as children solutions. In all simulation, Deb (2000) have used η_c=1. For details of this crossover see (Deb and Agrawal, 1995).

Neighborhood Orthogonal Crossover operator was proposed by Leung (2001). The orthogonal crossover operator acts on two parents and generates a set of new individuals from the search space defined by the two parents. The search space is quantized into a finite number of points, and then orthogonal design is applied to select a small but representative sample of points as potential offspring. The main advantage of this crossover is that the orthogonal array can specify a small number of uniformly scattered individuals over the search space. Then the best individuals among them can be considered as the new offspring.

Consider the parents $A_{i,j}$=[a_1,$a_{2,...}a_n$] and $B_{i,j}$= [b_1,$b_{2,...,}b_n$] where the search space is [$\underline{x_{AB}}$,$\overline{x_{AB}}$] and $\underline{x_{AB}} = min(a_i,b_i)$, $\overline{x_{AB}} = max(a_i,b_i)$, for $i = 1,2,...,n.$

Each domain value [$\underline{x_{AB}}$,$\overline{x_{AB}}$] is quantized into Q levels such that the successive levels are equally distant. The domain of each i^{th} dimension is quantized to $\beta_{i,1},\beta_{i,2},...,\beta_{i,Q}$ where

$$\beta_{i,j} = \begin{cases} min(a_i,b_i) & j = 1 \\ min(a_i,b_i) + (j-1).(\dfrac{|a_i - b_i|}{Q-1}) & 2 \le j \le Q-1 \\ max(a_i,b_i) & j = Q \end{cases} \qquad (2.5)$$

For simplicity the n variables $A_{i,j}$=[a_1,$a_{2,...}a_n$] are divided into F groups and each group is considered as one factor. Each factor is quantized and produces $f_1 = (a_1,....,a_{k_1}), f_2 = (a_{k_1+1},....,a_{k_2}),...,f_F = (a_{k_{F-1}+1},....,a_{k_F})$.

Here $k_1,k_2,...,k_{F-1}$ are random numbers such that $1 < k_1 < k_2 ... < k_{F-1} < n$.
After quantization, Q levels of i^{th} factors are:

$$\begin{cases} f_i(1) = (\beta_{k_{i-1}+1,1}, \beta_{k_{i-1}+2,1},..., \beta_{k_i,1}) \\ f_i(2) = (\beta_{k_{i-1}+1,2}, \beta_{k_{i-1}+2,2},..., \beta_{k_i,2}) \\ \\ f_i(Q) = (\beta_{k_{i-1}+1,Q}, \beta_{k_{i-1}+2,Q},..., \beta_{k_i,Q}) \end{cases} \qquad (2.6)$$

The orthogonal array $L_{M_2}(Q^F) = [b_{i,j}]_{M_2 \times F}$ is then applied to generate the following M_2 offspring:

$$\begin{cases} f_1(b_{1,1}), f_2(b_{1,2}),..., f_F(b_{1,F}) \\ f_1(b_{2,1}), f_2(b_{2,2}),..., f_F(b_{2,F}) \\ \\ f_1(b_{M_2,1}), f_2(b_{M_2,2}),..., f_F(b_{M_2,F}) \end{cases} \tag{2.7}$$

The orthogonal of $L_4(2^3)$ and $L_9(3^4)$ are as follows:

$$L_4(2^3) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix} \quad L_9(3^4) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \\ 1 & 3 & 3 & 3 \\ 2 & 1 & 2 & 3 \\ 2 & 2 & 3 & 1 \\ 2 & 3 & 1 & 2 \\ 3 & 1 & 3 & 2 \\ 3 & 2 & 1 & 3 \\ 3 & 3 & 2 & 1 \end{bmatrix} \tag{2.8}$$

The details of the crossover is discussed in (Leung, 2001).

Mutation

The mutation operator is designed so that one or more of the chromosome's genes will be mutated at a small probability. The goal of the mutation operator is to prevent the genetic population from converging to a local minimum and to introduce a bit more diversity to the population. Without this operator, the population would rapidly become uniform under the so-called conjugated effect of selection and crossover operator (Liu *et al.*, 2002a). There are a number of mutation methods such as bit-flipping (for binary representation), uniform mutation, non-uniform mutation, gaussian mutation, parameter-based mutation and cauchy mutation. Details of mutation can be found in (Bäck *et al.*, 2000; Liu and Han, 2003).

Here we shall discuss the parameter-based mutation operator (Deb, 2000; Deb and Goyal, 1996) in brief. Parameter-based mutation operator is applied for exploring the vicinity of a parent solution, in which the variable x_i is mutated to x_i' (where lower and upper boundaries are not specified) as below:

$$x_i' = x_i + \bar{\delta}\,\Delta_{max} \tag{2.9}$$

where Δ_{max} is the maximum permissible perturbance to x_i allowed in the parent solution and $\bar{\delta}$ is the perturbance factor, which can be calculated for a random number u ($0 \le u \le 1$)

$$\bar{\delta} = \begin{cases} (2u)^{1/(\eta_m+1)} - 1 & \text{if } u \le 0.5, \\ 1 - (2(1-u))^{1/(\eta_m+1)} & \text{otherwise,} \end{cases} \tag{2.10}$$

where η_m is the distribution index for mutation and takes any nonnegative value.

For variables where lower and upper boundaries ($\underline{x_i}$ and $\overline{x_i}$) are specified, the above equation may be changed as follows:

$$\bar{\delta} = \begin{cases} \{2u + (1-2u)(1-\delta)^{\eta_m+1}\}^{1/(\eta_m+1)} - 1 & \text{if } u \le 0.5, \\ 1 - \{2(1-u) + 2(u - 0.5)(1-\delta)^{\eta_m+1}\}^{1/(\eta_m+1)} & \text{otherwise,} \end{cases} \tag{2.11}$$

where $\delta = \min[(x_i - \underline{x_i}), (\overline{x_i} - x_i)/(\overline{x_i} - \underline{x_i})]$.

Genetic Parameters

The parameters usually used in genetic algorithms are the population size, generation number, probability of crossover, and probability of mutation. Population size represents the total number of individuals in the population of the GAs. The maximum number of generations the GAs will execute is indicated by the generation number. The proportion of parents undergoing crossover and mutation in a generation are controlled by the probability of crossover (P_C) and probability of mutation (P_M) respectively.

2.4.3 Handling Constraints in GA

Although GAs perform well for unconstrained or simple Constrained Optimization Problems (COPs), they may face difficulties when applied to solving highly constrained

problems since the initial version of GAs are designed as unconstrained optimization procedures. The traditional search operators of GAs (i.e., crossover and mutation) are blind to constraints. In such a circumstance, it is very likely that the candidate solutions generated by these operators during the search process would violate certain constraints (Chootinan and Chen, 2006). Hence constraint handling is one of the major concerns when applying GAs to solve constrained optimization problems. The main challenge in constrained optimization is to find the feasible as well as the optimal solution. Over the past decade various constraint-handling techniques using genetic algorithms have been proposed. According to Coello (2002) these techniques can be grouped as follows:

- Penalty functions,
- Special representations and operators,
- Repair algorithms,
- Separation of objectives and constraints, and
- Hybrid methods.

Penalty functions were popularly used in the conventional methods for constrained optimization and were amongst the first methods used to handle constraints with evolutionary algorithms (Coello, 2002; Fletcher, 1990). The principal idea of this kind of method is to redefine or reformulate the constrained optimization problem as an unconstrained one, by introducing a penalty term into the original objective function to penalize constraint violations. The individuals are penalized in different ways based on the constraint violations. In the static penalty method (Homaifar *et al.*, 1994; Morales and Quezada, 1998), the penalty term added to the objective function increases with the degree of constraint violation. In the dynamic penalty approach (Joines and Houck, 1994), the penalty term increases with both the degree of constraint violation and generation number. The greatest penalty that can be imposed on an infeasible solution is applied in the death penalty method (Hoffmeister and Sprave, 1996) i.e. the infeasible solutions are not considered for selection for the next generation. The idea of simulated annealing (Kirkpatrick *et al.*, 1983) is used in annealing penalty methods (Carlson and Shonkwiler, 1998; Michalewicz, 1995; Michalewicz and Attia, 1994). When the

algorithm is trapped in a local optimum the penalty coefficients are changed. In the co-evolutionary penalty method (Coello, 2000c) the penalty is split into two values to provide enough information about how many constraints are violated and the corresponding amounts of violation. Adaptive penalty methods (Bean and Hadj-Alouane, 1993; Ben Hadj-Alouane and Bean, 1997; Coit *et al.*, 1996; Crossley and Williams, 1997; Eiben and Hauw, 1998; Farmani and Wright, 2003; Rasheed, 1998) can make use of information obtained during the search to adjust their own parameters. Due to their simplicity and ease of implementation penalty methods were the most common methods used in solving real world problems (Farmani and Wright, 2003). However most of the methods require a careful fine-tuning of parameters to obtain competitive results. They also lack generality and are usually only fit for optimization problems with certain constraint types (Cai and Wang, 2006; Coello, 2002).

Some researchers change the representation to simplify the shape of the search space and have developed special operators to preserve the feasibility of solutions at all times. The main application of this approach is naturally in problems in which it is very difficult to locate at least a single feasible solution (Coello, 2002). For example, Davidor (1991) introduced a varying-length genetic algorithm to generate robot trajectories, and defined a special crossover operator called analogous crossover, which uses phenotypic similarities to define crossover points in the parent strings. The GENOCOP proposed by Michalewicz (1994) and Michalewicz and Janikow (1996) is based on designing specialized operators that incorporate knowledge of the constraints. This method uses projection operators that map feasible points back to feasible boundaries. GENOCOP tries to locate an initial (feasible) solution by sampling the feasible region. If it does not succeed after a certain number of trials, the user is asked to provide such a starting point. GENOCOP works efficiently for problems with linear constraints (Michalewicz, 1994). GENOCOP II enhanced the performance and was able to solve general nonlinear programming problems (Michalewicz and Attia, 1994).

In the Decoder method, the chromosome does not directly encode a solution in the feasible region but rather "gives instructions" on how to build a feasible solution. Each decoder imposes a relationship between a feasible solution and a decoded solution

(Dasgupta and Michalewicz, 1997). The homomorphous mapping approach (Koziel and Michalewicz, 1999) converts constrained problems into unconstrained optimization problems by using a mapping between an *n*-dimensional cube and the feasible space of the given problem. However, the implementation of this method is very difficult especially for nonconvex feasible search spaces. It requires initial feasible solutions which are difficult to find for most complex problems.

The repair algorithms attempt to improve infeasible solutions to feasible by taking advantage of the problem's characteristics (Chootinan and Chen, 2006; Jing *et al.*, 1996; Jing *et al.*, 1997; Michalewicz and Nazhiyath, 1995). The repair method might be very effective if the relationship between decision variables and constraints could be easily characterized. Developing a repair procedure is usually problem-dependent and so prior knowledge of the problem is required in order to design an efficient repair procedure. However, the characteristics of the solution space for real-world problems are often unknown. Sometimes repairing infeasible solutions can be as complex as solving the original problem hence it is also time-consuming when the problem is involved with complex constraints (Chootinan and Chen, 2006; Coello, 2002).

Some approaches such as Superiority of feasible points methods (Deb, 2000; Powell and Skolnick, 1993) and Multi-objective optimization techniques (Cai and Wang, 2006; Coello, 2000a; Surry and Radcliffe, 1997) handle constraints and objectives separately.

Deb (2000) suggested a modification of Powell and Skolnick (1993) that requires no penalty parameters. This method uses a tournament selection operator, where two individual solutions are compared at a time using the following criteria:

- A feasible individual is always better than an infeasible individual.
- If both of the individuals are feasible, then the individual with lower objective function value is better (considering minimization problem).
- If both of them are infeasible, then the one with less constraint violation is better. The total constraint violation (CV) of an individual is considered here as the sum of absolute values by which the constraints are violated.

The results of Deb (2000) are very encouraging. However the main drawback of this scheme is that it is hard to maintain a reasonable proportion of infeasible and feasible solutions in the population, and the use of niching methods (Deb and Goldberg, 1989) combined with higher than usual mutation rates is apparently necessary to avoid stagnation (Cai and Wang, 2006; Coello, 2002).

Runarsson and Yao (2000) introduced a stochastic ranking method in their evolutionary strategy-based algorithm in which the fitness of each individual is determined through a stochastic ranking process. The ranking is achieved through a stochastic version of bubble sort, in which the individuals are compared only to the adjacent neighborhoods. The comparison is based on either the objective function or the constraint violation, randomly determined by a user-specified probability parameter P_f. Although the method proved to be effective in solving a wide range of constrained optimization problems, it is also sensitive to the choice of probability parameter.

The Multi-objective algorithms (Cai and Wang, 2006; Coello, 2000a; Surry and Radcliffe, 1997) have been used in the solution of constrained single objective optimization problems, by treating the constraint violations as additional objectives. Generally the constraint violations and the objective function are optimized using multi-objective optimization methods. That means single-objective constrained optimization of $f(\boldsymbol{X})$ is redefined as a multi-objective optimization problem in which we will have $(m+1)$ objectives, where m is the total number of constraints. Then they apply any multi-objective optimization technique to solve. Although the idea of handling constraints through multi-objective optimization is very attractive, the approach appears less robust than for constrained single objective algorithms (Farmani and Wright, 2003). Finding the feasible solutions using the multi-objective technique is difficult since most of the time is spent on searching infeasible regions (Runarsson and Xin, 2005). For highly constrained problems, simply considering constraints as objectives might not introduce enough pressure to direct the search toward the region of the optimum (Farmani and Wright, 2003).

Different hybridization of algorithms has been introduced in recent times. Kim and Myung (1997) presented a two-phase evolutionary programming method. An

evolutionary algorithm is used to optimize the function in the first phase. In the second phase, Lagrange multipliers are used to place emphasis on the violated constraints whenever the best solution does not fulfill the constraints.

2.5 Multi-agent Systems (MAS)

In the last decade, Agents and multi-agent systems open a new era of analyzing, designing, and implementing complex systems. The technologies, methods, and theories of agent and multi-agent systems are currently contributing to many diverse domains including information retrieval, user interface design, robotics, electronic commerce, computer mediated collaboration, computer games, education and training, smart environments, ubiquitous computers, and social simulation (Zhang and Zhang, 2004).

Multi-agent systems are composed of multiple interacting autonomous computing elements, known as agents. According to Wooldridge and Jennings (1995) an agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives. Nicholas (2001) elaborated this definition as: agents are clearly identifiable problem-solving entities with well-defined boundaries and interfaces; situated (embedded) in a particular environment over which they have partial control and observability—they receive inputs related to the state of their environment through sensors and they act on the environment through effectors; designed to fulfill a specific role—they have particular objectives to achieve; autonomous—they have control both over their internal state and over their own behavior; capable of exhibiting flexible problem-solving behavior in pursuit of their design objectives—being both reactive (able to respond in a timely fashion to changes that occur in their environment) and proactive.

In MAS, each agent has incomplete information or capabilities for solving the problem, thus each agent has a limited viewpoint, and there is no global system control. The data is decentralized and the agents are connected thorough different schemes, usually following mesh and hierarchical structures (Oprea, 2004). Multi-agent systems are ideally suited to representing problems that have multiple problem solving methods,

multiple perspectives and/or multiple problem solving entities. Such systems have the traditional advantages of distributed and concurrent problem solving, but have the additional advantage of sophisticated patterns of interactions (Jennings *et al*., 1998).

Nicholas (2001) has defined the canonical view of a multi-agent system shown in Figure 2.5. In the canonical view of a multi-agent system, it can be seen that adopting an agent-oriented approach to software engineering means decomposing the problem into multiple, autonomous components that can act and interact in flexible ways to achieve their set objectives. The key abstraction models that define the agent-oriented mind-set are agents, interactions, and organizations. Finally, explicit structures and mechanisms are often used to describe and manage the complex and changing web of organizational relationships that exist between the agents.

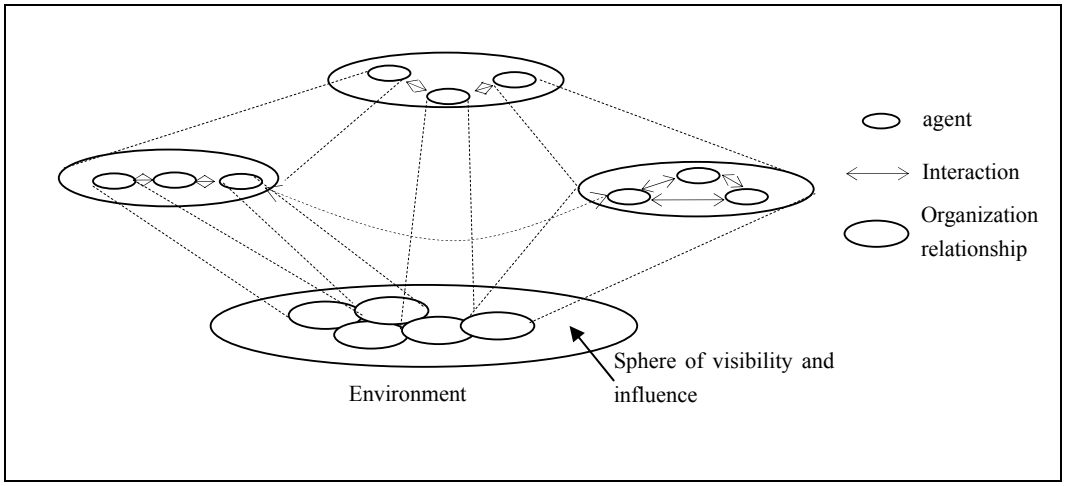


Figure 2.5: Canonical view of a Multi-agent system

2.5.1 Characteristics of Multi-agent Systems

The main characteristics of a multi-agent system (Vlassis, 2007) that distinguish it from other systems are given below:

Agent design: The agents involved in MAS can be heterogeneous where the design differences of agents may involve the hardware or the software. The agents can be homogeneous which are designed in an identical way and have a priori the same

capabilities. Agent heterogeneity can affect all functional aspects of an agent from perception to decision making, while in single-agent systems the issue is simply nonexistent.

Environment: The agents may need to face a static (time invariant) or dynamic (nonstationary) environment.

Control: There is no central control or process that collects information from each agent and then decides what action each agent should take. The decision making of each agent lies to a large extent within the agent itself.

Knowledge: In MAS, the levels of knowledge of each agent about the current world state can differ substantially. Each agent must consider the knowledge of each other agent in its decision making.

Communication: Agents may need to communicate with each other in several cases, for instance, for coordination among cooperative agents or for negotiation among self-interested agents.

2.5.2 Advantages of MAS

MAS involves a set of autonomous agents working together to solve problems that are beyond the capabilities of individual agents. Some of the advantages of using MAS technology in large software systems are (Sycara, 1998):

- Computational efficiency due to the concurrency of computation.
- Reliability, as the whole system can undergo a 'graceful degradation' when one or more agents fail.
- Extensibility, as the number and the capabilities of agents working on a problem can be altered.
- Robustness, the system's ability to tolerate uncertainty, because suitable information is exchanged among agents.
- Maintainability, it is easier to develop and maintain modular software than a monolithic one, MAS ensures easy development and reusability

2.6 Agent-based Evolutionary Algorithms (AEAs)

With the development of modern technologies, it creates demand for automated systems that solve more complex problems utilizing information from different sources. Recently different computational intelligence techniques such as evolutionary algorithms, fuzzy logic, neural networks are incorporated into agents to solve these complex problems. An agent-based evolutionary system is such a computationally intelligent system that may be considered as an extension to classical evolutionary algorithms.

The key idea of AEA is to incorporate the Evolutionary algorithms and MAS to exploit their combined strength. There exist different approaches to incorporate them. The first approach is the incorporation of evolutionary processes into a multi-agent system (MAS) at a population level (Kisiel-Dorohinicki, 2002). This approach is basically a population based evolutionary algorithm (EA), for solving complex decision problems, where the individuals of the population are defined as agents (Siwik and Kisiel-Dorohinicki, 2006; Zhong *et al.*, 2004). The evolutionary process realized in the multi-agent systems presents new possibilities such as: agents can act independently and in consequence social relations in agents' population may be developed. The evolution process is decentralized and is performed with no common cadence. Perception of the environment by agents and social relations enable rivalry and competition among agents that assure decentralized process of agents' selection. With the incorporation of evolutionary concept, besides interaction mechanisms typical for MAS, agents are able to reproduce new offspring and die (eliminated from the system). A decisive factor of an agent's activity is its fitness, expressed by amount of possessed non-renewable resource called life energy. The Selection process is realized in such a way that agents with high energy are more likely to reproduce, while low energy increases possibility of death. Some AEA architecture designs have been proposed by Kisiel-Dorohinicki (2002) and Byrski and Schaefer (2009). There are also several algorithms (Alkemade *et al.*, 2005; Chira *et al.*, 2008; Dobrowolski *et al.*, 2001; Siwik and Kisiel-Dorohinicki, 2006; Zhong *et al.*, 2004) based on this concept.

In the second type of agent-based evolutionary algorithms, the MAS incorporates an evolutionary algorithm as a search technique to improve certain functionality of some selected agents or the entire system. For example, a MAS may contain different types of agents, and one or more types of these agents deal with some optimization tasks (Choi *et al.*, 2001; Liu and Frazer, 2002; Meng *et al.*, 2007). More applications of this type of AEA are discussed in section 2.7.3.

2.7 Application of Intelligent Systems

So far some well-known intelligent techniques have been briefly discussed. This section will provide an idea about the application areas of three intelligent systems; GAs, MAS and AEA.

2.7.1 Application Areas of GAs

EAs, particularly GAs, have received a lot of attention regarding their potential as optimization techniques for complex numerical functions (Michalewicz, 1995). GAs have been successfully applied in solving different types of problems. Some of the noteworthy areas to which GAs have been put are discussed below.

2.7.1.1 General Optimization

Traveling Salesman Problem: The Traveling Salesman Problem (TSP) is one of the most important and well known combinatorial optimization problems, since it is simple to state but difficult to solve (Hung Dinh *et al.*, 2007). GAs have been applied to the TSP due to their global ability for problems with huge search spaces. The earliest attempts at applying GAs to the TSP are pure GAs such as (Goldberg and R. Lingle, 1985; Grefenstette *et al.*, 1985; Whitley *et al.*, 1991). Hybrid GAs have extended its performance to successfully solve large scale TSP problems (Helsgaun, 2000; Merz and

Freisleben, 1997; Merz and Freisleben, 2001).

Routing: The Vehicle Routing Problem (VRP) becomes more important with the development of modern logistics (Yueqin *et al.*, 2007). A typical vehicle routing problem involves simultaneously determining the routes for several vehicles from a central supply depot to a number of destinations (customers) and returning to the depot, without exceeding the capacity constraints of each vehicle. Applications of genetic algorithms to VRPs incorporating time windows have been reported by Thangiah *et al.* (1991), Potvin and Bengio (1996), and Thangiah (1995). Applications of GAs have also been reported for multi-depot routing problem (Ombuki-Berman and Hanshar, 2009; Thangiah and Salhi, 2001), for Dynamic vehicle routing (Hanshar and Ombuki-Berman, 2007) and a school bus routing problem (Thangiah and Nygard, 1992). Some hybrid approaches to vehicle routing using GAs have also been reported by (Jeon *et al.*, 2007; Potvin *et al.*, 1996).

Telecommunications networks: Telecommunications networks are interconnected by routers. Each router has a routing table, which specifies the next node in a route to a specified destination according to a routing path. The routing tables are produced by routing algorithms. The objective is to maximize the network utilization, and minimize the transmission delay and data loss. Other objectives required by the networks may be reliability, cost, and traffic load balancing in the network. This is an NP-hard optimization problem (He and Mort, 2000). Over recent years, several researchers have applied genetic algorithms to telecommunications routing problems. Pioneering papers using GAs include Cox *et al.* (1991), Davis *et al.* (1993), and Pan and Wang (1991). Recently different types of GAs (Bentall *et al.*, 1997) (for heavily loaded networks) including hybrid GAs (He and Mort, 2000; Sinclair, 1999) are also used in this area.

Job-shop Scheduling Problem: The Job-shop scheduling problem (JSP) is an extremely hard problem because it requires very large combinatorial search space and the precedence constraint between machines. This is an NP-hard optimization problem with multiple criteria: factors such as cost, tardiness, and throughput must all be taken into account, and job schedules may have to be rearranged on the fly due to machine

breakdowns, employee absences, delays in delivery of parts, and other complications, making robustness in a schedule an important consideration. GAs as presented in Chryssolouris and Subramaniam (2001), Ferrolho *et al.* (2007), Gen *et al.* (1994), Jensen (2003), and Lae-Jeoung and Cheol Hoon (1995) have successfully solved JSPs.

Timetabling Problem: The timetabling problem consists of allocating a number of events to a finite number of time slots (or periods) such that the necessary constraints are satisfied (Burke and Newall, 1999). The timetable problem in general is known to be NP-complete, meaning that no method is known to find a guaranteed-optimal solution in a reasonable amount of time (Burke *et al.*, 1995). Variants on genetic algorithms have appeared in the literature for timetabling problems (Burke and Newall, 1999; Burke *et al.*, 1996; Corne *et al.*, 1994; Paechter *et al.*, 1995).

2.7.1.2 Robotics

In recent years genetic algorithms have been applied to robot path and motion planning problems. Tian and Collins (2004) proposed a novel trajectory planning method for a robot manipulator whose workspace includes several obstacles. To generate the robot's trajectory Tian and Collins (2004) developed a genetic algorithm to search for valid and optimal solutions to the trajectory in task space. Yano and Toyoda (1999) applied a genetic algorithm to solve the position and movement of an end-effector on the tip of a two-joint robot arm. Shintaku (1999) developed a simple method based on a genetic algorithm, where a polynomial approximates time histories of the trajectory in joint space. Pack *et al.* (1996) developed a method to search for valid solutions in configuration space based on a genetic algorithm. Davidor (1991) as well showed how to apply genetic algorithm techniques to the task of planning the path which a robot arm is to take in moving from one point to another.

2.7.1.3 Aerospace Engineering

In the field of aeronautical engineering, a series of studies for aerodynamic design with genetic algorithms have been carried out. In the aerodynamic wing optimization problem, the objectives are to minimize aerodynamic drag at supersonic cruising speeds, minimize drag at subsonic speeds, and minimize aerodynamic load (the bending force on the wing) (Obayashi and Sasaki, 2004). These objectives are mutually exclusive, and optimizing them all simultaneously requires tradeoffs to be made. Several multi-objective GAs have been applied to the aerodynamic wing optimization problem (Obayashi and Sasaki, 2004; Obayashi *et al.*, 2000; Sasaki *et al.*, 2001).

Genetic Algorithms have successfully generated low-earth orbit sparse coverage satellite constellations that appear to outperform traditionally developed constellations (Williams *et al.*, 2001). The objective of these constellations is to minimize the maximum revisit time over a latitude band of interest. Williams *et al.* (2001) applied a multi-objective genetic algorithm to the task of spacing satellite orbits to minimize coverage blackout. Deb *et al.* (2007) also presented the development of a multi-objective optimization software (GOSpel) for finding optimal interplanetary trajectories between any two planets for a dual minimization of travel time and launch velocity which is directly related to the fuel consumption.

2.7.1.4 Economics and Finance

The areas of economics and finance, with special reference to predictability issues related to stock and foreign exchange markets, seem to attract increasing interest during the last few years (Andreou *et al.*, 2002). More and more traders now rely on genetic algorithms, neural networks, chaos theory, and other computerized decision-making approaches to help them develop winning investment strategies (Richard, 1994). Mahfoud and Mani (1996) presented a genetic algorithm based system and applied it to the task of predicting the future performances of individual stocks. Andreou *et al.* (2002) has presented a new hybrid algorithm based on a GA for the evolution of the architecture of Multi-Layered Perceptron neural networks and a localized version of the

Extended Kalman Filter for the training. The application of this algorithm on the task of exchange rate forecasting of different currencies was very positive and encouraging. Another important research issues in finance is building effective corporate bankruptcy prediction models, because they are essential for the risk management of financial institutions. Ahn and Kim (2009) proposed a GA based approach to enhance the prediction performance of case-based-reasoning for the prediction of corporate bankruptcies.

2.7.1.5 Electrical Engineering and Circuit Design

Efforts using techniques from evolutionary computation (specially GAs) for different analog circuit design automation have appeared over the last few years (Xing *et al.*, 2005). Genetic algorithms have been successfully applied to select filter component sizes (Horrocks and Khalifa, 1994), to select filter topologies (Grimbleby, 1995), to design operational amplifiers using a small set of topologies (Kruiskamp, 1996), and to automatically generate circuit designs (Lohn and Colombano, 1999). Genetic algorithms have also been used to evolve antennas with pre-specified properties, including wire antennas (Kuwahara, 2005; Linden, 1997), patch antennas (Villegas *et al.*, 2004), and antenna arrays (Buckley, 1996).

2.7.1.6 Pattern Recognition and Data Mining

The use of GA for pattern recognition has been widely studied (Man *et al.*, 1996). Smith and Gemperline (2000) have designed a wavelength selection and optimization of pattern recognition methods using a genetic algorithm. Forrest *et al.* (1993) presented an immune system model based on genetic algorithms to study the pattern-recognition processes and learning that take place at both the individual and species levels in the immune system. Roth and Levine (1992) applied GA, based on a minimal subset representation, to perform primitive extraction from geometric sensor data. Tsang (1995) proposed a GA based technique for matching images of object shapes that have

been subject to affine transformation caused by variations in the camera position. A Faceprint system was designed in New Mexico State University (Caldwell and Johnston, 1991) for reproducing the feature of a suspected criminal's face. A genetic algorithm is used to generate binary reference functions for optical pattern recognition and classification by (Mahlab *et al.*, 1991). Hybrid evolutionary learning algorithms have also been designed to synthesize a complete multiclass pattern recognition system. For example, Rizki *et al.* (2002) designed a Hybrid Evolutionary Learning for Pattern Recognition (HELPR) that blends elements of evolutionary programming, genetic programming, and genetic algorithms to perform a search for an effective set of feature detectors.

Li *et al.* (2007) proposed a data mining genetic algorithm, to mine the association rules from an image database. Tzung-Pei *et al.* (2008) introduced a genetic algorithm based framework for finding membership functions suitable for data mining problems.

2.7.2 Application Areas of MAS

Agent-based systems have been applied in solving a wide variety of problems. Major applications of agent-based systems are as follows: manufacturing, process control, telecommunication systems, air traffic control, traffic and transportation management, information filtering and gathering, electronic commerce, business process management, entertainment and medical care (Jennings *et al.*, 1998). A brief description of these areas are given below, for a comprehensive review in all of these areas see (Chaib-draa, 1995; Jennings *et al.*, 1998; Jennings and Wooldridge, 1998; Parunak, 1999).

Manufacturing: MAS have been applied in different areas of manufacturing, namely manufacturing control (Parunak, 1999), configuration design of manufacturing products (Darr and Birmingham, 1994), collaborative design (Cutkosky *et al.*, 1993), scheduling and controlling manufacturing operations (Oliveira *et al.*, 1997; Sprumont and P.Muller, 1997), controlling a manufacturing robot (Overgaard *et al.*, 1996), and determining production sequences for a factory (Wooldridge *et al.*, 1996).

Process Control: Agent-based systems have been used for electricity transportation management, monitoring and diagnosing faults in nuclear power plants (Huaiqing and Chen, 1997), spacecraft control (Francois *et al.*, 1992), and climate control (Clearwater *et al.*, 1996).

Telecommunications: Telecommunication systems are large, distributed networks of interconnected components which need to be monitored and managed in real-time (Jennings *et al.*, 1998). The feature interaction problem (Griffeth and H.Velthuijsen, 1994), Network Control (Schoonderwoerd *et al.*, 1997), transmission and switching (Nishibe *et al.*, 1993), and network management (Adler *et al.*, 1989) are some examples for which agent-based systems have been constructed.

Air Traffic Control and Transportation Systems: *OASIS* (Ljunberg and Lucas, 1992) presents a sophisticated agent-realised air traffic control system where agents are used to represent both aircraft and the various air-traffic control systems in operation. The domain of traffic and transportation management is well suited to an agent-based approach because of its geographically distributed nature. For example, Burmeister *et al.* (1997) describes a multi-agent system for implementing a future car pooling application.

Information Management: With the development of the richness and diversity of information available to us in our everyday lives, the need to manage this information has grown. The lack of effective information management tools has given rise to an information overload problem (Jennings *et al.*, 1998). Agent-based systems opened a new era in this area. Electronic mail filtering agents (Pattie, 1994) can learn to prioritise, delete, forward, sort, and archive mail messages on behalf of a user. A multi-agent system (Sycara *et al.*, 1996) has been designed to integrate information finding and filtering in the context of supporting a user to manage his financial portfolio. A personal assistant (Chen and Sycara, 1998) that learns user interests and on the basis of these compiles a personal newspaper, a personal assistant agent for automating various user tasks on a computer desktop (Caglayan *et al.*, 1997), and a web browsing assistant (Lieberman, 1995) are some other applications in this area.

Electronic Commerce: Some commercial decision making is already placed in the hands of agents. An example is realizing the marketplace by creating "buying" and "selling" agents for each good to be purchased or sold respectively (Chavez and Kasbah, 1996).

Medical Applications: The interest in applying agent technology to medical applications has been a growing one (Cortés *et al.*, 2008). From such seminal and inspiring work as agent based patient monitoring by Hayes-Roth *et al.* (1989) and health care by Huang *et al.* (1995), the use of agents in medical science has been continuously evolving and covering more aspects. Intelligent agents are normally used to observe the current situation and knowledge base, then support the expert's decision-making on an action consistent with the domain they are in, and finally perform the execution of that action on the environment. For example, Laleci *et al.* (2008) designed MAS for providing a Clininical Desicion Support system for remote monitoring of patients at their homes and at the hospital, to decrease the load on medical practitioners and also healthcare costs. HealthAgents designed by Lluch-Ariet *et al.* (2008) improves the classification of brain tumors through multi-agent decision support over a secure and distributed network of local databases or Data Marts.

2.7.3 Application Areas of Agent-based EA

AEA is a relatively newer area than GAs or MAS. To solve complex real world problems, AEA opens a new era to incorporate intelligent techniques like EAs and MAS. A number of agent-based hybrid evolutionary algorithms have appeared in the literature for solving different types of problems.

The applications of the first type of AEA (incorporation of evolutionary processes into a multi-agent system at population level, discussed in section 2.6) are mostly in solving different types of optimization problems. For example, Dobrowolski *et al.* (2001) used an evolutionary multi-agent system for solving unconstrained multi-objective problems. Socha and Kisiel-Dorohinicki (2002) developed an agent-based

evolutionary approach to search for a global solution in the pareto sense for multi-objective optimization. Siwik and Kisiel-Dorohinicki (2006) developed a semi-elitist evolutionary multi-agent system and they solved the so called MaxEx multi-objective problem. Niching techniques are aimed at maintaining the diversity through forming subpopulations for evolutionary algorithms in multi-modal domains. Similar techniques have applied to evolutionary multi-agent systems in Dreżewski and Kisiel-Dorohinicki (2006). Job-shop scheduling problems are solved with multi-agent evolutionary algorithms by Zhong *et al.* (2005) and Yan *et al.* (2004). Zhong *et al*. (2004) used a MultiAgent Genetic Algorithm (MAGA) for solving unconstrained global numerical optimization problems. Liu *et al*. (2006) used a multiagent evolutionary algorithm for constraint satisfaction problems. Chira *et al*. (2008) proposed a geometric agent-based model for several difficult unimodal and multimodal real-valued functions with many dimensions. Byrski and Schaefer (2009) applied their evolutionary agents systems to a difficult global optimization problem (optimization of the artificial neural network architecture).

The second type of AEA usually applies EA as a part of the decision making process. For example, Lim and Zhang (2002) designed an intelligent multi-agent system with GA which integrates process planning and production scheduling, in order to increase the flexibility of manufacturing systems in coping with rapid changes in the market. This system consists of various autonomous agents who have the capability of communicating with each other and making decisions based on their knowledge.

Pendharkar (2007) designed a multi-agent system for manufacturing flow shop scheduling, where the agents contained a knowledge base of dispatching rules and a genetic algorithm was used that learns new dispatching rules over time. Cetnarowicz *et al.* (1996) proposed a new technology of designing and building agent systems based on genetic methods, and a draft concept of a model-based approach to such systems. They have applied this technology to a self-developing prediction system. Smith *et al.* (1999) incorporated EA-based mechanisms into agent-based decentralized business applications. Liu and Frazer (2002) showed the design process as generative and evolutionary processes that are implemented by a group of cooperative agents.

Usually it is difficult to design controllers for multi-agent systems without a comprehensive knowledge about the system. To overcome this limitation, Jeong and Lee (1997) used a genetic algorithm to discover rules that govern emergent cooperative behavior. They proposed a self-organizing genetic algorithm for automating the discovery of rules for multi-agents playing soccer.

Yang *et al.* (2006) used a GA based multi-objective optimization technique NSGA-II to decide on the composition of forces using a complex land combat multi-agent scenario planning tool. Sahin *et al.* (2008b) introduced a Force-based Genetic Algorithm (FGA) for self-spreading mobile nodes deployed over an unknown territory. Wireless mobile nodes adjust their speed and direction using a genetic algorithm, where each mobile node exchanges its genetic information (of speed and direction) encoded in its chromosomes with the neighboring nodes. The improved version of FGA is presented in Sahin *et al.* (2008a).

2.8 Chapter Summary

In this chapter, different types of optimization problems and their solution methodologies have been discussed. Most real world optimization problems are constrained. In solving these constrained optimization problems, solution approaches are needed to satisfy different types of linear or nonlinear, equality or inequality or both constraints. Conventional methods are unlikely to provide quality solutions within reasonable amount of time for real world complex constrained optimization problems. During the last decades, several heuristic methods have been proposed to solve these problems. Among them genetic algorithms is one of the most successful in solving different types of optimization problems. To handle the constraints, different techniques have been proposed to guide the search process of GAs. However several algorithms in the literature have struggled while solving COPs, especially when the feasible space is very tiny compared to the whole search space. Interactions of the constraints and existence of equality constraints are some reasons behind that. Furthermore traditional GAs suffers from slow convergence to locate a precise enough solution because of their

failure to exploit local information, and hence they are not well suited for fine tuning.

Some hybridized algorithms such as agent-based evolutionary algorithms have appeared in the literature, incorporating the EAs with intelligent agent systems. These algorithms show enhanced performance in solving optimization problems like unconstrained global optimization problems, constraint satisfaction problems, and multi-objective problems. However, good performance in solving constrained optimization problems with agent-based evolutionary algorithms is, to the best of our knowledge, yet to come in the literature. This motivates to design a new agent-based evolutionary algorithm for solving COPs, and different techniques to enhance the performance.

.

Chapter 3

Genetic Algorithms in Solving COPs

Many real world decision processes require solving Constrained Optimization Problems (COPs). In this chapter, a simple genetic algorithm is implemented for solving COPs. The performance of the algorithm is investigated and analyzed using a set of state-of-the-art test problems. The experimental studies show the limitations of genetic algorithms in solving COPS; this is the motivation for improving the algorithm in this thesis.

3.1 Introduction

A large number of real world optimization problems are nonlinear and need to satisfy different constraints. These constraints may involve equality, inequality or both types. The objective function and the constraints, which may be linear or nonlinear, are here assumed to be continuous (for details see Chapter 2). The aim of this thesis is to develop effective solution approaches for solving these constrained optimization problems.

Genetic Algorithms (GAs) are the most widely used approaches to computational evolution and solving different types of optimization problems (Davis, 1991). In the beginning of the research for this thesis, it would be interesting to see the performance of a Simple Genetic Algorithm (SGA) in solving COPs.

In this chapter, a genetic algorithm is implemented and its performance is investigated in solving the COPs. To design the SGA, well-known crossover and

mutation operators, and a suitable constraint handling technique are used. A set of well-known test problems is used to investigate the performance. In pursuit of maximum performance further investigation is made to see the performance of the algorithm by changing different parameters such as the probability of crossover, the probability of mutation and the population size, and additional fitness evaluations. The experimental result shows the limitations of SGA in solving different types of COPs. The knowledge gained in this chapter helps in designing improved solution approaches for solving COPs.

The rest of this chapter is organized as follows. The next section describes the design of a simple genetic algorithm and its components. The experimental results and the effects of different components of the algorithm are described in section 3.3. Finally, section 3.4 concludes the chapter and discusses the challenges to be addressed in the following chapters.

3.2 Simple Genetic Algorithm

Over the last few decades genetic algorithms have been widely employed as effective search and optimization methods in numerous fields of applications (Safe *et al.*, 2004). In this chapter a simple genetic algorithm is designed for solving COPs. The design of the SGA and its components are discussed in this section.

In the SGA, the solutions for the initial population $P^{t=0}$ are randomly generated within the boundary of each decision variable. The individual solutions are evaluated and ranked based on their fitness. A set of individuals is selected as parents to produce offspring using crossover with probability of crossover P_C. This new population is called C^t. A percentage of individual solutions from C^t with P_M probability apply mutation. After mutation, the population C^t is called as $C^{t'}$. To generate a new population P^{t+1}, the parent population P^t is merged with the evolved child population $C^{t'}$ and then they are ranked based on fitness. The top ranked individuals form the next generation P^{t+1}. The process is continued until the termination condition is reached.

The main steps of the proposed algorithm are as follows.

Pseudo code: Simple Genetic Algorithm (SGA).

Set generation no. $t = 0$;
Generate the initial population P^t at random;
REPEAT
  Evaluate the fitness of each individual in P^t and rank them;
  Apply tournament selection on P^t to select the parents then apply crossover (with P_C probability) and generate C^t;
  Apply mutation on C^t with P_M probability and generate $C^{t'}$;
  Produce generation P^{t+1} from P^t and $C^{t'}$;
  Set $t = t + 1$;
UNTIL the terminating condition is reached.

The different components of the algorithm are discussed below.

3.2.1 Representation

As the search spaces of the optimization problems are continuous and the variables under consideration are real, in this research floating point/real-coded representation is used to represent the solutions.

3.2.2 Fitness Evaluation and Constraint Handling

For optimizing a constrained problem, the search technique should find not only the feasible solutions from the search space but also the optimal solutions. So, while evaluating solutions in solving COPs, attention should be given to both the objective function value and the constraint violations of the solutions. For each individual, the objective function value and total Constraint Violation (CV) are calculated. While

solving these problems the constraints are normalized in some cases. The total CV of an individual is considered here as the sum of absolute values by which the constraints are violated, however they are not normalized in this research. The pair-wise comparison (Deb, 2000) is used in ranking and selection, which indirectly handles the constraints. In the pair-wise comparison, the best infeasible individual is assigned worse fitness than the worst feasible individual. As such, while comparing two individuals an infeasible individual is penalized and a feasible individual is rewarded, so the constraints are handled indirectly. Details of this constraint handling technique are discussed in section 2.4.3.

3.2.3 Selection

As discussed in chapter 2, there are several opportunities for biasing the selection for mating. To design the SGA, tournament selection (Bäck *et al.*, 2000) is used. The tournaments are played between two individual solutions and the better solution is chosen as a parent. The other parent is also selected in the same way. It is shown in Goldberg and Deb (1991) that the tournament selection has better convergence and computational time complexity properties compared to any other reproduction operator that exists in the literature. Still tournament selection dominates in the practice of GA.

3.2.4 Crossover

With crossover, usually new offspring are generated with an expectation that they combine the best features from the parents. Crossover is applied to a set of individuals, each selected with a probability P_C.

The Simulated Binary Crossover (SBX) operator proposed by Deb and Agrawal (1995) is used here. Based on the search features of the single-point crossover used in binary-coded genetic algorithms, the SBX operator respects the interval schemata processing, in the sense that common interval schemata of the parents are preserved in

the offspring (Deb and Agrawal, 1995). SBX operator performs well in solving problems having multiple optimal solutions with a narrow global basin (Deb, 2000; Deb and Agrawal, 1995). Details of SBX are discussed in section 2.4.2.

3.2.5 Mutation

After crossover, the mutation operator is applied to a certain percentage of individuals (with mutation probability P_M). In SGA, the parameter-based mutation operator is used, which allows the selected individual to explore its neighborhood. In chapter 2 this mutation is discussed, and more details of it can be found in (Deb, 2000; Deb and Goyal, 1996). Deb (2000) has reported that for solving optimization problems in real space with arbitrary feasible regions shape, real-coded GAs with SBX and a parameter-based mutation operator have been found to be useful.

3.3 Experimental Studies

In this section, the performance of simple genetic algorithms in solving a set of well-known benchmark problems is studied. Then the effects of different components on the performance of the algorithm are analyzed in quest of improved performance.

3.3.1 Benchmark Problems

The performance of SGA is evaluated using a set of 13 benchmark problems, studied by Michalewicz and Schoenauer (1996), Koziel and Michalewicz (1999), and further studied by Runarsson and Yao (2000) and others. The benchmark problems include different forms of objective function (linear, quadratic, cubic, polynomial, nonlinear) and different number of variables (n). The problems g02, g03, g08, and g12 are maximization problems and the other nine are minimization problems. The maximization problems are transformed into equivalent minimization problems. The

main characteristics of the benchmark problems are presented in Table 3.1, and the detailed mathematical representations are provided in the Appendix A.

The equality constraints of g03, g05, g11, and g13, $h_j(\boldsymbol{X})=0$ have been converted into inequality constraints $-\delta \le h_j(\boldsymbol{X}) \le \delta$, where δ is a small tolerance value. The use of δ allows the algorithm to find some feasible solutions easily by increasing the solution space. This is a common practice with equality constraints in EAs (Deb, 2000; Elfeky *et al.*, 2006).

Table 3.1: Characteristics of the test problems.

Fn	(n)	Obj. Fuc.	ρ	LI	NI	LE	NE	AC	Optimal
g01	13	Quadratic	0.0111%	9	0	0	0	6	-15.000
g02	20	Nonlinear	99.8474%	0	2	0	0	1	-0.803619
g03	10	Polynomial	0.0000%	0	0	0	1	1	-1.000
g04	5	Quadratic	52.1230%	0	6	0	0	2	-30665.539
g05	4	Cubic	0.0000%	2	0	0	3	3	5126.498
g06	2	Cubic	0.0066%	0	2	0	0	2	-6961.814
g07	10	Quadratic	0.0003%	3	5	0	0	6	24.306
g08	2	Nonlinear	0.8560%	0	2	0	0	0	-0.095825
g09	7	Polynomial	0.5121%	0	4	0	0	2	680.630
g10	8	Linear	0.0010%	3	3	0	0	6	7049.331
g11	2	Quadratic	0.0000%	0	0	0	1	1	0.750
g12	3	Quadratic	4.7697%	0	9^3	0	0	0	-1.000
g13	5	Nonlinear	0.0000%	0	0	0	3	3	0.053950

ρ = Ratio between the feasible space and the search space, LI = Linear Inequalities, NI = Nonlinear Inequalities, LE = Linear Equalities, NE = Nonlinear Equalities, AC = Active Constraints.

3.3.2 Experimental Results and Discussions

In this study the performance of SGA is investigated in solving constrained optimization problems. As the crossover operator is mainly responsible for the search aspect of genetic algorithms, even though mutation operator is also used for this purpose sparingly (Deb, 1999), a high probability for crossover (P_C = 0.90) is used (as

in Deb (2000)) and a probability for mutation (P_M = 0.2). The number of fitness evaluations is set to 350,000 as in (Elfeky *et al.*, 2006; Runarsson and Yao, 2000), which allows a maximum of 3500 generations with a population size of 100.

The algorithm is executed for 30 independent runs with different seeds to solve each of the test problems. The best, median, mean, standard deviation (st.dev.), and worst results, as well as execution time, for the test problems are given in Table 3.2. An '×' in the Table indicates that the algorithm did not find any feasible solution.

Table 3.2: Statistics for 30 independent runs of the SGA.

Fn	Optimal	Best	Median	Mean	St.Dev.	Worst	Time(s)
g01	-15.000	-14.998	-13.815	-13.990	9.41E-01	-11.780	4.70
g02	-0.803619	-0.782757	-0.739116	-0.724945	3.93E-02	-0.632602	17.43
g03	-1.000	×	×	×	×	×	4.14
g04	-30665.539	-30664.743	-30662.894	-30662.347	2.55E+00	-30653.675	2.92
g05	5126.498	×	×	×	×	×	3.52
g06	-6961.814	-6945.396	-6920.632	-6920.196	1.61E+01	-6888.569	3.45
g07	24.306	25.615	27.755	28.310	2.41E+00	36.594	6.50
g08	-0.095825	-0.095825	-0.095825	-0.095825	5.32E-09*	-0.095825	2.93
g09	680.630	680.808	681.648	681.821	6.50E-01	683.944	5.32
g10	7049.331	7166.255	7823.128	8376.182	1.45E+03	13284.257	3.12
g11	0.750	×	×	×	×	×	2.55
g12	-1.000	-1.000	-1.000	-1.000	1.81E-10*	-1.000	50.57
g13	0.053950	0.457442	0.922264	1.031979	6.59E-01	3.854752	4.29

*Though the best, worst, median, and mean results are the same, standard deviation is positive due to rounding error.

From the results given in Table 3.2, it can be seen that SGA has achieved the optimum in two problems (g08 and g12). In four problems (g01, g04, g06 and g09), the best results are within 1% of the optimum. The achieved results are within 10% of the optimum in three test problems (g02, g07 and g10). In one problem (g13), the result is far away (747.90%) from the optimum, and for three problems (g03, g05 and g11) SGA could not find any feasible solutions. The mean results are also very far from the

optimum in most of the problems; on average they are 186.54% from the optimum. In some problems the percentage deviation of the achieved mean from the optimum is very high, such as 1812.84% for g13, 18.82% for g10, 16.47% for g07, 9.79% for g02 and 6.73% for g01.

For g12, SGA achieved optimum in early generations for every run, and so SGA is allowed to execute maximum 500 generations for this problem. This problem involves 729 constraints and took SGA the longest time to solve it (on average 50.57 seconds).

Though the performance of SGA is not very pleasing, its simplicity of design means that SGA converges prematurely and is fast in solving the test problems. On average it took only 8.57 seconds to solve each problem.

The performance of SGA is better for problems having considerable feasible space ($\rho > 1\%$) than for problems with tiny feasible space. For calculating the ratio of feasible space over the search space (ρ) a metric is used, suggested by Michalewicz and Schoenauer (1996): $\rho = |F|/|S|$ where $|S|$ is the number of random solutions generated (1,000,000 in this case), and $|F|$ is the number of feasible solutions found (out of the total randomly generated solutions). For the problems with considerable feasible space, for example g02 (ρ =99.85%), g04 (ρ =52.12%), g12 (ρ =4.77%), the achieved best results are on average within 0.866% of the optimum and the mean results are on average within 3.267% of the optimum. On the other hand, in those problems with tiny feasible space ($\rho < 1\%$) the performance of SGA is not so good. In three problems, it could not find any feasible solutions. For the other seven problems, the achieved mean results are on average 265.092% from the optimum. This shows SGA suffers in solving problems with tiny feasible space.

It is worth noting here that there are four problems (g03, g05, g11, and g13) involving equality constraints. The ratios of the feasible space over the search space for these problems are 0.00%. The existence of the equality constraints reduces the size of the feasible space. For this type of problems SGA is seriously deficient. The algorithm has found feasible solutions in only one problem, though the achieved best result is 747.90% away from the optimum.

The convergence curves of best and mean result of the population for 3500 generations with SGA for different problems are given in Figures 3.1–3.10 (g03, g05, and g11 are not shown, because SGA could not find any feasible solutions for these problems).

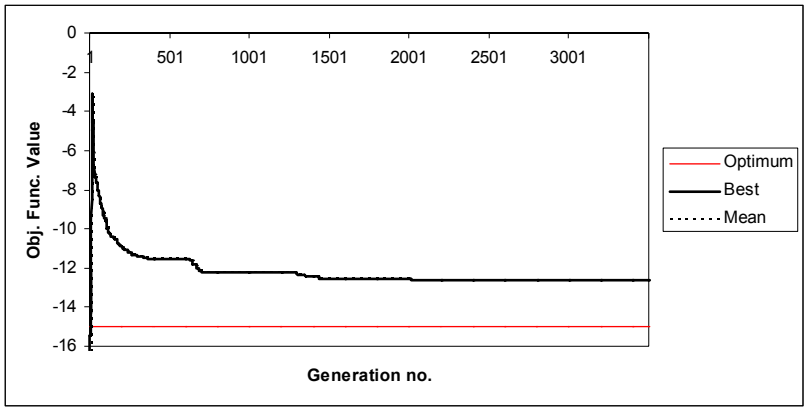


Figure 3.1: Convergence Curve of the best and mean objective value for problem g01.

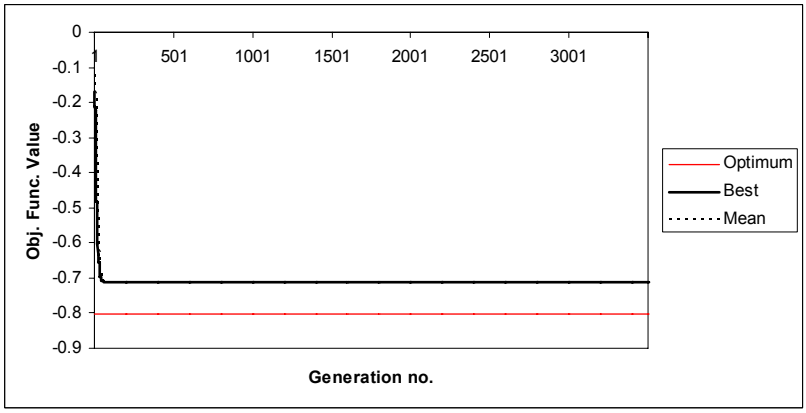


Figure 3.2: Convergence Curve of the best and mean objective value for problem g02.

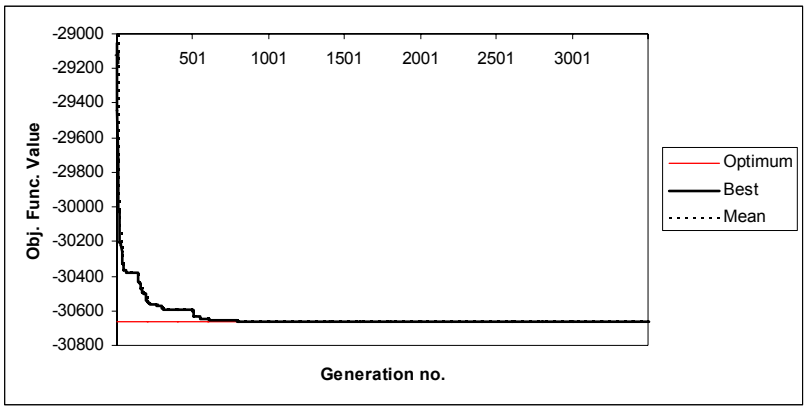


Figure 3.3: Convergence Curve of the best and mean objective value for problem g04.

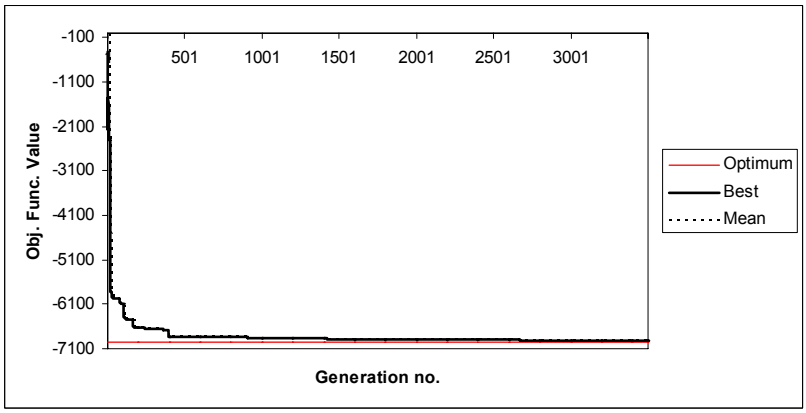


Figure 3.4: Convergence Curve of the best and mean objective value for problem g06.

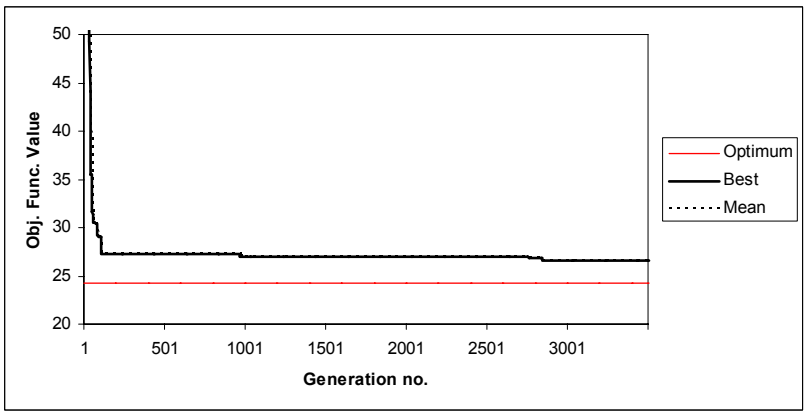


Figure 3.5: Convergence Curve of the best and mean objective value for problem g07.

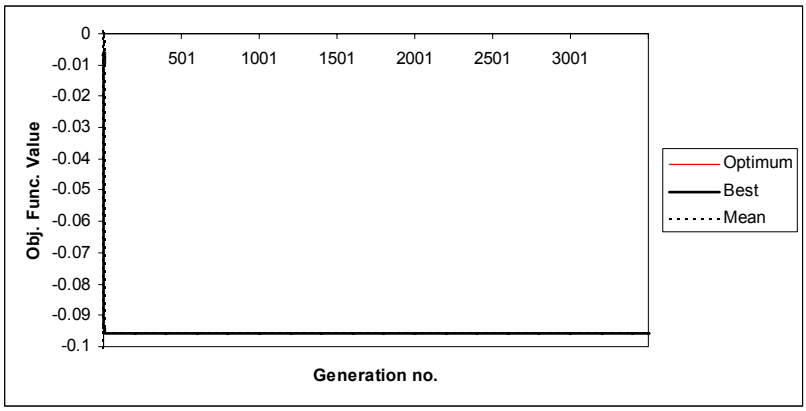


Figure 3.6: Convergence Curve of the best and mean objective value for problem g08

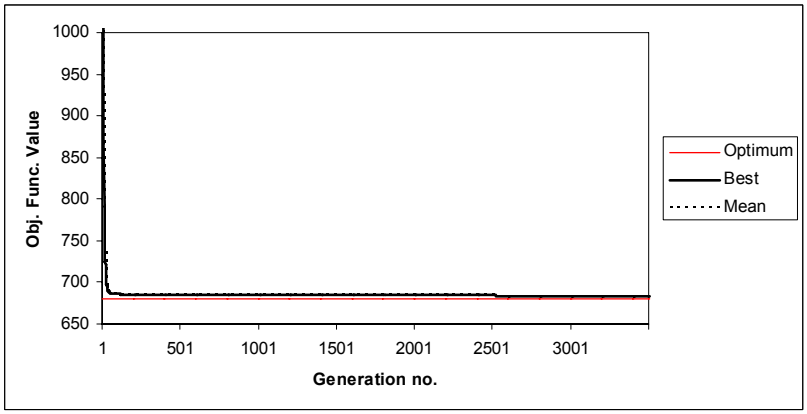


Figure 3.7: Convergence Curve of the best and mean objective value for problem g09.

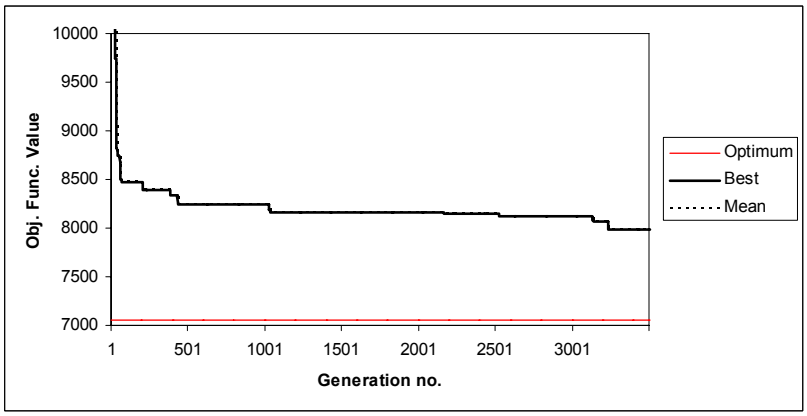


Figure 3.8: Convergence Curve of the best and mean objective value for problem g10.

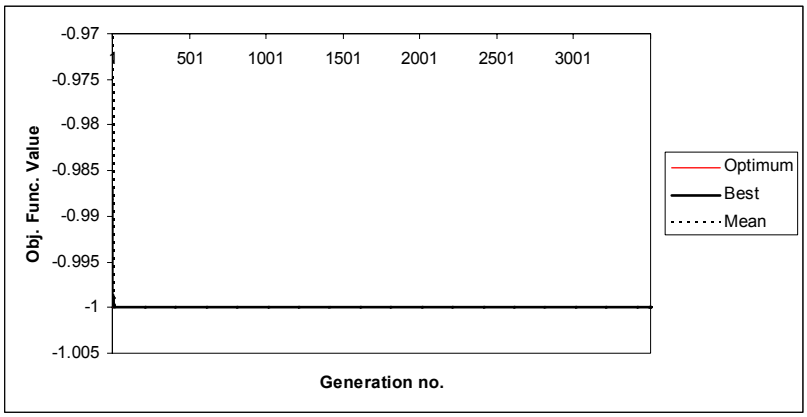


Figure 3.9: Convergence Curve of the best and mean objective value for problem g12.

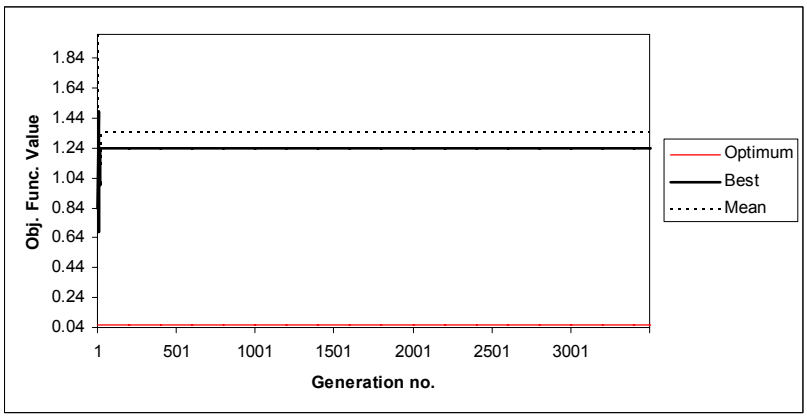


Figure 3.10: Convergence Curve of the best and mean objective value for problem g13.

3.3.3 Effects of Parameters and Operators

Genetic algorithms work efficiently when the right values for parameters such as probability of crossover, probability of mutation, and population size are chosen (Bingul *et al.*, 2000). When these parameters are set optimally, it is very beneficial as the GA would yield better or similar fitness values with similar or less computational cost.

In this section, a number of experiments are reported to analyze the effects of different parameters used in SGA e.g. the effect of probability of crossover (P_C), probability of mutation (P_M), population size. In the following four sub-sections, these

experiments are discussed in detail.

3.3.3.1 Probability of Crossover (P_C)

Crossover operator is considered as the main search operator of genetic algorithms. It is one of the most important features distinguishing it from other search algorithms (Tian, 2001). In this section, the effect of crossover on SGA with different probabilities is investigated.

The performance of the algorithm is tested (over 30 runs each) with values for P_C of (0.80, 0.85, 0.90, and 0.95) for all the test problems. The other parameters (P_M=0.20, population size = 100) remain the same. A higher value of P_C allows more offspring to be generated from the parents, which may help the algorithm to perform better. However, after a certain level, higher values of P_C may cause diversity reduction in the population. While analyzing the results, if we consider the best results, with lower P_C = 0.80 SGA achieved better result in only one problem (g13). With a large value of P_C (0.95) it only achieved better results in g06. For two problems g02 and g09 it achieved better results with P_C = 0.90, and with P_C = 0.85 SGA achieved better results in four problems (g01, g04, g07 and g10). The performance remains the same for all values of P_C in g08 and g12; in those problems, SGA achieved the optimum in early generations. For the other three problems SGA could not find any feasible solutions.

Sometimes, the best result may be an outlier for the population based stochastic algorithms. If we consider the mean results, with very high value of P_C (0.95) SGA achieved better results in g06. For two problems (g07 and g10) SGA performs better with lower P_C (0.80). In g01 and g04 SGA achieved better mean results using P_C = 0.85. For three problems (g02, g09, and g13) SGA achieved better results with P_C = 0.90.

After analyzing the results, we can conclude that for better performance of SGA we should use neither too low nor too high value of P_C. In this experiment, SGA achieved better results mostly using P_C in the range of (0.85 – 0.90). However, even with these

parameters SGA still suffers in solving the equality constrained problems. For three problems, it could not find any feasible solutions. The average deviation of the mean results is still more than 150% from the optimum results with all values of P_C.

Figure 3.11 shows the convergence curve of SGA using different values of P_C (0.80, 0.85, 0.90, and 0.95) for problem g04. In some problems, ρ is very high (e.g. g02, ρ =99.84%), in some other problems ρ is very low (0.00%). Here problem g04 is chosen as an example as for this problem ρ is 52.123% which is in between the extreme values. The curve shows that SGA using P_C = 0.90 converges more slowly but with better final performance.

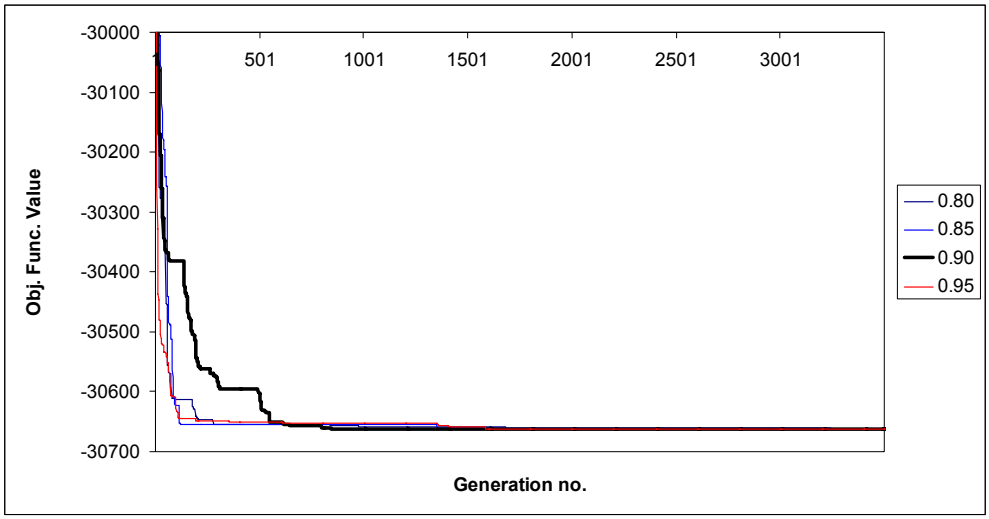


Figure 3.11: Convergence Curve of SGA using different values of P_C for problem g04.

3.3.3.2 Probability of Mutation (P_M)

In genetic algorithms, mutation is used to introduce genetic diversity from one generation of a population to the next generation. Mutation is known as the "background" operator in the genetic algorithm, and it has a full range of alleles so that the individuals are not trapped in local optima (Holland, 1975). Without mutation, the

evolution would be stagnated because no new variations are created (Soon *et al.*, 2008).

In SGA, after crossover a certain percentage of the individuals are randomly selected to apply mutation with a probability P_M. The performance of the algorithm (over 30 runs each) is tested with different values for P_M of (0.05, 0.1, 0.15, 0.20, 0.25, and 0.3) while keeping the other parameters the same (P_C = 0.90, population size = 100).

With higher values of P_M more individuals are allowed to apply mutation, ensuring more diversity in the population. Diversity is an important issue for the performance of any population based search algorithms. With the increase of P_M, SGA performs better up to a certain level, then the performance does not improve significantly. Considering the best results achieved by SGA with different P_M, SGA performed better with higher value of P_M. For g03 and g06 it has achieved better results using P_M=0.30. For some problems like g08 and g11 with all values of P_M, it achieves the same results. If we consider the mean values of the 30 runs, SGA achieves better results in g01, g02, g09 and g10 with P_M=0.25, and with P_M = 0.20 it achieves better results in g13. Though the performance of SGA has improved with different values of P_M, the overall performance of SGA is still not convincing. The average deviation of the mean results is still more than 180% from the optimum with any value of P_M.

Figure 3.12 shows the performance of SGA with different values of P_M for problem g04. It can be seen that the best results are achieved by SGA after P_M = 0.05. Then the results remain almost the same. However if we consider the mean results, it is very clear that the performance improves up to P_M = 0.25. After that, the performance does not improve significantly.

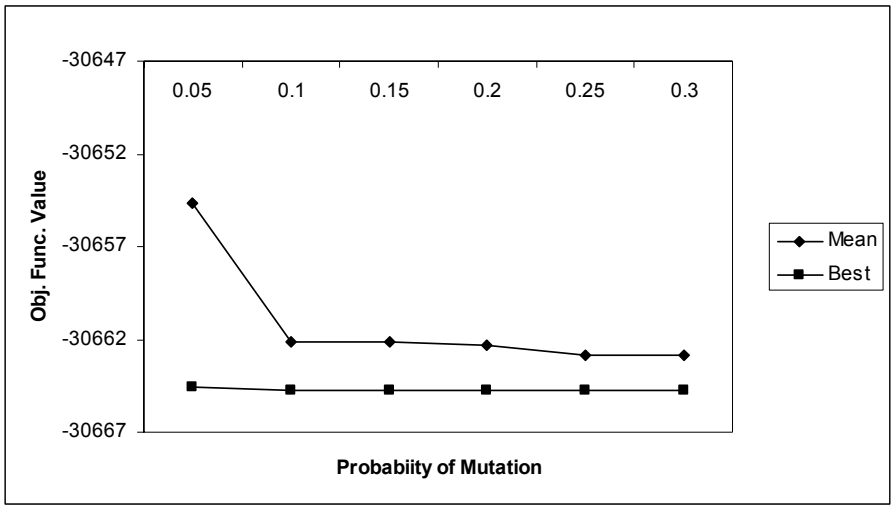


Figure 3.12: Effect of probability of mutation (P_M) on problem g04.

3.3.3.3 Population Size

Population size is another important parameter for the performance of GAs**.** If the population size is too small, then an insufficient number of individuals are sampled during the evolutionary process and the algorithm would not yield the best possible solution. On the other hand, if the population size is too large, the algorithm becomes inefficient as more tests are performed than necessary for each generation (Bingul *et al.*, 2000). In this section, the performance of SGA is tested with different population sizes. The aim of the experiment is to see how SGA behaves with different population sizes, whether SGA could achieve good quality solutions with higher population size.

To find the answers, SGA is executed with different population sizes (40, 100, 500, 1000, and 1500) for 30 independent runs for each problem. The other parameters remain constant (e.g. P_C = 0.9, P_M = 0.20). To keep the budget of fitness evaluations fixed (350,000), the maximum numbers of generations 8750, 3500, 700, 350 and 233 are used respectively.

The experimental results show that the performance of the algorithm improves with the increase of the population size, though it increases the computational time per

generation. With higher population size, the population becomes more diverse which is helpful for better performance of the algorithm.

The achieved results of SGA with population size of 1500 (the average % deviation of best results from optimum per problem is 13.42874%) are better than the results with low population size (e.g. the average % deviation of best results from optimum per problem is 75.78% with population size 100). In problems g07, g09 and g13, the best results are better with population size of 1500. Considering mean results, again the results with 1500 are better in g04, g05, g07 and g13. With higher population size (1500), SGA achieves feasible solutions in g05, however the mean results are still 2.17% from the optimum. For the higher population sizes SGA needs more computational time per generations: with population size 1500 it took on average 5.94 times more execution time than with population size 100. Though with huge population size, SGA seems to improve its performance, still it could not solve half of the equality constrained problems such as g03 and g11. The solution quality for other equality constrained problems is also not satisfactory; the achieved best result is still 146.03% from the optimum in g13.

In the experiments, considering the size of the test problems, maximum population size is considered up to 1500. The experimental results show that although the performance is improved with higher population size, SGA could not overcome its limitations for the problems with equality constraints. However the performance of the algorithms may be improved with further increase of the population size.

3.3.4 The effect of more Fitness Evaluations

So far, SGA is executed up to 350,000 fitness evaluations. It would be interesting to see whether SGA can improve the solutions if it were allowed to run for more fitness evaluations. To investigate this, the algorithm is executed up to 500,000 fitness evaluations.

For most of the problems, the results are improved. However, the improvements are not huge, and the results are still far from optimum in most of the problems. For

problems g01, g04, g06, and g09, the best results are improved by a maximum of 0.03%. Only in g07 has it improved the best results by more than 1% (2.88%), however the result is still 2.34% from the optimal.

While considering mean results, for all these problems the greatest improvement is made in g01, it is still only 0.60986%. The achieved mean results are still more than 5% from the optimum in several problems e.g. g01 (5.94%), g02 (9.13%), g07 (15.35%), and g10 (18.37%). SGA improved its performance, by achieving feasible solutions in problems g03, and g11. However, the mean results of g03 and g11 are still 98.26% and 15.46% away from the optimum respectively. In g13, the result remains the same, where the mean result is 1812.84% far from the optimum. For g05, still it could not find any feasible solutions at all.

This experimental study shows that it is unlikely to find good quality solutions even with the additional fitness evaluations.

3.4 Chapter Summary

This chapter presents a simple genetic algorithm for solving constrained optimization problems. A set of benchmark problems is used to investigate the performance of the algorithm. Although SGA is fast and achieved optimum results in two problems out of a set of 13, the results are not convincing enough for practical use of the current version of the algorithm. SGA suffers in solving problems with tiny feasible space. Especially when the test problems involve equality constraints, the performance of SGA is very poor. For the equality constrained problems it could not solve most problems, and the performance is also not satisfactory for the remaining problems.

The effect of different parameters on the performance of the algorithm is investigated. Although the results can be improved with additional computational cost, SGA is still unlikely to find good quality solutions for several problems.

The experimental results demonstrate that there is a need for improved algorithms in

solving the constrained optimization problems. Special measures should be taken while solving problems with tiny feasible space, especially for problems involving equality constraints. Improved algorithms, and different techniques to enhance the performance of the algorithm are discussed in the next few chapters.

Chapter 4

Agent-based Evolutionary Algorithms

In this chapter, a new agent-based evolutionary algorithm is proposed for solving constrained optimization problems, where the agents have the ability to independently select a suitable Life Span Learning Process (LSLP). Each agent represents a candidate solution of the optimization problem and tries to improve its solution through co-operation with other agents. Evolutionary operators consist of only crossover and one of the self-adaptively selected LSLPs. The performance of the proposed algorithm is tested on a set of benchmark problems, and the experimental results show convincing performance.

This chapter discusses the new algorithm, the different components of the algorithm, and the related issues. The experimental studies are presented in the next chapter.

4.1 Introduction

Many real world decision processes require solving optimization problems, which may not contain nice mathematical properties required by some solution techniques. Most of these problems have different types of constraints involving a set of equalities, non-equalities or both. The difficulties in solving these constrained optimization problems arise from the challenge of finding good feasible solutions. Solving this type of problems has become a challenging area in computer science and operations research due to the presence of high dimensionality, nonlinear parameter interaction, and multimodality of the objective function as well as due to the physical, geometric, and

other limitations of different constraints (Liang and Suganthan, 2006).

Evolutionary Algorithms (EAs) have brought a tremendous advancement in the area of computer science and optimization with their ability to solve many complex problems (Sarker *et al.*, 2003). Genetic algorithms, the most well known branch of EAs, have been successfully applied to many numerical and combinatorial optimization, classifier system, and engineering problems (Goldberg, 1989; Michalewicz, 1994; Sarker *et al.*, 2003). GAs are stochastic algorithms which simulate both the natural inheritance by genetics and the Darwinian strive for survival (Michalewicz and Janikow, 1996). Nevertheless, most GAs developed are unconstrained search techniques and lack an explicit mechanism to bias the search in constrained search spaces (Liang *et al.*, 2006). Furthermore traditional GAs suffer from slow convergence to locate a precise enough solution because of their failure to exploit local information (Tang *et al.*, 2007), and face difficulties solving multi-modal problems which have many local solutions within the feasible space (Takahama and Sakai, 2006). Hence it is well established that they are not well suited for fine tuning search (Krasnogor and Smith, 2005; Molina *et al.*, 2005; Muruganandam *et al.*, 2005; Zhong *et al.*, 2004) and so, to improve the performance, hybridization of algorithms has been introduced in recent times.

The improved performances are achieved by hybridizing evolutionary algorithms with Local Search (LS) techniques: so-called Memetic Algorithms (MAs). MAs have been successfully applied across a wide range of problem domains such as combinatorial optimization, optimization of non-stationary functions, and multi-objective optimization (for details see chapter 2). They converge to high quality solutions as well as search more efficiently than their conventional counterparts (Tang *et al.*, 2007). MAs are inspired by Dawkins' notion of a meme (Dawkins, 1976) defined as a unit of information that reproduces itself as people exchange ideas. One of the critical issues regarding the performance of MAs is the selection of appropriate LS while hybridizing LS with GAs. If the selection of LS is not appropriate for a particular problem then MAs may not perform well; the performance may even be worse than GAs alone (Davis, 1991; Hart, 1994; Ong and Keane, 2004). Many types of local searches are available in the literature but it is very difficult to know which type is

appropriate for a particular problem.

Agent-based computation introduces a new paradigm for conceptualizing, designing and implementing intelligent systems, and has been widely used in many branches of computer science (Ferber, 1999; Sycara, 1998). The agents are discrete individuals situated in an environment having a set of characteristics and rules to govern their behavior and interactions. They sense the environment and act on it in pursuit of a set of goals or tasks for which they are designed (Stan and Art, 1997).

To mitigate the shortcoming of MAs mentioned above, in this chapter an Agent-based Memetic Algorithm (AMA) is proposed for solving constrained optimization problems. Here an agent represents a candidate solution of the problem, carries out cooperative and competitive behaviors, and selects the appropriate local search adaptively to find optimal solutions for the problem in hand. In the proposed algorithm, the concept of MAs follows the model of adaptation in natural systems, where an individual of a population may be improved through self-learning along with the evolutionary adaptation of the population (Krasnogor and Smith, 2005; Moscato, 1989).

Recently, a number of agent-based hybrid algorithms have appeared in the literature for solving different problems (for details see chapter 2). However, to the best of our knowledge, the application of agent-based memetic algorithms to COPs is new in the literature. The real potential of AMA has not been fully explored yet, and very little has been done in this area.

In the proposed framework of agent-based memetic algorithm, for each agent, the neighborhood agents are compared with others to find the winner (like competition) with whom it exchanges genetic materials (like cooperation) through the well known simulated binary crossover proposed by Deb and Agrawal (1995), and learn through the proposed different types of life span learning processes, to solve a COP with a suitable constraint handling technique. Note that it does not use any mutation operator, as the life span learning process would cover the purpose of mutation.

The life span learning processes are designed based on several local and directed search procedures. An agent chooses a LSLP as a local search operator self-adaptively.

As we generally see in GAs, an individual in a generation produces offspring and the offspring may be mutated to change the genetic materials. In reality, beside reproduction, an individual learns and gains experiences in different ways during its life time. This process is represented by the proposed LSLPs. As MAs rely on the concept of natural evolution and learning, the proposed algorithm makes it even more meaningful. An individual in the population of a certain generation lives for a certain period of time, and explores the environment independently and interacts with other individuals in many different ways to enhance learning. As an individual may decide to have a particular learning process based on its belief, a number of different LSLPs are incorporated in AMA. The individual's ability to use its belief, to interact with the environment, and to make independent decisions for exploration and learning, qualifies an individual to be called an agent in the population.

In AMA, the agents are arranged in a lattice-like environment, as for cellular genetic algorithms (Alba and Dorronsoro, 2005; Nakashima *et al.*, 2003; Whitley, 1993) which use a lattice-based population. In cellular GAs, each individual is located in a cell of the lattice. Except for the neighborhood structure, all operations of cellular GAs and traditional GAs are identical. The problem of premature convergence in cellular GAs is also similar to the traditional GAs (Folino *et al.*, 2001). Though AMA uses a lattice-based population for the individuals, here each individual is considered as an agent, which has its own purpose and behaviors. Unlike traditional genetic operators (selection, crossover and mutation), in the proposed algorithm, the agents use SBX only with its neighboring agents through cooperation and competition, and apply LSLP.

This work has some similarities with Ong and Keane (2004). Both use Meta-Lamarckian learning, applying MA to optimization problems. Both use the concept of multi-method LSs. AMA differs from Ong and Keane (2004) in several ways, however. The adaptation mechanism in MA (Ong and Keane, 2004) is adaptive type whereas AMA is self-adaptive type. A heuristic approach (named as *Subproblem Decomposition*) used in MA with Meta-Lamarckian learning (Ong and Keane, 2004) selects a meme based on the knowledge gained from only the k nearest individuals. However, in AMA an agent selects a meme/LSLP based on the knowledge experienced by the parents. So

the adaptation level of both the algorithms is Local-level adaptation (Ong *et al.*, 2006). In the framework of MA with Meta-Lamarckian learning (Ong and Keane, 2004), it includes LSs with other genetic operators such as selection, mutation and crossover. AMA only uses crossover and life span learning with a constraint handling technique. The merit function for performance of the local search techniques proposed in AMA is a relative measure based on the objective function value for the feasible agents and constraint violations for the infeasible agents from their initial condition, details can be found in section 4.6. However the reward function used in Ong and Keane (2004) uses a relative reward parameter with the simple measurement of fitness improvement. Importantly, AMA and Ong and Keane (2004) deal with different types of optimization problems: AMA is designed for constrained real-valued optimization problems while (Ong and Keane, 2004) considers mainly real-valued function optimization with only variable bounds.

To test the performance of the algorithms a number of state-of-the-art test problems are solved and the results are compared with several existing well-known algorithms. The comparisons show that the results of AMA are quite acceptable quality. The detailed experimentation of the proposed algorithm is described in the next chapter.

The rest of this chapter is organized as follows. The next section describes the agent-based evolutionary algorithms and related issues. Section 4.3 presents the proposed AMA.The AMA operators are explained in section 4.4. The fitness evaluation and constraint handling, and selection of LSLPs are described in section 4.5 and 4.6 respectively. Finally the last section concludes the chapter.

4.2 Agent-based Evolutionary Algorithms

As mentioned in chapter 2, the agent-based evolutionary algorithms can be of two different approaches. The first type is population based evolutionary algorithms, for solving complex decision problems, where the individuals of the population are defined as agents (Siwik and Kisiel-Dorohinicki, 2006; Zhong *et al.*, 2004). The second type is

the multi-agent system that incorporates an evolutionary algorithm as a search technique to improve certain functionality of some selected agents or the entire system (Liu and Frazer, 2002). There is a long debate about the first type whether we should call it an '*agent based system*' or what does it add to EAs by naming it an '*agent based EA*'? This section discusses how an agent-based EA can be different from an independent EA.

Here the agents are defined a bit differently. The individuals in the population of EAs are not agents rather, based on the individual's belief and learning experiences, each agent stands (or supports) for one of the individuals of the population. A population of agents is endowed with a set of individualistic and social behaviors, in order to explore a local environment within the solution space. The combination of agents' local view exploration with EAs global search ability would establish a superior balance between exploitation and exploration when solving complex optimization problems. In fact, when we define the individual as an agent we can bring anything (like rules for communication, cooperation and competition, intelligence, memory, and learning) onboard which the traditional EAs do not deal with. Davidsson *et al.* (2007) indicated that we must capitalize the strength of two approaches in a new hybrid method as they complement each other. The agent activities, which can be considered with EAs in the context of optimization problem solving, are discussed below.

Environment of Agents

The environment includes the agent's social network structure, and the size of the neighborhood for interactions. The network topology usually includes ring, two-dimensional lattice, random small-world, and star type. However, we may consider any special structure such as self-organizing network. In optimization problem solving, the two-dimensional lattice-like structure is widely used. There are some similarities of this structure with cellular genetic algorithms (Nakashima *et al.*, 2003). The neighborhood size controls the amount of interaction and diversity in the entire population.

Behavior of Agents

The agents can be cooperative or competitive. Cooperative agents share information

with their neighboring agents, whereas competitive agents compete with the neighboring agents. The quality of an agent is represented by fitness value or energy (Zhong *et al.*, 2004). An agent with a higher fitness value has a better chance of survival in its neighborhood. De Jong (2008) stated that the agent behavior is basically a combination of "nature and nurture", those are both inherited and learned components.

Learning of Agents

De Jong (2008) indicated that evolution operates at the population level while "lifetime learning" occurs at the individual (agent) level. The agents learn throughout their life span which improves their quality (fitness value). This learning process can be chosen by the individual agents independently. For example, in optimization problem solving, the local search techniques could be the learning processes for an agent. Vasile and Locatelli (2008) indicated that each agent performs a sequence of actions at every generation according to their defined behavior such as inertia, follow-the-tail, random-step, linear blending and quadratic blending. These basically represent local search techniques, which is labeled as learning processes in this research.

Reasoning Capability of Agents

It is well known that to move in the direction of producing the fastest rate of improvement in the fitness value is not always best (Thornton and Boulay, 1999). So an agent must find reasoning for its next move. An agent may apply either quantitative or qualitative judgment (or both) based on its own belief, social interaction, knowledge and intelligence. In optimization problem solving, the choice of self-learning process may be based on either simple rules or chosen adaptively (like adaptation to environment changes). The adaptation process also requires some rules or procedures to follow. To make a rational choice of self-learning process, the knowledge must be retrieved, reused, revised and retained. That means it requires systematic archiving of relevant information. Bajo and Corchado (2006) defined the knowledge revising process as the reasoning cycle.

The reasoning capability of agents will make a clear difference between the agent-

based EAs and EAs alone. It is known that the incorporation of MAS with EAs would increase the computational time per generation of the new algorithm. However it is expected that a carefully designed hybrid algorithm (MAS plus EAs) would not only improve the quality of solution but also reduce the overall computation time, as is the case for memetic algorithms (Hasan *et al.*, 2008).

4.3 Agent-based Memetic Algorithm (AMA)

Memetic algorithms can be considered as a marriage between the population-based global search and the heuristic-based local search (Krasnogor and Smith, 2005; Moscato, 1989). The global search explores the search space while the local search exploits the obtained solution of an individual. This approach may reflect the natural adaptation and learning through (Krasnogor, 2002; Krasnogor and Smith, 2005; Moscato, 1989):

- Evolutionary adaptation of the population and
- Individual localized learning.

Besides the evolutionary adaptation and learning, certain individuals may develop themselves through self-learning and exploiting their own potential. Depending on the environment, available resources, opportunities, and self-potential, the individuals enhance their performance through different types of learning. This additional learning step, which mimics both the natural and artificial knowledge building process, will add more useful information than localized learning alone.

In this research, the agent concept is incorporated with memetic algorithms, where an agent stands for a candidate solution in the population. The characteristics of an agent can be defined as follows (Liu *et al.*, 2002b):

- Ability to live and act in the environment.
- Ability to sense the local environment.
- Purpose driven.

- Reactive behavior.

In the proposed algorithm, the goal of each agent is to improve its fitness while satisfying constraints. Following the natural adaptation process, in the proposed AMA the agents improve their fitness by selecting intelligently a suitable self learning technique, together with the evolutionary adaptation of the population. As shown in Figure 4.1, if the goal is achieved the process stops, otherwise the modified agents go through the same process. The agents are arranged in a lattice-like environment E of size $M \times M$ (where M is always an integer). The agents communicate with their surrounding neighbor agents and exchange information with them through comparison and the crossover operator.

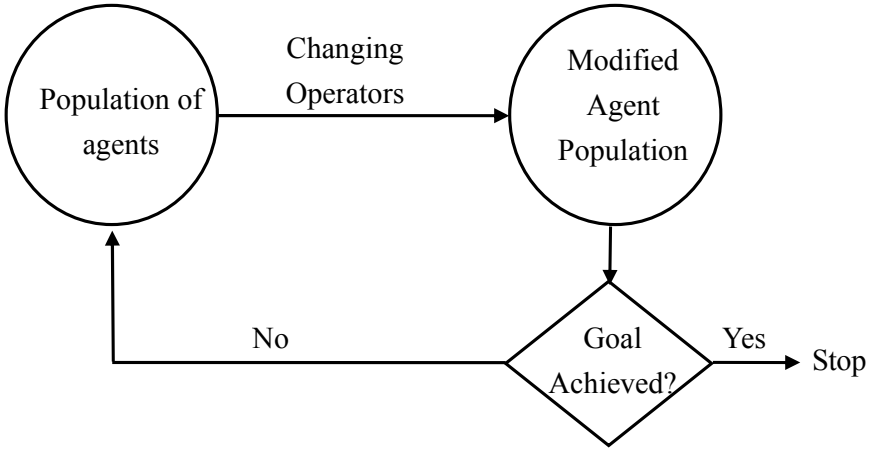


Figure 4.1: Agent-based Memetic Algorithm.

The overlapped small neighborhoods of the lattice-like environment help in exploring the search space because the induced slow diffusion of solutions through the population (by the competition and co-operations of the agents inside each neighborhood) provides a kind of exploration (diversification), while exploitation takes place by the individual agents through their learning processes. For a larger size of neighborhood, the overlapping of the neighborhoods in comparisons and competitions is higher. In this case, the dominant individuals can spread their genetic material throughout the population faster than a small neighbourhood, which may result in

premature convergence in the population. Different sizes of neighborhood are considered in this research and the effect of neighborhood sizes is discussed in the next chapter.

For LSLPs, a number of search processes are proposed as an appropriate choice of a LSLP is very important for the performance of the algorithm. In each generation an agent may select one of the several LSLPs based on several simple rules.

AMA involves Meta-Lamarckian learning (Ong and Keane, 2004), rather than Lamarckian or Baldwinian learning. In the Lamarckian mechanism, the genotypes are modified by learning in order to improve the fitness. The improvement is therefore passed to these chromosomes. The idea is that the learnt behavior can directly change genotypes. Therefore, the acquired knowledge through learning is directly coded into the genotype, and knowledge can be transferred to the offspring (Houck *et al.*, 1996; Ishibuchi *et al.*, 2005; Whitley *et al.*, 1994). Baldwinian learning uses improvement procedures to change the fitness landscape, but the solution that is found is not encoded back into the genotype (Guimaraes *et al.*, 2006; Houck *et al.*, 1996; Ishibuchi *et al.*, 2005; Spalanzani, 2000; Whitley *et al.*, 1994). In AMA the improved agent (after applying LSLP) is sent back into the population which follows the Lamarckian learning. Since multiple LSLPs (i.e. LSs) are used during a MA search in the spirit of Lamarckian learning, it can be said AMA is following Meta-Lamarckian learning.

The main steps of the proposed algorithm are as follows:

Step 1. Create a random population, which consists of $M \times M$ agents.

Step 2. Arrange the agents in a lattice-like environment.

Step 3. Evaluate the agents individually.

- If the stopping criterion has been met, go to step 7; otherwise continue.

Step 4. For each agent examine its neighborhood.

- Select an agent from its neighborhood and perform crossover.

Step 5. Select a certain percentage of agents.

- Select self-adaptively a life span learning process.

Step 6. Go to step 3.

Step 7. Stop.

To solve COPs along with crossover operator and different type of LSLPs, a suitable constraint handling technique is added to handle the constraints in AMA: they are dealt with indirectly in fitness evaluation. As the search spaces of the optimization problems and the variables under consideration are real and continuous, real numbers are used to represent the solutions.

The next sections provide details of how these steps are done: crossover in section 4.4.1, LSLPs in section 4.4.2, fitness evaluation and constraint handling in section 4.5 and selection of LSLPs in section 4.6.

4.4 AMA Operators

To search for optimum solutions efficiently, the search techniques of the agents need to ensure two essential goals: exploration and exploitation. Exploration ensures that all parts of the search space are investigated; exploitation concentrates searching around the solutions found so far (Ong and Keane, 2004; Torn and Zilinskas, 1989). In the proposed algorithm, the usual selection, ranking, and mutation processes of the

traditional MAs/GAs (Guimaraes *et al.*, 2006) are not directly used. AMA only applies simulated binary crossover operators (Deb and Agrawal, 1995) and the LSLPs which ensures both the goals: exploration and exploitation.

As every agent interacts with its neighborhood to exchange information, the information is diffused to the whole agent lattice. After this exploration, a certain percentage of the agents are selected for different types of LSLPs for exploitation of the currently obtained solution. During the learning process, the agent tries to improve its present fitness by changing the solution vector. It continues to learn for a certain number of steps. But if the fitness decreases then it stops the learning process. Details of these operators are discussed below.

4.4.1 Crossover

In the proposed AMA, simulated binary crossover (Deb and Agrawal, 1995) is used. SBX operator performs well in solving problems having multiple optimal solutions with a narrow global basin, and has been used in different applications successfully (Deb, 2000; Deb, 2001; Deb and Beyer, 2001). Details of SBX are discussed in chapter 2.

When this crossover operator is applied on the solution of an agent $A_{i,j}$ (located in location (i,j) in the lattice), the agent searches for its best neighbor agent to mate. The better offspring from these two, denoted as $N_{i,j}$, is stored in a pool. After completing the crossover in a given generation, the fitness of each $N_{i,j}$ $(1 \leq i,j \leq M)$ is compared with its parent $A_{i,j}$. If the new agent's solution's fitness is better than its parent then it takes the place of $A_{i,j}$ and $A_{i,j}$ dies. Otherwise $A_{i,j}$ would survive.

4.4.2 Life Span Learning Processes

After crossover a certain percentage of agents (with P_L probability) of the population are selected to learn with the designed LSLPs i.e. to apply local and directed search. These LSLPs are designed to find better fitness values by changing the variable

vector of the existing solutions in different ways.

Here four different LSLPs are designed. The first is totally random in nature, the second is restricted random, the third is gradient-based, and the last is directed search. The random LSLPs ensure diversity, and the directed searches try to move towards a better solution which is not necessarily in the individual's locality. The pseudo codes of the LSLPs are given below:

Let an agent $A_{i,j}$, residing at cell location (i,j) with solution vector of n variables $[a_1, a_2, \ldots, a_n]$, be selected for LSLP. Let m be the maximum number of learning steps, and Δ is a positive small value for perturbation. The procedure for calculating Δ is discussed later in this section.

LSLP Type 1

Step 1. Choose a variable r randomly from n variables;

Step 2. Calculate Δ, add / subtract Δ with the variable value a_r and evaluate fitness, and detect in which direction the fitness of the solution vector improves;

Step 3. For t= 1 to m do

Step 3.1. Change the variable value a_r with Δ according to the direction found in step 2;

Step 3.2. If the fitness deteriorates go to step 1 else go to step 3;

[End of Step 3 Loop]

Step 4. Continue the previous steps (1 to 3) until all n variables are modified and then go to step 5;

Step 5. Stop.

LSLP Type 2

Step 1. For all variables r =1 to n do

Step 2. Calculate Δ, add / subtract Δ with the variable a_r and evaluate fitness, and detect in which direction the fitness of the solution vector improves;

Step 3. For t = 1 to m do

Step 3.1. Change the variable a_r with Δ according to the direction found in step 2;

Step 3.2. If the fitness deteriorates, go to step 1 else go to step 3;

[End of Step 3 Loop]

[End of Step 1 Loop]

Step 4. Stop.

LSLP Type 3

Step 1. For all variables r = 1 to n do

Step 2. Calculate Δ, add/subtract Δ with the variable a_r and evaluate fitness, and detect in which direction the fitness of the solution vector improves;

Step 3. Change the variable a_r according to the direction found in step 2. Find the improvement of the fitness for this change;

[End of Step 1 Loop]

Step 4. Rank the variables based on their effect on the fitness improvement;

Step 5. For all n variables starting from highest rank do

Step 5.1. For t = 1 to m do

Step 5.2. Change a_r as $a_r = a_r \pm \Delta$; based on the direction found in step 2;

Step 5.3. If the fitness deteriorates go to step 5, otherwise go to step 5.1;

[End of Step 5.1 Loop]

[End of Step 5 Loop]

Step 6. Stop.

LSLP Type 4

Step 1. Find the agent with best fitness in the current generation with solution vector $[b_1, b_2, ..., b_n]$;

Step 2. For all variables $r = 1$ to n do

Step 2.1. For $t = 1$ to m do

Step 2.2. Calculate Δ;

Step 2.3. If $(a_r > b_r)$ then $a_r = a_r - \Delta$;

Step 2.4. If $(a_r < b_r)$ then $a_r = a_r + \Delta$;

[End of Step 2.1 Loop]

[End of Step 2 Loop]

Step 3. Stop.

The first and second LSLPs are random in nature. Suppose an agent with the solution vector of three variables [1.4, 2.0, 3.0] is using LSLP type 1. It will generate an index number randomly in the range (1–3). Let the random index be 2, and Δ=0.1. a_2 will be changed to $a_2 = a_2 \pm \Delta$; if the fitness increases by adding Δ (i.e. now a_2=2.1), the agent will try to go in this direction by adding Δ for m times. While learning, if the fitness decreases or the agent has passed m steps, this learning period stops and the cycle starts again. The agent will generate another index number e.g. 3. Now a_3 will be selected to search for better fitness for maximum m times as before. This process will continue until all the variables have been selected at least once.

The second LSLP type is same as the first, except for the sequence of variable selection. Instead of selecting the variables randomly, the variables are selected in ascending order of the index i.e. a_1 will be selected first then a_2 and so on.

The third LSLP is like the gradient-based search with the central difference method. Initially every variable is changed by Δ. Then based on the effect of changing each variable on the fitness, the variables are ranked. Variables are selected to be modified according to the rank. Suppose by changing a_1 in the positive direction, the fitness of the solution of the agent is improved by 1%, for a_2 the improvement is 20%, and for a_3

the improvement is 30%. Then the sequence of selection of the variables will be a_3, a_2, and then a_1.

The last type of LSLP is similar to directed search. Here, the selected agent $A_{i,j}$ with solution vector [a_1, a_2, …, a_n] tries to reach nearer to the best agent of the previous generation with solution vector [b_1, b_2, …, b_n]. It attempts to change each variable a_1, a_2, …, a_n by Δ to get closer to b_1, b_2,…, b_n in up to m learning steps.

The first and second types of LSLP of the agent try to exploit the existing solution vector by attempting to change the variables. The third type attempts to move the solution vector in the direction of gradient and the last type of LSLP leads the agents towards the current best solution. The last two LSLPs try to make the agents converge; the first two maintain diversity.

During the LSLP the variables are changed with $\pm \Delta$. The direction of Δ (add/ deduct) is selected by observing the modified fitness value of the agent. The value of Δ should be very small and gradually be decreasing with the generation numbers, to obtain a high quality solution (high precision) at the end. Here the value of Δ is considered as $\Delta = |G(0,1/g)|$ (for the first three types of LSLPs), where $G(0,1/g)$ is a Gaussian random number generator with zero mean and standard deviation $(1/g)$, here g is the present generation number. In the fourth LSLP $\Delta = (b_r - a_r)/2$, where b_r is the r^{th} solution variable of the previous best agent, a_r is the r^{th} variable of the present agent and a_r is updated in each step, which speed up the directed search.

4.5 Fitness Evaluation and Constraint Handling

Evolutionary algorithms are well-known for their success in solving unconstrained optimization problems. For solving constrained problems, an additional mechanism must be incorporated into the fitness function or the evolution strategies to guide the search direction (Liang and Suganthan, 2006). During the last decade several methods were proposed for handling constraints for real valued optimization problems (Coello Coello, 2002; Michalewicz and Schoenauer, 1996). A brief discussion of constraint

handling techniques can be found in section 2.4.3.

In AMA, the goal of the individual agent is to minimize the objective function value while satisfying the constraints. To improve the fitness, agents first apply crossover operators with their best neighbors. The best neighbor is found by using pair-wise comparison among the neighbors. The pair-wise comparison indirectly handles the constraints. Like Deb (2000) while comparing the fitness of two individual agents it considers:

- A feasible individual is always better than an infeasible individual.
- If both of the individuals are feasible, then the individual with lower objective function value is better (considering minimization problem).
- If both of them are infeasible, then the one with less constraint violation is better. The total Constraint Violation (CV) of an individual is considered here as the sum of absolute values by which the constraints are violated.

It is assumed here that the fitness of the best infeasible agent is worse than the worst feasible agents. As such while comparing two agents, the infeasible agent is penalized and feasible agent is rewarded, so the constraints are handled indirectly.

The equality constraints have been converted into inequality constraints $-\delta \leq h_j(\boldsymbol{X}) \leq \delta$, where δ is a small tolerance value. Actually the presence of equality constraints (with $\delta = 0$) makes the feasible region very small compared to the search space, which makes it harder for an evolutionary algorithm to find feasible solutions. A large value for δ allows the algorithm to find some feasible solutions easily by temporarily increasing the feasible space. If the search space is reduced after some generations by decreasing δ, the algorithm tries to improve the previous feasible solutions to fit into the re-defined feasible space. This dynamically changing value for δ is considered in AMA while dealing with equality constraints. Details of the δ calculation will be discussed in next chapter.

4.6 Selection of LSLPs

A certain percentage of the agents are allowed to apply different type of LSLPs to exploit their current position for improving fitness. To select an appropriate LSLP, the agents check a number of simple rules. Initially all the agents are assigned different types of LSLPs randomly with Improvement Index (*II*) zero. Here, *II* indicates the rate of fitness improvement made by a particular LSLP type assigned to an agent. A positive value of *II* indicates that fitness is improved by the LSLP, while a negative value indicates deterioration of fitness. When selecting a LSLP, an agent checks the parents' type of LSLPs and *II* values. The parents may have different type of LSLPs associated with *II* values. The offspring will choose the LSLP which has the higher *II* value. Since an agent selects a LSLP based on the knowledge experienced by the parents the adaptation level of the algorithms is Local-level adaptation (Ong *et al.*, 2006).

The feasible and infeasible agents are separated for the calculation of *II* as a feasible agent is given more preference than an infeasible one (this indirectly handles the constraints). While calculating the *II*, if an infeasible solution vector of an agent becomes feasible after applying a LSLP, the LSLP is rewarded by assigning the maximum *II* value (+1). On the other hand a LSLP is penalized which converts a feasible agent to an infeasible one by assigning worst *II* value (−1). As discussed earlier, we should consider the objective function values for feasible agents that remain feasible after applying the LSLP, and constraint violations for infeasible agents that remain infeasible after the LSLP. Considering minimization problems, if the solution remains feasible after the LSLP, the improvement index for feasible agent is then based on the fitness values as follows:

$$II = \frac{(Obj.\ func.\ value\ \ before\ \ LSLP) - (Obj.\ func.\ value\ \ after\ \ LSLP)}{(Obj.\ func.\ value\ \ before\ \ LSLP)} \tag{4.1}$$

Note that *II* is positive if the objective function value is smaller after LSLP, as desired for a minimization problem.

If an agent with an infeasible solution uses a LSLP which still results in an infeasible solution, the improvement index for the infeasible agent is based on total

constraint violations (CV) as follows:

$$II = \frac{(Total\ CV\ before\ LSLP) - (Total\ CV\ after\ LSLP)}{(Total\ CV\ before\ LSLP)} \tag{4.2}$$

The value of II is restricted in the range $-1 \leq II \leq +1$ by assigning values outside this range to the boundary values.

4.7 Chapter Summary

This chapter has introduced a new agent-based memetic algorithm for solving constrained real-valued optimization problems, by tailoring multi-agent concepts into a new memetic algorithm. The individual candidate solutions of problems are represented as agents with additional characteristics. Solving constrained real-valued optimization by using agent-based memetic algorithm is new in the literature.

The performance of the algorithm is investigated in solving a set of test problems. The details of experimental results, the comparisons with other algorithms, and the effect of different components shall be discussed in the next chapter.

Chapter 5

Experimental Studies of AMA

The design of a new AMA is discussed in the previous chapter. This chapter reports the experimental studies of the algorithm. It provides the detailed results of a set of benchmark problems, comparison of the results with other well-known algorithms, and the effect on the performance of different components of the algorithm. The results show that the proposed algorithm is efficient in solving COPs.

5.1 Introduction

In the previous chapter, a framework of AMA for solving constrained optimization problems has been proposed. In this chapter, the performance of the algorithm is investigated by solving a set of 18 test problems which includes five new problems plus 13 existing well-known problems. The results are compared with four GA based well-known algorithms (such as Koziel and Michalewicz, 1999; Chootinan and Chen, 2006; Farmani and Wright, 2003; and Elfeky *et al.*, 2006), one multi-populated differential evolution algorithm (Tasgetiren and Suganthan, 2006) and one Evolutionary Strategy (ES) based algorithm (Runarsson and Yao, 2000). In addition to the comparison of the best fitness values, statistical significance testing is also carried out. The comparisons show that the proposed approach gives mostly improved or comparable results to other algorithms.

A number of experiments are designed and carried out to see the effect of different parameters such as probability of LSLP, neighborhood size, crossover, and population

size. These experimental results are analyzed, and their findings are discussed.

The organization of this chapter is as follows: the next section discusses the test problems and their solutions using the proposed AMA. Then in section 5.3, the solutions of AMA are compared with other algorithms. The contributions of different components of AMA are analyzed in section 5.4. Finally the last section concludes the chapter.

5.2 Test Problems and Experimental Results

A set of 18 benchmark problems is used to test the performance of the proposed AMA. The first 13 problems (indicated as g01-g13) are well known in the literature, initially studied by Michalewicz and Schoenauer (1996), Koziel and Michalewicz (1999), and Michalewicz (1995), and further studied by Runarsson and Yao (2000), Chootinan and Chen (2006) and others. The other 5 problems (indicated here as B01-B05) are new and collected from the literature (Floudas, 1999; Himmelblau, 1972). The benchmark problems include different forms of objective function (linear, quadratic, cubic, polynomial, nonlinear) and different number of variables (n). The characteristics of the test problems are given in Table 5.1 and the detailed mathematical representations are provided in the Appendix A and B. The first 13 problems are also presented in Table 3.1, and repeated in Table 5.1 for convenience.

The equality constraints $h_j(\boldsymbol{X})=0$ of g03, g05, g11, g13, B01, and B02 have been converted into inequality constraints $-\delta \le h_j(\boldsymbol{X}) \le \delta$, where δ is a small tolerance value. Initially if we assign a very small value for δ, the solution space will be too small for the algorithm to find feasible solutions. A large value for δ allows the algorithm to find some feasible solutions easily by increasing the solution space. If the search space is reduced after some generations by decreasing δ, the algorithm tries to improve the previous feasible solutions to fit into the new solution space. This dynamic value for δ is considered in this research while dealing with equality constraints. Initially δ is assigned 1. After every 16% of the maximum generation number, δ is divided by 10. Finally after 80% of the generations δ is left fixed at 0. The value of δ is reached to 0 after five steps, which allows the algorithms to improve the solutions gradually.

Table 5.1: Characteristics of the test problems.

Fn	(*n*)	Obj. Fuc.	ρ	LI	NI	LE	NE	AC	Optimal
g01	13	Quadratic	0.0111%	9	0	0	0	6	-15.000
g02	20	Nonlinear	99.8474%	0	2	0	0	1	-0.803619
g03	10	Polynomial	0.0000%	0	0	0	1	1	-1.000
g04	5	Quadratic	52.1230%	0	6	0	0	2	-30665.539
g05	4	Cubic	0.0000%	2	0	0	3	3	5126.498
g06	2	Cubic	0.0066%	0	2	0	0	2	-6961.814
g07	10	Quadratic	0.0003%	3	5	0	0	6	24.306
g08	2	Nonlinear	0.8560%	0	2	0	0	0	-0.095825
g09	7	Polynomial	0.5121%	0	4	0	0	2	680.630
g10	8	Linear	0.0010%	3	3	0	0	6	7049.331
g11	2	Quadratic	0.0000%	0	0	0	1	1	0.750
g12	3	Quadratic	4.7697%	0	9^3	0	0	0	-1.000
g13	5	Nonlinear	0.0000%	0	0	0	3	3	0.053950
B01	10	Nonlinear	0.0000%	0	0	3	0	3	-47.765
B02	3	Quadratic	0.0000%	0	0	1	1	2	961.715
B03	5	Nonlinear	0.0204%	4	34	0	0	4	-1.905
B04	9	Quadratic	0.0000%	0	13	0	0	6	-0.866025
B05	2	Linear	79.6556%	0	2	0	0	2	-5.508

ρ = Ratio between the feasible space and the search space, LI = Linear Inequalities, NI = Nonlinear Inequalities, LE = Linear Equalities, NE = Nonlinear Equalities, AC = Active Constraints.

From Table 5.1, it is very clear that the benchmark problems are different in number of variables, type of objective functions, and type of constraints. For different types of problems, one would expect to use different parameters for population size, and probability of learning (P_L). Initially the algorithm is run for each of the problems varying the population size ($M \times M$) from $M = 9$ to 25 and P_L from 0.05 to 0.25 with an increment of 0.05. After this experimentation, the population size and P_L are selected based on the performance of the algorithm. The population size and the values of P_L used are given in the last column of Table 5.2. During the LSLP the agent is allowed at

most m = 10 steps. The effect of learning steps have been investigated: lower values of m slow down the convergence. On the other hand with larger values of m the performance does not improve significantly with increase of the additional computational cost. The maximum number of generations was set in this experiment at 3500. The initial solution vectors for the agents are randomly generated within the bounds of each decision variable.

The best, mean, standard deviation, median, and worst of 30 independent runs with 30 different random seeds are given in Table 5.2. Following the same format used in the literature (Elfeky *et al.*, 2006; Koziel and Michalewicz, 1999; Runarsson and Yao, 2000) the results are rounded to 6 decimal places in g02, g08, g13, B04 and for the remaining problems the results are rounded to 3 decimal points. The maximization problems here are transformed into equivalent minimization problems.

All the experiments in this research have been carried using Visual C++ at computers with Microsoft Windows XP operating systems (CPU 1.66 GHz, 1 GB RAM).

Table 5.2 shows that among the first 13 problems (g01-g13) the proposed algorithm has achieved the optimum for nine (g01, g02, g03, g04, g06, g08, g11, g12, and g13), and for the new five problems (B01-B05) it has achieved the optimum for four (B02, B03, B04, and B05). Though for the remaining problems AMA could not achieve optimum, the achieved best results are very close to the optimum results. In g05, g09 the achieved best results are within 0.0005% of the optimum (0.00027%, 0.00015%). For g07 and B01 the best results are within 1% of the optimum (0.07817%, 0.02512%). Only in g10 is the achieved best result more than 1% from the optimum (3.278%). In spite of being a population based stochastic search algorithm, the mean results achieved by AMA for the 30 runs are also very impressive. For 9 problems (g01, g03, g04, g06, g08, g11, g12, B03, and B05) the achieved mean results are exactly the same as optimum. For 7 problems the achieved mean are within 0.5% of the optimum results. In the other two problems the mean results are close to the optimum, e.g. within 6.09% in g10 and within 2.066% in B01. The other results e.g. median, worst, standard deviations achieved by AMA are of very good quality. The results show AMA can be applied

successfully in solving different types of COPs.

Table 5.2: Statistics for 30 independent runs of the proposed AMA.

Fn	Optimal	Best	Mean	St.Dev.	Median	Worst	PS/P_L
g01	-15.000	-15.000	-15.000	0.00E+00	-15.000	-15.000	400/0.2
g02	-0.803619	-0.803619	-0.803500	2.20E-05	-0.803488	-0.803465	400/0.2
g03	-1.000	-1.000	-1.000	6.59E-06*	-1.000	-1.000	400/0.2
g04	-30665.539	-30665.539	-30665.539	1.46E-04	-30665.539	-30665.538	289/0.15
g05	5126.498	5126.512	5148.966	6.41E+01	5134.349	5482.953	400/0.2
g06	-6961.814	-6961.814	-6961.814	2.88E-08*	-6961.814	-6961.814	169/0.15
g07	24.306	24.325	24.392	5.18E-02	24.378	24.491	256/0.15
g08	-0.095825	-0.095825	-0.095825	0.00E+00	-0.095825	-0.095825	400/0.2
g09	680.630	680.631	680.721	5.26E-02	680.726	680.802	324/0.05
g10	7049.331	7280.436	7479.064	9.84E+01	7498.673	7598.573	400/0.2
g11	0.750	0.750	0.750	2.99E-08*	0.750	0.750	400/0.2
g12	-1.000	-1.000	-1.000	0.00E+00	-1.000	-1.000	100/0.2
g13	0.053950	0.053950	0.054020	4.84E-05	0.054130	0.054340	400/0.2
B01	-47.765	-47.752	-46.777	5.35E-01	-46.633	-45.676	400/0.2
B02	961.715	961.715	961.722	9.15E-03	961.717	961.75	400/0.2
B03	-1.905	-1.905	-1.905	7.20E-04	-1.905	-1.901	400/0.2
B04	-0.866025	-0.866025	-0.866014	9.87E-06	-0.866016	-0.865994	400/0.2
B05	-5.508	-5.508	-5.508	9.03E-16*	-5.508	-5.508	400/0.2

*Though the best, worst, median, and mean results are the same, standard deviation is positive due to rounding error.

5.3 Comparison with Other Algorithms

To compare the performance of AMA with others, several well-known algorithms are considered. The first 13 problems (g01-g13) are widely used in the literature. For these 13 problems, AMA is compared with five well-known algorithms namely Koziel and Michalewicz's GA (Koziel and Michalewicz, 1999), Runarsson and Yao's ES-based algorithm (Runarsson and Yao, 2000), Chootinan and Chen's GA with gradient-based

repair method (Chootinan and Chen, 2006), Farmani and Wright's self-adaptive fitness formulation based GA (Farmani and Wright, 2003), Elfeky *et al.*'s GA (Elfeky *et al.*, 2006). However these algorithms have not attempted the five new problems. Tasgetiren and Suganthan's multi-populated differential evolution algorithm (Tasgetiren and Suganthan, 2006) has solved these problems.

Koziel and Michalewicz's GA (abbreviated as KM) depends on a homomorphous mapping between an n dimensional cube and the feasible part of the search space. Its drawback is that requires an initial feasible solution.

Runarsson and Yao (2000) used an interesting ranking procedure known as stochastic ranking in their ES-based algorithm and solved the first 13 problems. This algorithm is well-known for its very good performance on these 13 problems. This algorithm is abbreviated in this thesis as RY.

Chootinan and Chen (2006) (abbreviated as CC) used a repair procedure embedded into a simple GA as a special operator. They solved only the first 11 problems.

Farmani and Wright (2003) (indicated here as FW), proposed a self-adaptive fitness formulation for constrained optimization and solved only the first 11 problems. They designed a two stage dynamic penalty method which applies a small penalty for slightly infeasible solutions with reasonable fitness values. In this way, it permits those infeasible individuals to survive and be promoted to a feasible region near the optimal solution.

Elfeky *et al.* (2006) used GAs with a new ranking, selection, and triangular crossover methods. The algorithm is abbreviated as TC. The idea behind this new method is the exploitation of some features of constrained problems. Here they have calculated the constraint violation without penalizing the individuals, and have used this information to rank and select the individuals. TC has solved only the 9 problems out of first 13 that have only inequality constraints.

While comparing with different types of algorithms the population size is used 400 and maximum number of generation is considered 875 to ensure the same number of fitness evaluations (350,000) used by the other algorithms (Chootinan and Chen, 2006;

Elfeky *et al.*, 2006; Farmani and Wright, 2003; Koziel and Michalewicz, 1999; Runarsson and Yao, 2000) for the first 13 problems. Crossover is applied to all agents and the probability of LSLP (P_L) is 0.2 in all tests.

Table 5.3 gives the results (best, mean, standard deviation) of RY, KM, CC, FW, TC and proposed AMA for the 13 test problems for 30 independent runs. The cells are left empty when the algorithms have not reported that particular problem.

Table 5.3: Comparison of results with different algorithms for 13 problems (g01-g13).

Fn	Optimal		KM	RY	FW	CC	TC	AMA
		Best	-14.786	-15.000	-15.000	-15.000	-15.000	-15.000
g01	-15.000	Mean	-14.708	-15.000	-15.000	-15.000	-15.000	-15.000
		St.Dev	-	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
		Best	-0.799530	-0.803515	-0.802970	-0.801119	-0.803516	-0.803619
g02	-0.803619	Mean	-0.79671	-0.781975	-0.79010	-0.785746	-0.791345	-0.803500
		St.Dev		2.00E-02	1.20E-02	1.37E-02	9.42E-03	2.20E-05
		Best	-1.000	-1.000	-1.000	-1.000	-	-1.000
g03	-1.000	Mean	-1.000	-1.000	-1.000	-0.999	-	-1.000
		St.Dev	-	1.90E-04	7.50E-05	5.99E-05	-	6.59E-06*
		Best	-30664.500	-30665.539	-30665.500	-30665.539	-30665.531	-30665.538
g04	-30665.539	Mean	-30655.3	-30665.539	-30665.200	-30665.539	-30665.531	-30665.537
		St.Dev	-	2.00E-05	4.85E-01	0.00E+00	9.16E-03	4.27E-04
		Best	-	5126.497	5126.9890	5126.4981	-	5126.512
g05	5126.498	Mean	-	5128.881	5432.08	5126.4981	-	5148.966
		St.Dev	-	3.5E+00	3.89E+03	0.00E+00	-	6.41E+01
		Best	-6952.100	-6961.814	-6961.800	-6961.814	-6961.814	-6961.807
g06	-6961.814	Mean	-6342.6	-6875.940	-6961.800	-6961.814	-6961.814	-6961.804
		St.Dev	-	1.60E+02	0.00E+00	0.00E+00	3.70E-12	2.25E-03
		Best	24.620	24.307	24.480	24.329	24.307	24.315
g07	24.306	Mean	24.826	24.374	26.580	24.472	25.057	24.315
		St.Dev	-	6.60E-02	1.14E+00	1.29E-01	2.38E-01	1.08E-14*

Fn	Optimal		KM	RY	FW	CC	TC	AMA
		Best	-0.095825	-0.095825	-0.095825	-0.095285	-0.095825	-0.095825
g08	-0.095825	Mean	-0.089157	-0.095825	-0.095825	-0.095285	-0.095825	-0.095825
		St.Dev	-	2.60E-17	0.00E+00	2.70E-09	4.23E-17	0.00E+00
		Best	680.910	680.630	680.640	680.630	680.630	680.645
g09	680.630	Mean	681.16	680.656	680.720	680.638	680.659	680.671
		St.Dev	-	3.40E-02	5.92E-02	6.61E-03	1.98E-02	9.18E-03
		Best	7147.900	7054.316	7061.340	7049.2607	7054.316	7280.436
g10	7049.331	Mean	8163.6	7559.192	7627.890	7049.5659	7493.719	7479.064
		St.Dev		5.30E+02	3.73E+02	5.70E-01	3.87E+02	9.84E+01
		Best	0.750	0.750	0.750	0.750	-	0.750
g11	0.750	Mean	0.750	0.750	0.750	0.750	-	0.750
		St.Dev	-	8.00E-05	0.00E+00	3.21E-08	-	2.99E-08*
		Best	-0.999	-1.000	-	-	-1.000	-1.000
g12	-1.000	Mean	-0.999	-1.000	-	-	-1.000	-1.000
		St.Dev	-	0.00E+00	-	-	0.00E+00	0.00E+00
		Best	0.054000	0.053957	-	-	-	0.053950
g13	0.053950	Mean	0.064000	0.067543	-	-	-	0.054020
		St.Dev	-	3.10E-02	-	-	-	4.84E-05

KM = Koziel and Michalewicz (1999), RY = Runarsson and Yao (2000), FW = Farmani and Wright (2003), CC = Chootinan and Chen (2006), TC = Elfeky *et al.* (2006), AMA= proposed Algorithm, * Standard deviation is positive due to rounding error.

KM has solved 12 problems except g05 (the algorithms did not provide quality results for this problem); AMA performs better in 10 of these problems and the same as KM for the other 2 problems where both algorithms achieve the optimum.

In 5 problems (g01, g03, g08, g11, g12) the best and mean results of AMA are exactly the same as the optimum results. RY has also achieved the same results for those problems. FW and CC have achieved optimum for the first four problems (g01, g03, g08, g11), however they have not tried g12 and g13. TC also achieved the optimal for g01, g08, and g12, and they have not considered the other problems involving equality

constraints (g03, g05, g11, and g13).

For g07, AMA could not achieve the optimum but the mean of AMA is the best among these algorithms. The deviation of mean from optimum for AMA is only 0.03703%, when KM, FW, TC could not reach within 2% (2.13939%, 9.35571%, 3.08977% respectively).

The best result with AMA in g06 is 0.00010% away from the optimum result, which is better than the achievements of KM and FW. In this problem the achieved mean of AMA is 0.00014% from the optimum whereas KM, RY and FW are 8.89443%, 1.23350%, 0.00020% from optimum respectively.

AMA could not achieve the optimum in g04, however the achieved best result is only 0.000003% away from the optimum. The achieved mean result of AMA is 0.000007% from the optimum, which is better than KM (0.033389%), FW (0.001105%), and TC (0.000026%). For g10 the mean result of AMA is also better than KM, RY, FW, and TC.

For g02 and g13 the performance of AMA is superior to the other algorithms. In g02 AMA has achieved the optimum and the deviation of achieved mean result of AMA from optimum is 0.01481%. RY, FW, CC, and TC could not achieve mean results even within 1% of the optimum in this problem (e.g. 2.69332%, 1.68226%, 2.22406%, and 1.52734% respectively). Among the other algorithms only KM's mean is within 0.85974% of optimum, which is far way from AMA's result.

AMA also achieved optimum in g13. FW, CC, and TC have not attempted g13. In this problem the achieved mean result of AMA is 0.12975% from the optimum, which is much better than the other two algorithms (KM, RY) as the achieved mean results of KM and RY are 18.62836% and 25.19555% away from the optimum.

The five new test problems (B01-B05) were not solved by KM, RY, FW, CC, and TC. The results of AMA are compared for B01-B05 with that of multi-populated differential evolution algorithm (denoted here as TS) proposed by Tasgetiren and Suganthan (2006). TS employed the notion of the *near feasibility threshold* to penalize the infeasible solutions. However TS used a higher number of fitness evaluations

(500,000). So, for comparing AMA results, the same number of fitness evaluations is used. The best, mean and the standard deviations of these algorithms are given in Table 5.4. For these five new problems (B01-B05) from Table 5.4 we can see AMA has achieved optimal results for 4 problems (B02, B03, B04, B05), the best results for B01 is 0.025% from the optimal. The mean and standard deviations of the 30 runs results are also competitive with TS.

Table 5.4: Comparison of results for problems B01-B05

Fn	Optimal		TS	AMA
B01	-47.765	Best	-47.765	-47.752
		Mean	-47.765	-46.960
		St.Dev	7.94E-15	1.92E-01
B02	961.715	Best	961.715	961.715
		Mean	961.715	961.716
		St.Dev	4.30E-06	9.59E-04
B03	-1.905	Best	-1.905	-1.905
		Mean	-1.905	-1.905
		St.Dev	0.00E+00	7.03E-08*
B04	-0.866025	Best	-0.866025	-0.866025
		Mean	-0.866025	-0.866014
		St.Dev	4.15E-17	7.17E-05
B05	-5.508	Best	-5.508	-5.508
		Mean	-5.508	-5.508
		St.Dev	0.00E+00	1.44E-09*

TS = Tasgetiren and Suganthan (2006), AMA= proposed Algorithm. * Standard deviation is positive due to rounding error

The best fitness measures may not reflect the true performance of the algorithms. This is due to the fact that EAs/MAs are stochastic search algorithms which may produce only a single best solution with many other poor solutions. The best solution might be considered here as an outlier. To avoid this, we can perform statistical

significance testing for comparing two algorithms. If the distribution of sample is not known, it is logical to use nonparametric tests such as Mann-Whitney-Wilcoxon (MWW) test (Anderson *et al.*, 1996; Conover, 1980), and Kolmogorov-Smirnov two sample test (Conover, 1980). However, these algorithms require the full datasets which is not available in the literature. A "goodness-of-fit" test such as the chi-square test (Conover, 1980) is performed on the dataset which shows the dataset of AMA follows approximately normal distribution. If other data sets are normally distributed, we can use student's *t*-test to compare the actual difference between two means in relation to the variation in the data. It is assumed that other datasets are approximately normal. Based on this assumption, student's *t*-test is performed to compare the algorithms.

The performance of CC is quite promising (achieved optimal in 8 problems), but it solved only 11 problems. However RY has solved all the 13 problems (g1-g13) and the performance of RY is clearly better than the other algorithms. The proposed AMA is compared with RY. The mean and standard deviation of 30 runs of both algorithms and the comparisons using the Student's *t*-test are presented in Table 5.5. The absolute value of *t* (indicated as *t*-C) is considered here. If the calculated *t* value exceeds the tabulated *t* value, then it can be said that the means are significantly different with 95% confidence levels. If there is a significant difference it is indicated "Yes", otherwise "No", in Table 5.5. If the results are significantly different then it is indicated that the algorithm with lower mean (for minimization problems) as the better algorithm in the final column. When there is no significant difference between the means, it is indicated Equal. The degrees of freedom is considered here 60 and *t*-tabulated value is 2.

From the *t*-test result in Table 5.5, we can see AMA is significantly better than RY in 4 test problems (g02, g06, g07 and g13), RY is better in 2 problems (g04, g09), and there is no significant difference between the results of these two algorithms for the other 7 problems. In those 7 problems both the algorithms have achieved optimal in 5 problems. It is worth noting that when RY is better, the difference is very tiny (small fraction of a percent). But when AMA is better, the difference is much larger (for example 25.03% improvement for problem g13).

This indicates that AMA is not only able to solve COPs but also performs better than

some other algorithms.

Table 5.5: Student's *t-test* between RY and AMA for 13 benchmark problems.

Fn	Optimal	Mean		St.Dev.		Significance Level		Better/ Equal?
		RY	AMA	RY	AMA	*t*-C	95%	
g01	-15.000	-15.000	-15.000	0.00E+00	0.00E+00	0.00	NO	Equal
g02	-0.803619	-0.781975	-0.803500	2.00E-02	2.20E-05	5.89	YES	AMA
g03	-1.000	-1.000	-1.000	1.90E-04	6.59E-06	0.00	NO	Equal
g04	-30665.539	-30665.539	-30665.537	2.00E-05	4.27E-04	24.93	YES	RY
g05	5126.498	5128.000	5148.966	3.50E+00	6.41E+01	1.79	NO	Equal
g06	-6961.814	-6875.940	-6961.804	1.60E+02	2.25E-03	2.94	YES	AMA
g07	24.306	24.374	24.315	6.60E-02	1.08E-14	4.91	YES	AMA
g08	-0.095825	-0.095825	-0.095825	2.60E-17	0.00E+00	0.00	NO	Equal
g09	680.630	680.656	680.671	3.40E-02	9.18E-03	2.28	YES	RY
g10	7049.331	7559.192	7479.064	5.30E+02	9.84E+01	0.81	NO	Equal
g11	0.750	0.750	0.750	8.00E-05	2.99E-08	0.00	NO	Equal
g12	-1.000	-1.000	-1.000	0.00E+00	0.00E+00	0.00	NO	Equal
g13	0.05395	0.067543	0.054020	3.10E-02	4.84E-05	2.39	YES	AMA

RY = Runarsson and Yao (2000), AMA= proposed Algorithm.

5.4 Effects of Operators and Parameters

In this section, a number of experiments is reported to show the effects of different search operators and parameters on the algorithm performance. In all these experiments, a fixed number of fitness evaluations is used for a fairer comparison.

5.4.1 LSLP

As mentioned earlier, a certain percentage (with probability P_L) of the agents are randomly selected to apply a life span learning process. Four types of LSLPs are designed here, and agents select one of them adaptively. This section investigates: what if AMA does not use LSLP, or just uses a particular LSLP always, or uses them adaptively?

To answer these questions, the following six sets of experiments were carried out:

- In the first set (No LSLP) AMA has not used any LSLP.
- In the second set (Random LSLP) AMA has used only LSLP type 1.
- In the third set (Restricted Random LSLP) AMA uses only LSLP type 2.
- The fourth set (Gradient-based LSLP) uses only LSLP of type 3.
- The fifth set (Directed LSLP) uses only LSLP of type 4.
- In the sixth set, the agent selects one of the LSLPs adaptively.

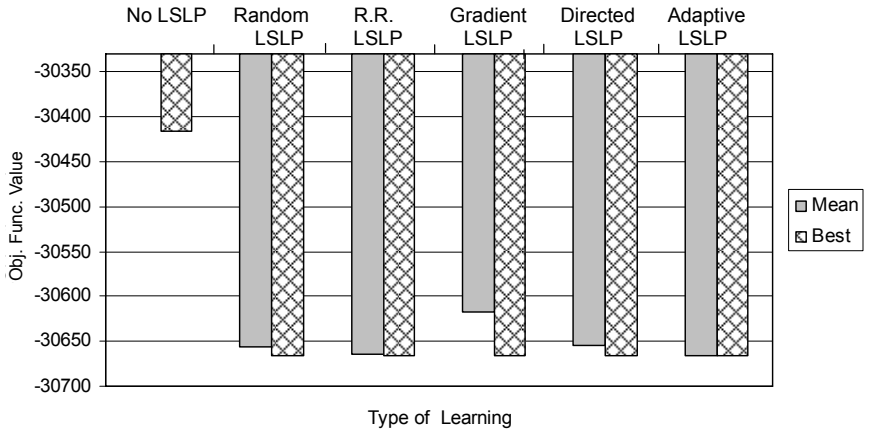
The best, mean, and standard deviations of the six sets for 30 independent runs are compared. Without LSLP (i.e. using only crossover) the results for all the test problems are worse than the other 5 sets. This is due to lack of diversity in the population which is ensured by the LSLP in this algorithm. The other five sets can all solve (and achieve the optimum) problems g01, g08, g12, B05 as these problems have a large feasible region compared to the other problems. Random and restricted random search based LSLP (Type 1 and 2) perform well, as both try to maintain diversity in the population. Sometimes random LSLP performs better than restricted random, and vice versa. However, they may suffer from over diversification e.g. in solving g07 Random LSLP's standard deviation was maximum. Directed search-based LSLP always tries to make the agent follow the previous generation's best agent, which may force the algorithm to converge. Sometimes it may cause convergence to a local optimum, which deteriorates the performance of the algorithms. For example, in solving problems g03, g07, g11, B04 the performance of Directed LSLP is worse than other sets (except No LSLP).

Gradient-based LSLP also tries to converge to the solutions as it changes the variables based on their effect on the objective function. For some problems it performs poorly e.g. g04, as the direction of producing the fastest rate of improvement in the fitness value is may no always best. The sixth set, i.e. adaptive approach in which the agent adaptively selects any of these four LSLPs, ensures both diversity and convergence. The performance of this set is the best of these six sets.

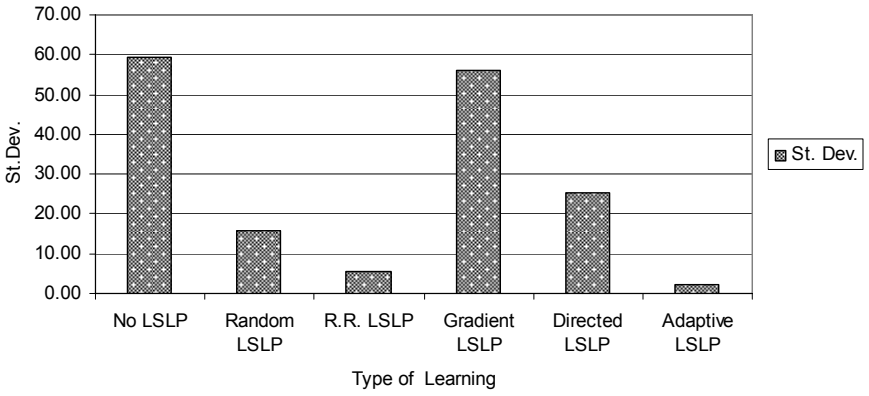
In some problems like g05, g13 the ratio of feasible space over the search space (i.e. the value of ρ) is very low and for some problems the value of ρ is very high like g02 (ρ =99.84%), g08 (ρ=85.60%). For the rest of this analysis g04 is used as an example as for this problem ρ is 52.123% which is in between the extreme values. Figure 5.1 presents the best, mean, and standard deviation of the 30 runs of these six experiments for test problem g04.

For this problem No LSLP's performance is worst. All other LSLPs achieve the optimum. In No LSLP the agents only use crossover, so after a while due to lack of diversity they converge to a local optimum. On the other hand Random and Restricted Random LSLPs try to change the variables randomly, aiming to maintain diversity, and have performed well. Directed LSLP tries to improve the present agent towards the previous generation's best, which shows satisfactory performance here. Gradient-based LSLP's performance is poor for g04. As this LSLP changes the variables based on their effect on the objective function, sometimes this may lead to a local optimum and may decrease the performance. The adaptive approach shows the best performance as it ensures both diversity and convergence.

From the experiments it is found that in adaptive LSLP the agents do not select any of LSLPs uniformly. On average, 80% of the time the agents prefer random and restricted random based LSLPs which try to diversify the population (the frequencies of random and restricted random based LSLP are respectively 68.45%, 12.31%). To ensure convergence they use gradient based and directed search on average 20% of the time. The frequency of directed search is the lowest, at less than 1%; this indicates that the agents rarely like to follow the previous generation's best as it may mislead towards local optima.



(1)



(2)

Figure 5.1: Effect of LSLP on g04. (1) Different types LSLPs vs. achieved best and mean results, (2) Different types LSLPs vs. St.Dev. of achieved results.

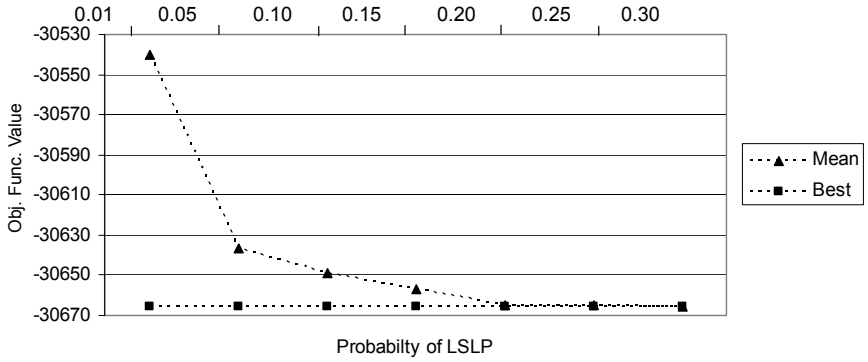
(All bar graphs start from zero. For the convenience of comparisons they are presented within a shorter range.)

5.4.2 Probability of using LSLP

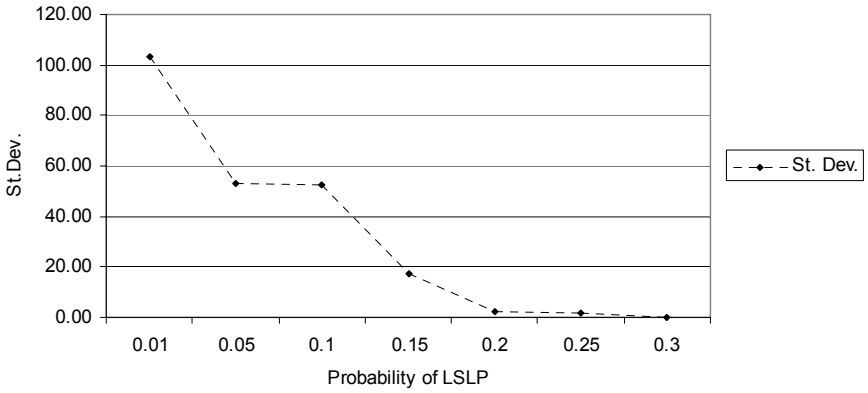
The probability of LSLP (P_L) is another important parameter of the algorithm. For low value of P_L a small number of agents are selected for LSLP, which allows lower diversity and so the algorithm may not perform well.

The performance of the algorithm is tested (over 30 runs each) with values for P_L of (0.01, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30). For most of the problems, the algorithm may

achieve the optimum as an outlier with any of these values of P_L. The mean and standard deviations improve with an increase of P_L, up to a point. For some easy problems like g08, g12, B05, any P_L over 0.05 achieves the same results. Sometimes a high value of P_L may over diversify the population, shown by high deviations in the results. In solving g03, g08, B04 the standard deviation of the results is more for P_L =0.3 than for lower values of P_L.



(1)



(2)

Figure 5.2: Effect of Probability of LSLP (P_L) on problem g04. (1) Probability of LSLP vs. achieved best and mean results, (2) Probability of LSLP vs. St.Dev. of achieved results.

Figure 5.2 demonstrated the best, mean and standard deviation of AMA in solving g04. It shows by increasing P_L both the mean and standard deviations improve gradually, but after 0.2 the results are not improved significantly. That indicates that increasing P_L can improve the performance of the algorithm, however, after certain point there will be no significant improvement.

5.4.3 Neighborhood Size

As discussed earlier, each agent communicates and interchanges information only with its allowed neighboring agents. The size of the neighborhood plays an important role in controlling the information diversification, hence indirectly controls the diversity and convergence of solutions. Higher neighborhood size means lower diversity of solutions as a better agent has a chance to dominate more agents in the entire lattice. Hence it will accelerate the convergence. However, we need to ensure enough diversity in the population to avoid the possibility getting trapped in local optima.

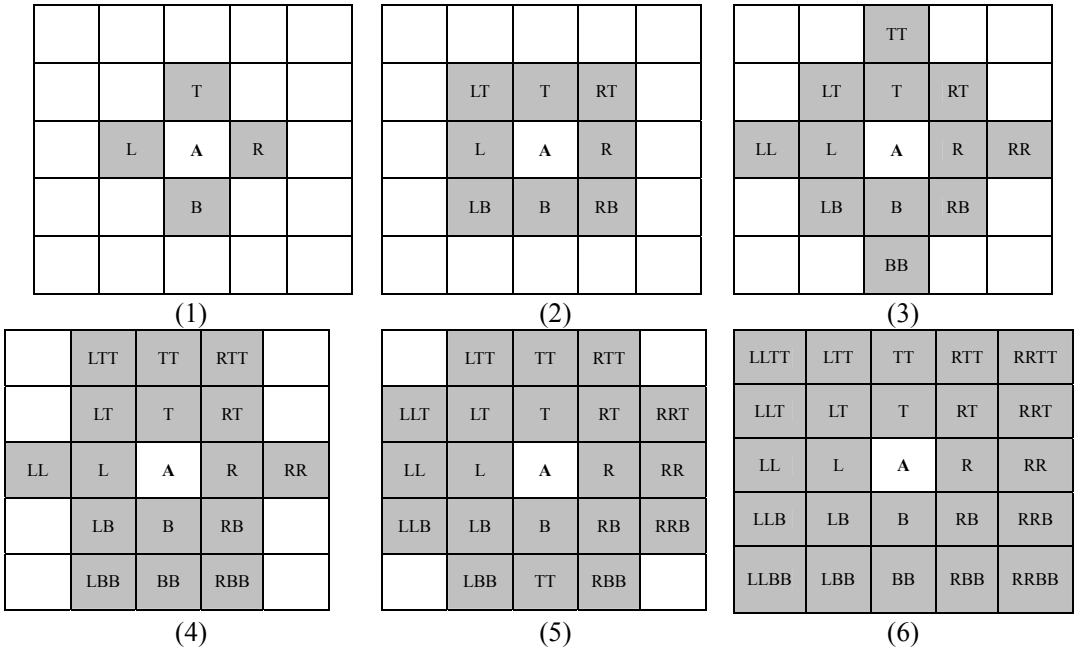
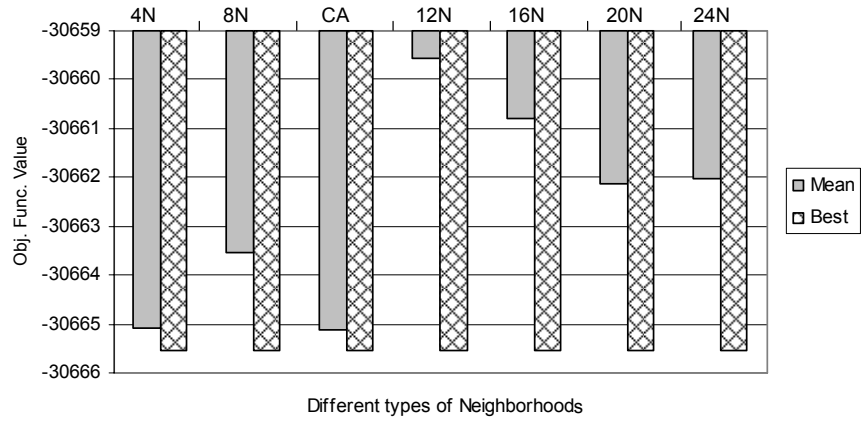


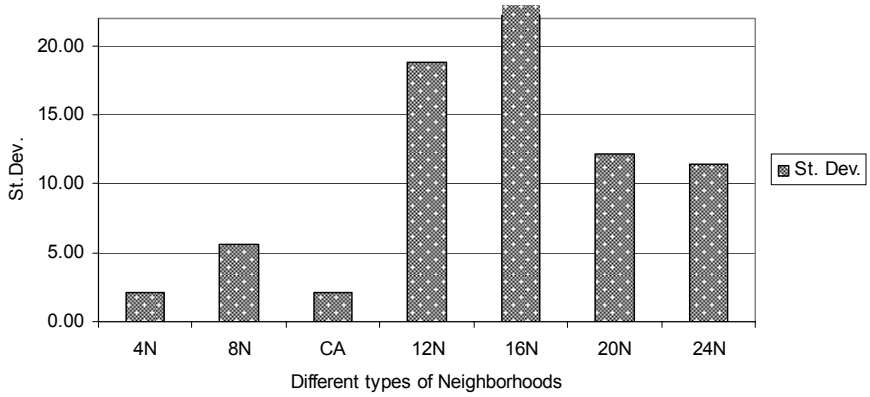
Figure 5.3: Different types of neighborhood. (1) Four Neighbors , (2) Eight Neighbors, (3) Twelve Neighbors, (4) Sixteen Neighbors, (5) Twenty Neighbors, (6) Twenty four Neighbors.

Several experiments have been carried out with different neighborhood sizes in order to find an appropriate size for better performance. We may consider the surrounding four Left (L), Right (R), Top (T) and Bottom (B) agents as neighbor agents; this neighborhood is indicated here as 4N. By increasing the number of neighbors from four to eight the agents are allowed to communicate and interchange information with more agents. As eight neighbors (8N), L, R, T, B, Left-Top (LT), Right-Top (RT), Left-Bottom (LB), and Right-Bottom (RB) agents are considered. The neighborhood size is extended to 12, 16, 20 and 24 neighbors. The different types of neighborhoods of an agent A are shown in Figure 5.3 by the shaded areas. Another neighborhood size is also considered which is called "Combined Approach", here the agents consider both four neighbors and eight neighbors interchangeably. Each approach is used for a certain number of generations (25% of the maximum number of generations) alternately. This approach is indicated as CA in the experiments.

Each agent mates with the best agents in its neighborhood and produces offspring. If the neighborhood size is larger, the overlapping of the neighborhoods in comparisons and competitions is higher. In this case, the dominant individuals tend to spread their genetic material throughout the population, which drives the population convergence prematurely. This is also reflected from the experiments: for most of the problems like g02, g04, g05, g06, g07, g09, g10, g13 the mean of the results of 12N, 16N, 20N, and 24N are worse than 4N, 8N and CA, though the best results achieved by all the approaches are optimal or very close to optimal. On the other hand, due to less overlapping, the 4N maintains good diversity in the population and performs better than 8N in g13, B01, B03, B04. However 8N helps for slower convergence than the larger neighborhood and performs even better than 4N in g06, B01. CA includes both the characteristics of 4N and 8N and achieves comparatively better results than any of these neighborhood sizes in g02, g04, g05, g06, g07, g09, g10, and g13. The performances of different neighborhoods are same for the g01, g03, g08, g11, g12, B05 and each of the approaches achieved the optimal and same mean.



(1)



(2)

Figure 5.4: Effect of Neighborhood size on problems g04. (1)Different Neighborhood size vs. achieved best and mean results, (2) Different Neighborhood size vs. St.Dev. of achieved results.

(All bar graphs start from zero. For the convenience of comparisons they are presented within a shorter range.)

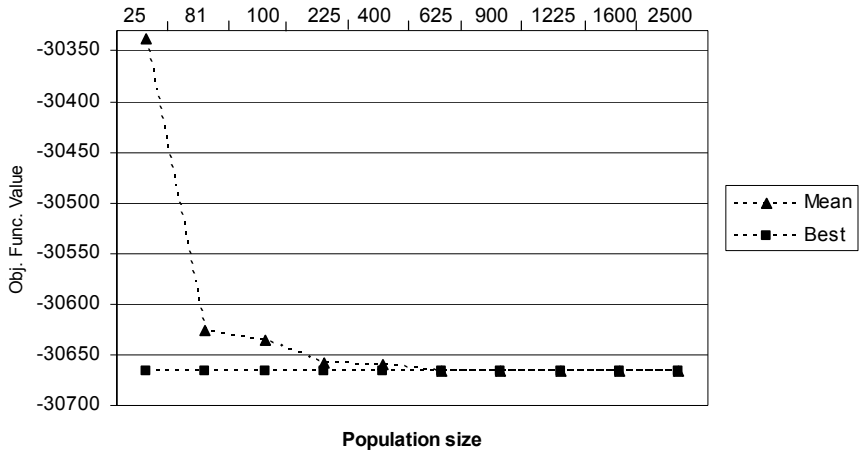
Figure 5.4 shows the best, mean and standard deviation achieved by the different neighborhood sizes for g04 in 30 independent runs. CA performs the best of the all approaches, however 4N and CA are very competitive and perform better (considering mean and standard deviation) than other neighborhood sizes. The best, mean, standard deviation achieved by CA are better than 4N.

5.4.4 Population Size

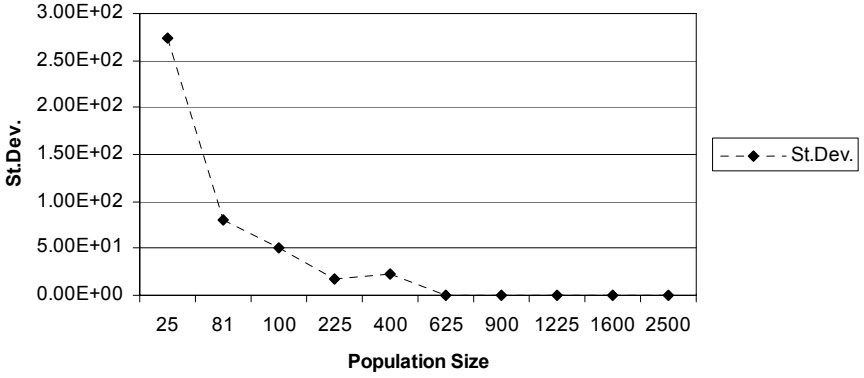
Population size affects the algorithm's performance. The agents start with randomly generated solutions within the boundary of each decision variable, then they communicate with others and exchange this information through the memetic operators to reach the goal. A larger population allows more communication of information and more diversity of initial solutions. As the total number of fitness evaluations (the population size multiplied by the number of generations) is fixed, the number of generations for a larger population size would be lower than that with a smaller population size.

A set of experiments is carried out with different population sizes: 25, 81, 100, 225, 400, 625, 900, 1225, 1600, and 2500 (as the agents are organized in $M \times M$, the population size is a square number). Keeping the same budget of fitness evaluations (350,000), the maximum number of generations used for them is 14000, 4320, 3500, 1555, 875, 560, 388, 285, 218, 140 respectively. The experimental results show that for very small population sizes (e.g. 25) the algorithm rarely finds the optimum, and the means of the results are also not impressive. For some easy problems like g01, g12, AMA can find the optimum and achieve good mean and standard deviations. By increasing population size the performance of the algorithms increases. However, after certain level of population increment the results do not improve significantly. For example in solving problems g02, g03, g04, g05, g08, g11, B01, B02 the performance of AMA increases with the increase of population size up to a population size of 400. Beyond 400, increasing the population size produces no significant improvement in performance.

Figure 5.5 shows the best, mean, and standard deviations of results of 30 runs for problem g04 with different population sizes. The optimum value for this problem is −30665.539. For all population sizes AMA has achieved the optimum (or very close to it) which is possible for AMA as a stochastic algorithm. However, for low population sizes (25, 81 or 100) the mean of best results from 30 runs of them could not reach −30650.0 and standard deviation is also large.



(1)



(2)

Figure 5.5: Effect of population size on problem g04. (1) Population size vs. achieved best and mean results, (2) Population size vs. St.Dev.of achieved results.

With the increase of population size the performance of the algorithm improves. If we consider the *t*-test result between population size 25 and 81, the performance is significantly improved at 95% confidence level, however for population size 81 to 100 the performance is not significantly changed. It has improved from 100 to 225 significantly and after that there is no significant difference in the performance by increasing the population size. This indicates that very low population sizes, despite

having more generations, cannot cover the whole search space. On the other hand, for large population size (above 225/400) AMA performs well however they continue for proportionally lower numbers of generations.

5.4.5 Crossover

In the preliminary framework of AMA, orthogonal crossover was used (this operator was proposed by (Leung, 2001). The orthogonal crossover operator acts on two parents and generates a set of new individuals from the search space defined by the two parents. The search space is quantized into a finite number of points, and then orthogonal design is applied to select a small but representative sample of points as potential offspring. Orthogonal crossover is computationally expensive as it generates several offspring from two parents, e.g. for problems containing more than 3 variables the orthogonal array $L_9(3^4)$ generates 9 offspring, where 3 is the number of *quantization level* and 4 is the number of *factors* (section 2.4.2 has discussed about this operator). For the same problem SBX generates only two offspring.

To compare the performance of AMA with both crossovers, AMA is executed with each of the crossover separately for the test problems. For the experiments other parameters are remain the same (e.g. population size 400, P_L =0.20). Table 5.6 and Table 5.7 show the results of AMA using SBX operator (indicated as AMA-SBX) and AMA using orthogonal crossover operator (indicated as AMA-OX), for the test problems g01-g13, and B01-B05.

While comparing the best results, both approaches achieve same results in 8 problems. AMA-OX performs better in three problems (g05, g06, and g10). For the remaining seven problems (g02, g07, g09, g13, B01, B02, and B04), AMA-SBX performs better in achieving best results. AMA-OX achieves better mean result in only g10. The mean results are same for three problems (g03, g08, and g12). However in g03 and g08 AMA-SBX achieves better standard deviations and same in g12. AMA-SBX achieves better mean results in the remaining all 14 problems. The experimental results show AMA with SBX is performing well. This crossover respects the interval schemata

processing and guarantees that the extent of the children is proportional to the extent of the parents (Deb and Agrawal, 1995; Ortiz-Boyer *et al.*, 2005). With SBX operator any arbitrary contiguous region can be searched, provided there is enough diversity maintained among the feasible parent solutions.(Deb, 2000).

Table 5.6: Comparison of results of AMA with SBX and Orthogonal crossover for 13 problems (g01-g13).

Fn	**Optimal**		**AMA-SBX**	**AMA-OX**
g01	-15.000	Best	-15.000	-15.000
		Mean	-15.000	-14.798
		St.Dev	0.00E+00	5.69E-01
g02	-0.803619	Best	-0.803619	-0.803514
		Mean	-0.803500	-0.798897
		St.Dev	2.20E-05	6.31E-03
g03	-1.000	Best	-1.000	-1.000
		Mean	-1.000	-1.000
		St.Dev	6.59E-06*	2.82E-05*
g04	-30665.539	Best	-30665.538	-30665.538
		Mean	-30665.537	-30660.315
		St.Dev	4.27E-04	9.38E+00
g05	5126.498	Best	5126.512	5126.509
		Mean	5148.966	5251.551
		St.Dev	6.41E+01	2.81E+01
g06	-6961.814	Best	-6961.807	-6961.810
		Mean	-6961.804	-6959.429
		St.Dev	2.25E-03	1.05E+01
g07	24.306	Best	24.315	24.317
		Mean	24.315	24.421
		St.Dev	1.08E-14*	7.56E-02
g08	-0.095825	Best	-0.095825	-0.095825
		Mean	-0.095825	-0.095825
		St.Dev	0.00E+00	7.62E-08*
g09	680.630	Best	680.645	680.689
		Mean	680.671	680.889
		St.Dev	9.18E-03	1.11E-01

Fn	Optimal		AMA-SBX	AMA-OX
g10	7049.331	Best	7280.436	7051.528
		Mean	7479.064	7323.868
		St.Dev	9.84E+01	2.06E+02
g11	0.750	Best	0.750	0.750
		Mean	0.750	0.762
		St.Dev	2.99E-08*	2.80E-02
g12	-1.000	Best	-1.000	-1.000
		Mean	-1.000	-1.000
		St.Dev	0.00E+00	0.00E+00
g13	0.053950	Best	0.053950	0.054007
		Mean	0.054020	0.259113
		St.Dev	4.84E-05	4.34E-01

* Standard deviation is positive due to rounding error.

Table 5.7: Comparison of results of AMA with SBX and Orthogonal Crossover for problems B01-B05.

Fn	Optimal		AMA-SBX	AMA-OX
B01	-47.765	Best	-47.752	-47.246
		Mean	-46.777	-46.475
		St.Dev	5.35E-01	3.14E-01
B02	961.715	Best	961.715	961.733
		Mean	961.722	968.085
		St.Dev	9.15E-03	5.46E+00
B03	-1.905	Best	-1.905	-1.905
		Mean	-1.905	-1.901
		St.Dev	7.20E-04*	4.57E-03
B04	-0.866025	Best	-0.866025	-0.866023
		Mean	-0.866014	-0.865841
		St.Dev	9.87E-06	2.31E-04
B05	-5.508	Best	-5.508	-5.508
		Mean	-5.508	-5.503
		St.Dev	9.03E-16*	4.36E-03

* Standard deviation is positive due to rounding error.

As SBX is simple and performs well in solving these types of problems with multiple optimal solutions with a narrow global basin, SBX is preferable for the AMA

framework.

5.4.6 Section Summary

In this section, experiments showed how the performance of the proposed AMA is affected by various parameter settings and design decisions, and provided explanations from the observations. It is established that:

- It is best to use adaptive approach in selecting a LSLP, which allows an agent adaptively to select any of these four LSLPs, ensuring both diversity and convergence.
- It is suggested to use a value of 0.20 for probability of learning P_L, which provides a balance between the diversity in the populations and performance of the algorithm.
- The size of the neighborhood plays an important role in controlling the information diversification. The "Combined Approach" shows better performance as it provides indirectly a balanced control of the diversity and convergence of solutions.
- Population size also affects the algorithm's performance. The experimental study shows the performance AMA performs best with population size 400.

5.5 Chapter Summary

This chapter investigates the performance of the AMA in solving a set of test problems which includes five new problems plus 13 existing well-known problems. The results show the proposed algorithm is robust in its handling of both linear and nonlinear equality and inequality constraints. As each of the agents exchanges information with its neighbors, AMA does not need any ranking for the whole population. The constraint handling techniques used here do not need any penalty functions or parameters. The agent selects a neighborhood agent by using pair-wise comparison to mate, which handles the constraints indirectly. Also in the self-adaptation

process of learning, while calculating the improvement index, the constraints are indirectly handled. These two levels of constraint handling with appropriate neighborhood size, SBX crossover, and LSLP ensure the superior performance of AMA in handling constraints.

The algorithm shows very impressive performance by achieving optimal results in 13 problems. The performance of the AMA is compared with five GA-based and one ES-based algorithms. The comparison results show that the proposed approach gives mostly improved or comparable results to other algorithms. A statistical significance tests is used and the results show the proposed algorithm's performance is better than the ES-based algorithms for the well-known 13 problems.

The effect of the proposed LSLPs is analyzed, showing that adaptively selecting one of the LSLPs achieves better results ensuring both diversity and convergence. Probability of Learning (P_L) is also an important parameter; the performance of the algorithm increases with the increase of P_L, but after a certain level it causes over diversification. The size of neighborhood also affects the performance of the algorithm. The combined approach (i.e. applying 4 neighbors and 8 neighbors interchangeably) performs better than the other types of neighborhood. The effect of population size is also investigated, which shows a low population size is not able to achieve good results. With the increase of population size the performance improves, however after a certain population size there is no significant improvement in the results.

The next chapter concentrates more in solving COPs where the feasible space is very tiny in comparison to the search space, which makes it hard for the algorithms to find even the feasible space.

Chapter 6

Problems with Tiny Feasible Space

The quality of individuals in the initial population influences the performance of evolutionary algorithms, especially when the feasible region of the constrained optimization problems is very tiny in comparison to the entire search space. This chapter proposes a simple method to improve the quality of randomly generated initial solutions by sacrificing very little in diversity of the population. The proposed method, which is recognized as the Search Space Reduction Technique (SSRT) in this thesis, directs the selected low quality infeasible solutions towards the feasible space. The performance of the proposed technique is tested using five different EAs by solving a number of state-of-the-art test problems and a real world case problem. The experimental results show SSRT improves the solution qualities as well as speeds up the performance of the algorithm.

6.1 Introduction

In many practical optimization problems, the feasible spaces are very tiny. These problems are very challenging as it requires searching a huge variable space in order to locate feasible points with acceptable quality. To solve problems with tiny feasible space, EAs usually take a long time to find even feasible solutions. With good quality initial solutions, the search operators reach the feasible region quickly and find better solutions. As the initial populations of EAs are randomly generated, they may not be good quality solutions. A careful preprocessing, with little sacrifice in diversity, can improve the initial solutions, which not only accelerates the convergence but also leads

to better solutions.

In this chapter, to improve the quality of the initial population, a simple search space reduction technique is presented, which can be considered as preprocessing or an additional step before applying EAs for solving constrained optimization problems. The COPs are considered having tiny feasible region when ρ (ratio between the feasible region and search space) is less than 1%. The main idea of SSRT is to move some of the poor quality infeasible solutions towards the feasible region. As the initial population of EAs is randomly generated to ensure diversity, it may cause delay in reaching a reasonably good solution for tiny feasible space. Once a good solution point is obtained, EAs usually converge nicely to an acceptable solution. To enhance the performance of the algorithm in reaching the feasible space quickly, the proposed SSRT guides the initial population to move towards the feasible region. The method finds a centroid from the initial feasible solutions (if any) with some infeasible solutions around the feasible space (i.e. solutions with lower constraint violations). A certain percentage of the worse infeasible solutions are then encouraged to move towards the centroid (only some are moved, not all, to maintain a certain level of diversity). Such a move would help certain individuals to reach the feasible region quickly by improving the solution quality.

By applying SSRT the randomly generated initial solutions are no longer random, rather they are directed towards the feasible space, which helps the algorithms to reach the feasible region faster and improve the solution quality. However, in implementing the process appropriately, the following questions must be answered.

- When should be the SSRT applied?
- How to calculate the centroid for SSRT?
- How long shall be SSRT applied as it decreases the diversity?

The experiments aim to find the answers to these questions.

Initially the SSRT has been tested with the agent-based memetic algorithms (presented in chapter 4). From the initial experiments and analysis of SSRT with AMA, it is clear that the performance of any EAs can be enhanced by incorporating SSRT in

solving constrained optimization problems with tiny feasible region. Then an extensive investigation is made for the effectiveness of SSRT by incorporating it with simple genetic algorithm and three other well known algorithms such as Deb *et al.* (2002), Elfeky *et al.* (2006), and Sarker and Ray (2005) from the literature for a set of benchmark problems with tiny search space and a real world case problem (Sarker and Quaddus, 2002; Sarker and Ray, 2009). The experiments show SSRT improves the solution qualities as well as speeds up the performance of the algorithms.

The rest of this chapter is organized as follows. Section 6.2 describes the proposed search space reduction technique and its different issues. Section 6.3 describes the performance of the proposed approach with different algorithms on the test problems and the effect of parameters used in SSRT. Finally, Section 6.4 concludes the chapter.

6.2 Search Space Reduction Technique

In a population-based method, such as EA, it is not expected that the random initial solutions would always be of good quality. Some algorithms like GENOCOP (Michalewicz, 1994; Michalewicz and Janikow, 1996) assume a feasible starting point (or feasible initial population), which implies that the user or the EA must have a way of generating (in a reasonable time) such a starting point. The homomorphous mapping method of Koziel and Michalewicz (1999) also requires an initial feasible solution.

The proposed SSRT deals with the initial randomly generated populations. It allows the most infeasible individuals to move towards the feasible region, before the evolutionary process starts, which is basically squeezing the search space. That means the evolutionary process starts with a better population in a reduced search space.

If there are no or few (less than 1%) feasible solutions in the initial random population, the EAs are allowed to apply SSRT. The infeasible solutions are then ranked based on the extent of constraints violation. The feasible solutions (if any) and a certain percentage of the top ranked infeasible individuals are then used to find a centroid. If there is no feasible individual, only the top ranked infeasible individuals are first

modified to reduce the violation of constraints and then use them to calculate the centroid.

After calculating the centroid a certain percentage of worst infeasible solutions in the population are allowed to move towards the centroid. Although this process guides the worst infeasible solutions towards the feasible space, it reduces the diversity of the population. To ensure diversity only a small number of the worst infeasible solutions are allowed to follow the centroid, and discontinue the process when the diversity of initial population is decreased to a certain level.

Figure 6.1 demonstrates the working principle of SSRT. First it calculates a centroid. If there are not enough feasible solutions (in the figure only one feasible) in the population, it considers a percentage of good quality infeasible solutions (shown in shaded area) to calculate the centroid. These good quality infeasible solutions are mentioned as allowable infeasible solutions. After that a portion of the non-allowable infeasible solutions (i.e. worst infeasible solutions in the population) move towards the centroid, which is shown by the arrows.

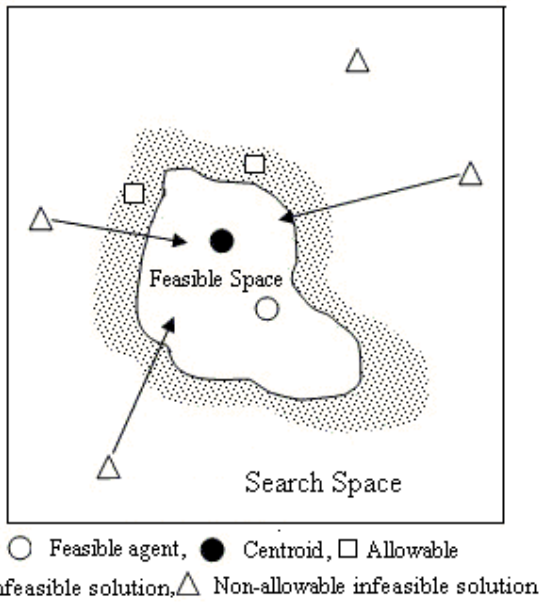


Figure 6.1: Search Space Reduction Technique

The proposed algorithm for SSRT is given below:

Step 1. Rank the infeasible solutions based on the Constraint Violation (CV). Define the allowable infeasible range. Calculate the diversity of the population.

Step 2. Check the number of feasible individuals. If it is more than 1% of the population, go to step 7. Otherwise go to step 3.

Step 3. If there are feasible individuals, calculate the centroid using the feasible and allowable infeasible individuals and go to step 5, else go to step 4.

Step 4. Select the top ranked infeasible solution (i.e. the best infeasible individual based on the CV).

a. Find the constraint which has maximum CV for this individual.
b. Select a variable randomly, which is involved in the constraint which is not yet modified.
c. Change the variable with $\pm\delta$ and mark as modified.
d. If the individual became feasible, go to step 2, else go to step 4e.
e. If all constraints are checked or all the variables are modified, then find the centroid of the allowable infeasible along with this one, otherwise go to step 4.

Step 5. Force a certain percentage of the non-allowable infeasible solutions to follow the centroid.

Step 6. Calculate the diversity of the population. If the diversity decreases up to a certain level then go to step 7, otherwise go to step 2.

Step 7. Stop.

Here the value of δ is small; $\delta = |G(0,1)|$, where $G(0,1)$ is a Gaussian random number generator with zero mean and standard deviation 1.

For calculating the centroid, for each variable of the centroid x_i^c, the arithmetic mean of the participating solutions of respective variables x_i is considered. For calculating the diversity, the mean Euclidian distance of all solutions from the centroid is taken.

A portion of the non-allowable solutions follow the centroid as follows:

$$x_i^n = \alpha\, x_i + (1-\alpha)x_i^c, \quad i = 1 \ldots\ldots n \tag{6.1}$$

Where x_i^n and x_i^c are the i^{th} variable of the non-allowable solution and the centroid respectively, n is the number of variables in the solution vector and α is a uniform random number from 0 to1.

6.2.1 Computational Cost

Since SSRT is applied to the randomly generated initial population before the evolutionary process, it will increase the computational cost to the selected EA. In the algorithm Step 1 needs to sort the infeasible solutions based on the constraint violations, and so simple bubble sort can be used, which needs $O(M^2)$ comparisons for worst case if the initial population contain M infeasible solutions. However if the size of M is very large then it would be better to replace the bubble sort with other efficient sorting algorithm, e.g. quick sort which needs less computational complexity $O(M \log_2 M)$. Step 3a requires finding the most violated constraint, which can be done by using normal linear search. If the problem needs to satisfy R number of constraints ($R = p$ inequality constraints + q equality constraints) then the linear search needs $O(R)$ comparisons for worst case. So the overall complexity of the SSRT is $O(M^2)$, which is governed by the sorting. We can consider this additional complexity acceptable if the solution quality of EAs is enhanced by SSRT.

6.2.2 Issues Regarding SSRT

The proposed SSRT should be applied when there are very few feasible solutions (e.g. less than 1% of the population size) in the initial random population. In SSRT, a centroid is calculated to guide the worst individuals. Using certain feasible and top ranked infeasible solutions the centroid is calculated. A good quality centroid will guide

the worst individuals to move towards the feasible space. In finding a quality centroid, we need to decide an appropriate number (or percentage) of infeasible solutions that will be used in calculating the centroid.

Another important issue in designing SSRT is the stopping criterion for SSRT. As the centroid attracts the other low quality individuals towards it, the diversity of the population decreases. However in any population based search algorithm, the diversity is an important factor. As diversity of the population decreases with application of SSRT, how long we should apply the SSRT?

Experimental studies are made regarding these issues in the next section.

6.3 Experimental Results and Discussions

The issue of dealing with constraints has long been difficult for optimization methods (Takahashi *et al.*, 2003). It has become harder when the feasible region is very tiny. In this study only those benchmark problems are chosen, used in Chapter 5, whose feasible region is very small compared to their search space. To get an estimate of how tiny is the feasible space of these problems, a metric suggested by Michalewicz and Schoenauer (1996) is used, $\rho = |F|/|S|$, where $|S|$ is the number of random solutions generated (1,000,000 in this case), and $|F|$ is the number of feasible solutions found (out of the total randomly generated solutions). Here only those problems are considered, whose ρ is less than 1%. The characteristics of these benchmark problems and the optimal values are shown in Table 6.1.

In this section first an investigation is made for the performance of SSRT on AMA with the benchmark problems and the effect of parameters used in SSRT. Then it discusses the effectiveness of SSRT by incorporating it with simple genetic algorithm and three different existing EAs for solving this type of constrained problems with tiny feasible solutions.

The research has not been restricted only on the benchmark problems: the

performance of SSRT is also tested on a real world case problem collected from the literature (Sarker and Quaddus, 2002; Sarker and Ray, 2009). Details of these experimentations are discussed in section 6.3.3.

Table 6.1: Characteristics and the optimal results of the benchmark problems

Prob	(n)	Obj. Fuc.	ρ	LI	NI	LE	NE	AC	Optimal
g01	13	Quadratic	0.0111%	9	0	0	0	6	-15.000
g03	10	Polynomial	0.0000%	0	0	0	1	1	-1.000
g05	4	Cubic	0.0000%	2	0	0	3	3	5126.498
g06	2	Cubic	0.0066%	0	2	0	0	2	-6961.814
g07	10	Quadratic	0.0003%	3	5	0	0	6	24.306
g08	2	Nonlinear	0.8560%	0	2	0	0	0	-0.095825
g09	7	Polynomial	0.5121%	0	4	0	0	2	680.630
g10	8	Linear	0.0010%	3	3	0	0	6	7049.331
g11	2	Quadratic	0.0000%	0	0	0	1	1	0.750
g13	5	Nonlinear	0.0000%	0	0	0	3	3	0.053950
B01	10	Nonlinear	0.0000%	0	0	3	0	3	-47.765
B02	3	Quadratic	0.0000%	0	0	1	1	2	961.715
B03	5	Nonlinear	0.0204%	4	34	0	0	4	-1.905
B04	9	Quadratic	0.0000%	0	13	0	0	6	-0.866025

ρ = Ratio between the feasible space and the search space, LI=Linear Inequalities, NI=Nonlinear Inequalities, LE= Linear Equalities, NE= Nonlinear Equalities, AC=Active Constraints

6.3.1 Experimentation with AMA

In this research, two sets of experiments are carried out for AMA to justify the necessity of SSRT on AMA. In this chapter AMA used the same parameters proposed in chapter 5.

- In the first set, 30 sets of initial populations are generated randomly for each test problem under consideration, and then AMA is used to solve the problems with the same initial population in 30 independent runs.

- In the second set of experiment, SSRT is applied to all the initial populations randomly generated in the first set of experiments, and then AMA uses the same modified populations for solving the each test problem in 30 independent runs.

The maximum number of generations is set in this experiment at 3500. The allowable range (AR) of infeasible solutions for calculating the centroid is at most 50% of the infeasible solutions and maximum diversity reduction allowed from the initial random population is 10%. The initial solution vectors for the solutions are randomly generated within the bounds of each decision variable. The probability of learning used here is 0.2. As higher population size initially provides enough diversity and AMA achieved good quality solutions with that (presented in chapter 5), here lower population size (100) is used to see the effect of SSRT more clearly.

The two sets of results of AMA are compared to see whether the results are improved with SSRT, and how fast the algorithms converge to the best results. The amount of time is also calculated to see how much time is expended for the execution. The best, median, mean, standard deviation, and worst results of 30 independent runs (with 30 different random seeds) are given in Table 6.2. The last two columns of Table 6.2 shows the average number of generations required to find the best results (as an indication of how quickly the algorithm achieves the quality solutions) and the whole execution time.

From Table 6.2 we can see AMA with SSRT has achieved better results in different aspects than without SSRT (bold fonts indicates better achievements of AMA with SSRT). If we consider the mean results, AMA with SSRT has achieved better mean results in 78.57% of cases and the same mean in the other 21.43% of problems. It has improved the mean results by more than 5% in several problems, such as 71.44% in problem g13, 5.71% in g01, and 5.61% in g06. The improvements in other problems are also remarkable.

AMA with SSRT has also performed better in achieving the best results in 35.71% of the problems, and the same best results in 42.85% of problems. On 78.57% times the worst results and on 57.14% times the median results of AMA with SSRT were better than only AMA.

Table 6.2: Performance of AMA with and without SSRT from 30 independent runs.

Fn	AMA	Best	Median	Mean	StDev	Worst	AvgGen	Time(s)
g01	NST	-15.000	-15.000	-14.190	1.30E+00	-10.109	3027.467	81.75
	ST	-15.000	-15.000	**-15.000**	**3.78E-09***	**-15.000**	3083.067	85.62
g03	NST	-1.000	-1.000	-1.000	2.76E-06*	-1.000	3500	80.86
	ST	-1.000	-1.000	-1.000	**4.32E-06***	-1.000	3500	69.91
g05	NST	5127.388	5186.761	5208.999	8.20E+01	5463.927	3500	27.62
	ST	5127.457	5226.753	**5200.580**	**4.94E+01**	**5249.292**	3500	28.08
g06	NST	-6961.813	-6961.809	-6592.092	1.38E+03	-1204.787	3352.4	20.78
	ST	-6961.813	**-6961.811**	**-6961.974**	**6.18E-01**	**-6961.810**	3375.6	14.47
g07	NST	24.329	24.436	24.510	3.12E-01	25.958	3191.4	207.66
	ST	**24.324**	**24.358**	**24.360**	**1.55E-02**	**24.384**	3057.8	150.64
g08	NST	-0.095825	-0.095825	-0.095825	1.49E-08*	-0.095825	1558.2	15.27
	ST	-0.095825	-0.095825	-0.095825	**4.23E-17***	-0.095825	1766.867	16.43
g09	NST	680.731	681.177	681.714	1.06E+00	684.146	2482.267	54.31
	ST	**680.659**	**680.906**	**680.882**	**9.80E-02**	**680.965**	2204.733	52.54
g10	NST	7077.282	7444.316	7453.023	2.79E+02	8062.957	3172.233	30.22
	ST	**7058.727**	**7143.765**	**7155.326**	**7.05E+01**	**7276.031**	3039.933	33.24
g11	NST	0.750	0.750	0.750	3.35E-03*	0.750	3500	14.92
	ST	0.750	0.750	0.750	**4.92E-09***	0.750	3500	15.24
g13	NST	0.054212	0.065926	0.193989	2.07E-01	0.811584	3500	40.72
	ST	**0.053969**	**0.055524**	**0.055411**	**9.22E-04**	**0.056924**	3500	46.17
B01	NST	-46.542	-44.950	-44.652	1.63E+00	-39.808	3500	85.48
	ST	**-47.292**	-45.594	**-45.870**	**5.39E-01**	**-45.293**	3500	85.43
B02	NST	961.716	964.836	965.085	2.39E+00	970.210	3500	17.98
	ST	961.721	**963.914**	**963.134**	**1.18E+00**	**964.474**	3500	19.33
B03	NST	-1.905	-1.884	-1.878	3.83E-02	-1.694	3308.567	45.95
	ST	-1.905	**-1.900**	**-1.900**	**3.14E-03**	**-1.894**	3476.6	48.13
B04	NST	-0.866022	-0.865896	-0.865748	3.73E-04	-0.864446	3301.4	117.61
	ST	-0.866018	**-0.865988**	**-0.865988**	**1.62E-05**	**-0.865960**	3135.467	106.22

NST= AMA without SSRT, ST = AMA with SSRT. Avg.Gen.= Average generation required to find best results, **Bold** font indicates the best result achieved by AMA with SSRT, * indicates though the best, worst, median, and mean results are the same, standard deviation is positive due to rounding error.

If we consider the average number of generations required to find out the best result, in 28.5% of problems AMA with SSRT is faster than the other approach. For all problems on average AMA needed 3170.995 generations to find the best results while AMA with SSRT needed 3152.862 generations. For the problems involving equality constraints (g03, g05, g11, g13, g14, g15) the dynamic relaxation (described in previous chapter) is used and so it is considered that the best results are found in both approaches after the algorithms are terminated (i.e after 3500 generations). For this reason in 54.55% of problems each approach needs the same number of generations. However if we exclude those problems from this calculation then AMA with SSRT reduced by an average of 1.09% (i.e, 31.73 generation per problem) the required average number of generations to find out the best result.

Not only that, if AMA is executed with the initial population generated by SSRT, then it takes less time for the whole execution than the time required by AMA with random initial population. The last column of Table 6.2 shows the amount of time required by each approach. It shows AMA with randomly generated initial population needs on average 60.08 seconds to solve a problem. On the other hand AMA with SSRT generated initial population needs only 55.10 seconds, which saves 8.28% of the execution time of AMA.

So we can say by applying SSRT in most of the problems AMA has improved either the solution or computational time or both.

Now the effects of parameters used in SSRT on AMA shall be discussed.

In applying SSRT, a centroid is calculated using certain feasible and top ranked infeasible solutions. In the process, we need to decide the number (or percentage) of infeasible solutions in calculating the centroid, and a stopping criteria for SSRT. As the diversity of the population decreases with application of SSRT, diversity measure is used as a stopping criterion. These two parameters will be discussed in the following subsections.

6.3.1.1 Effect of Allowable Range (AR) in calculating SSRT

While calculating the centroid, if we allow all of the infeasible solutions with the feasible solutions (if any), it may move the centroid towards an infeasible region. In the process, the infeasible solutions are ranked based on their constraint violation and consider a certain percentage of the top ranked infeasible solutions. As these participating solutions are better than the others, it is expected that the centroid may have better fitness than the low ranked solutions. To see the effect of the percentage of top ranked infeasible solutions used in SSRT on the overall solution, experiments have been carried out by varying the percentage of the top ranked infeasible solutions (from 10% to 50% with an increment of 10%) while leaving the other parameters constant e.g. population size 100, P_L 0.2, Diversity Reduction (DR) 10%. The solutions are compared with that of AMA without SSRT.

The best, mean, standard deviations, worst, median, and average number of generations required to find out the best result for 30 independent runs are compared. As the centroid guides the worse infeasible solutions, the quality of the centroid plays a vital role to the performance of the algorithms. If we consider a very small number of top ranked infeasible solutions with feasible solutions (if any, since the feasible space is very tiny), they may not provide a good quality centroid. If we increase the allowable range (AR), there is a high chance that it will produce a good quality centroid. However, the range of AR should not be too large, since the use of higher percentage could mislead the search process where multiple disjointed feasible spaces exist for a problem. In most of these test problems AR ranges 30% to 50% provides better results.

In Table 6.3 the results of problem g01 are shown as an example. For the best results of the 30 runs, in all the cases AMA has achieved the optimal. When we have considered 50% top infeasible to find the centroid, the performance of the algorithm is the best among the 6 sets of results. We should not consider too many infeasible solutions, which may not help the solutions rather direct them to other areas of the search space, resulting in longer processing time. Though the test problems are diverse in nature, in solving most of the problems AMA shows similar behavior.

Table 6.3: Effect of Allowable Range (AR) for calculating centroid on problem g01

AR (%)	Best	Mean	St. Dev.	Worst	Median	AvgGen
0	-15.000	-14.921875	2.97E-01	-13.828125	-15.000	3211.98
10	-15.000	-14.960937	2.14E-01	-13.828123	-15.000	3056.19
20	-15.000	-14.960937	2.14E-01	-13.828124	-15.000	3146.19
30	-15.000	-14.960937	2.14E-01	-13.828125	-15.000	3198.01
40	-15.000	-14.960937	2.14E-01	-13.828125	-15.000	3178.89
50	-15.000	-15.000000	3.78E-09	-15.000000	-15.000	3083.07

AR= Allowable range of infeasible solutions for centroid, Avg Gen = Average number of generation required to find the best result.

6.3.1.2 Effect of Diversity Reduction (DR)

The diversity of the population decreases when the low ranked infeasible solutions move towards the centriod. If the diversity of the population decreases too much then the performance of the algorithm also deteriorates. So for SSRT it is a critical issue to maintain diversity while attracting the low ranked infeasible solutions towards the feasible region. A small reduction of the diversity of the population by applying SSRT improves the performance of AMA. By applying SSRT, we can still provide sufficient diversity by controlling the diversity reduction.

To show the effect of reducing diversity during SSRT, a set experiments is carried out with different percentage of diversity reduction (10% to 50% with an increment of 10%) while keeping other parameters constant (Population size 100, P_L 0.20, AR 50%). SSRT is stopped when the diversity reduced to a certain percent (e.g. 10% for the first experiment) from the initial stage. In general, a low range of diversity reduction (10-20%) improves the performance of AMA. However a higher value of diversity deteriorates the quality of performance due to the lack of diversity in the population. The Results of the experiment for 30 independent runs for problem g01 are given in Table 6.4.

The experimental results show the small reduction of diversity like 10%-20% gives the algorithm better performance. However a large amount of diversity reduction is not

helping the AMA significantly. If we consider the mean, standard deviation and worst results, the performance of AMA is best with 10% relative diversity reduction. That indicates SSRT improves the performance of the algorithm, however we need to ensure enough diversity in the population although SSRT reduces the diversity up to certain level.

Table 6.4: Effect of Diversity Reduction (DR) on problem g01

DR(%)	Best	Mean	St. Dev.	Worst	Median	AvgGen
0	-15.000	-14.921875	2.97E-01	-13.828125	-15.000	3165.19
10	-15.000	-15.000000	3.78E-09	-15.000000	-15.000	3083.07
20	-15.000	-15.000000	2.91E-08	-15.000000	-15.000	3126.52
30	-15.000	-14.960937	2.14E-01	-13.828123	-15.000	3073.60
40	-15.000	-14.960937	2.14E-01	-13.828123	-15.000	2983.95
50	-15.000	-14.960937	2.14E-01	-13.828123	-15.000	3011.67

DR= Relative Diversity Reduction from the initial randomly generated population after SSRT, Avg Gen = Average number of generations required to find the best result.

6.3.1.3 Summary

The experiments described in this section show how the performance of SSRT is affected by various parameter settings. It is shown that:

- The allowable range for calculating the centroid should not be too large, since the use of higher percentage could mislead the search process. For best performance it is suggested to use AR in the range of 30% to 50%.
- Since SSRT reduces the diversity in the population, the experimental study shows the algorithm performs best with 10% relative diversity reduction.

For the experiments described in the next section, AR is 50% and relative diversity reduction is 10%.

6.3.2 Experimentation with other Evolutionary Algorithms

This section investigates the performance of SSRT with other evolutionary

algorithms. Three well known Evolutionary Algorithms by Deb *et al.*(2002), Elfeky *et al.* (2006), and Sarker and Ray (2005), and simple genetic algorithms have been chosen to evaluate the effect of SSRT.

A simple genetic algorithm (SGA) (same as chapter 3) using tournament selection, SBX crossover (Deb and Agrawal, 1995) and parameter based mutation operator (Deb, 2000) is used to see the performance of SSRT in solving constrained optimization problems with tiny feasible space.

Deb *et al.* (2002) have proposed a computationally fast and elitist multi-objective evolutionary algorithm, based on a nondominated sorting approach called NSGA-II. The difference between the conventional single objective GA and NSGA-II lies with the assignment of fitness of an individual. The fitness of an individual in NSGA-II is based on the non-domination level of an individual. Moreover, they have modified the definition of dominance in order to solve constrained multi-objective problems efficiently. NSGA-II is a well-known and well-accepted algorithm for solving both single- and multi-objective constrained optimization problems.

Multi-objective Constrained Algorithm (MCA) proposed by Sarker and Ray (2005) is a close variant of NSGA-II. It has two major differences, which include the selection of parents and the process of population reduction. The process is more computationally expensive than NSGA-II, and can be thought as a diversity maintaining mechanism which might be useful for problems where the diversity in the variable space is important. This algorithm can also be used for solving both single- and multi-objective constrained optimization problems. MCA performs better than NSGA-II for some special cases of constrained optimization problems (Sarker and Ray, 2005).

The previous chapter mentioned Elfeky *et al.*'s (2006) GA. They have introduced new ranking, selection, and triangular crossover methods to solve constrained optimization problems. They have exploited some features of constrained problems in the algorithm. This algorithm is indicated in this chapter as TC.

As with AMA, all these algorithms have been applied in two sets of experiments. In the first set, each of the algorithms is executed for 30 independent runs with 30 sets of

randomly generated initial populations. In the second set of experiments, each of the algorithms is executed for 30 independent runs with SSRT applied to the initial populations taken from the first set of experiments.

These two sets of results of each algorithm are then compared to see whether the results are improved with SSRT. In all four algorithms, a fixed set of parameters are used for SSRT (probability of crossover 0.90, probability of mutation 0.20), as our objective is to test the influence of SSRT on the performance of algorithms and we are not interested here in finding the best set of parameters corresponding to the best solutions. In addition, as we are focusing on a given aspect of the algorithm, it is not appropriate to make a general conclusion on the overall performance of the algorithms. As with AMA the maximum number of generations is set at 3500 and the population size is 100. The maximum number of fitness evaluations is set in this experiment at 350,000. For the best performance of TC, we have used the population size 30, probability of crossover 0.8 and probability of mutation 0.1 as suggested in (Elfeky *et al.*, 2006). Despite using less population size, the algorithm was applied up to 350,000 fitness evaluation counts. The allowable range (AR) of infeasible solution for calculating centroid is used at most 50% of the infeasible solutions, and the maximum diversity reduction from the initial random population is 10%.

The best, mean, standard deviation, worst, and median results of 30 independent runs (with 30 different random seeds) of the four algorithms are given in Table 6.5, 6.6, 6.7 and 6.8 and summarized in Table 6.9. Numbers in boldface mean that algorithms using SSRT achieved better result than without using SSRT. '×' symbol indicates that the particular algorithm could not achieve feasible solutions in all of the 30 independent runs.

From Table 6.5 we can see that SSRT has enhanced the performance of SGA in most of the problems. Although SGA without SSRT could not achieve any feasible solutions for g03, g05, and g11, with the help of SSRT it found a feasible solution in g05. For the other 11 problems, if we consider the mean results, in 81.82% of problems SSRT improves the performance of SGA. In 45.45% of problems the mean results are improved more than 5%, such as 5.45% in g01, 7.26% in g07, 9.36% in g10, 33.21% in

g13, and 8.83% in B04. While achieving the best results, 54.55% of the time SSRT improves the performance of SGA. For the case of achieving median and worst results SGA with SSRT achieved better results in 81.82% and 72.73% of problems respectively.

Table 6.6 shows the results for NSGA-II with and without SSRT. In achieving best results SSRT improves the performance of NSGA-II in 71.43% of problems, and in 21.43% of problems the performance remains the same. It is worth mentioning in problem g03 it has improved the best results by 5.19%, and by 31.62% in g13. It has improved the mean results in 78.57% of problems. For 64.29% of problems the median results, and in 42.86% of problems the worst results, are improved by SSRT.

Table 6.5: Performance of SGA with and without SSRT from 30 independent runs.

Fn	SGA	Best	Median	Mean	StDev	Worst
g01	NST	-14.998	-13.815	-13.990	9.41E-01	-11.780
	ST	-14.995	**-14.986**	**-14.753**	**4.76E-01**	**-13.811**
g03	NST	×	×	×	×	×
	ST	×	×	×	×	×
g05	NST	×	×	×	×	×
	ST	5277.204**	×	×	×	×
g06	NST	-6945.396	-6920.632	-6920.196	1.61E+01	-6888.569
	ST	**-6958.979**	**-6925.673**	**-6929.641**	**1.11E+01**	**-6916.950**
g07	NST	25.615	27.755	28.310	2.41E+00	36.594
	ST	**24.912**	**26.184**	**26.255**	**8.98E-01**	**27.799**
g08	NST	-0.095825	-0.095825	-0.095825	5.24E-09*	-0.095825
	ST	-0.095825	-0.095825	-0.095825	**7.76E-11***	-0.095825
g09	NST	680.808	681.648	681.821	6.50E-01	683.944
	ST	680.826	**681.375**	**681.322**	**2.70E-01**	**681.702**
g10	NST	7166.255	7823.128	8376.182	1.45E+03	13284.257
	ST	**7126.020**	**7568.463**	**7591.785**	**2.62E+02**	**7988.912**
g11	NST	×	×	×	×	×
	ST	×	×	×	×	×
g13	NST	0.457442	0.922264	1.031979	6.59E-01	3.854752
	ST	**0.078772**	**0.781332**	**0.689290**	**2.66E-01**	**0.970765**
B01	NST	-44.775	-41.773	-41.553	1.81E+00	-37.910
	ST	-42.576	-40.870	-40.536	**1.44E+00**	-37.794
B02	NST	964.124	975.253	973.961	3.76E+00	978.575
	ST	**961.240**	**973.966**	**972.857**	**5.06E+00**	979.036
B03	NST	-1.905	-1.903	-1.903	2.98E-03	-1.890
	ST	-1.905	**-1.904**	**-1.904**	**4.70E-04**	**-1.904**
B04	NST	-0.848895	-0.645441	-0.678532	8.76E-02	-0.575312
	ST	**-0.859386**	**-0.793935**	**-0.738429**	1.18E-01	**-0.528032**

NST= Algorithm without SSRT, ST = Algorithm with SSRT, ×'= feasible solution were not found, **Bold** font indicates the best result achieved by algorithm with SSRT. , * indicates though the best, worst, median, and mean results are the same, standard deviation is positive due to rounding error; ** indicates algorithm achieved feasible solutions in only 1 run.

Table 6.6: Performance of NSGA-II with SSRT and without SSRT from 30 independent runs.

Fn	NSGA-II	Best	Median	Mean	StDev	Worst
g01	NST	-15.000	-14.998	-14.646	8.07E-01	-12.429
	ST	-15.000	**-15.000**	**-14.999**	**7.19E-04**	**-14.997**
g03	NST	-0.616	-0.106	-0.171	1.66E-01	-0.003
	ST	**-0.648**	**-0.162**	**-0.194**	**1.50E-01**	**-0.009**
g05	NST	×	×	×	×	×
	ST	×	×	×	×	×
g06	NST	-6950.884	-6940.375	-6941.099	4.14E+00	-6935.730
	ST	**-6960.390**	**-6942.121**	**-6942.705**	5.47E+00	-6931.173
g07	NST	24.438	25.299	25.462	8.72E-01	27.901
	ST	**24.363**	**25.098**	**25.244**	**8.49E-01**	28.330
g08	NST	-0.095825	-0.095825	-0.095825	4.23E-17*	-0.095825
	ST	-0.095825	-0.095825	-0.095825	9.80E-17*	-0.095825
g09	NST	680.746	681.178	681.232	3.88E-01	682.302
	ST	**680.644**	**680.918**	**681.008**	**3.36E-01**	682.302
g10	NST	7128.741	7940.550	8252.011	1.12E+03	11953.460
	ST	**7077.327**	**7443.979**	**7835.171**	1.27E+03	16105.790
g11	NST	0.754	0.867	0.878	9.05E-02	0.999
	ST	**0.751**	**0.870**	**0.867**	**8.90E-02**	**0.996**
g13	NST	0.974036	0.994066	0.991300	8.59E-03	0.999997
	ST	**0.666042**	0.997573	**0.914083**	1.15E-01	**0.998898**
B01	NST	-46.874	-41.384	-41.831	1.64E+00	-39.527
	ST	**-46.894**	**-42.137**	**-42.079**	2.04E+00	-39.189
B02	NST	963.764	973.842	973.671	4.16E+00	978.789
	ST	**963.698**	974.373	974.351	**3.54E+00**	978.843
B03	NST	-1.905	-1.904	-1.903	2.07E-03	-1.896
	ST	-1.905	-1.904	**-1.904**	**7.95E-04**	**-1.902**
B04	NST	-0.864224	-0.666732	-0.731112	1.02E-01	-0.574267
	ST	**-0.865643**	**-0.770036**	**-0.761072**	**9.89E-02**	**-0.625385**

NST= Algorithm without SSRT, ST = Algorithm with SSRT, ×'= feasible solution were not found, **Bold** font indicates the best result achieved by algorithm with SSRT. , * indicates though the best, worst, median, and mean results are the same, standard deviation is positive due to rounding error.

Table 6.7: Performance of MCA with and without SSRT from 30 independent runs.

Fn	MCA	Best	Median	Mean	StDev	Worst
g01	NST	-15.148	-13.200	-13.656	1.48E+00	-10.300
	ST	**-15.149**	-13.200	**-13.639**	1.52E+00	-9.752
g03	NST*	-1.000	-1.000	-1.000	1.90E-04	-1.000
	ST*	-1.000	-1.000	-1.000	1.46E-03	-1.000
g05	NST	5131.295	5364.948	5450.763	3.27E+02	6111.453
	ST	**5126.582**	5429.567	5470.412	**2.89E+02**	**6058.473**
g06	NST	×	×	×	×	×
	ST	×	×	×	×	×
g07	NST	24.310	25.784	26.802	2.41E+00	34.997
	ST	24.319	**25.609**	**26.720**	2.76E+00	34.997
g08	NST*	-0.095825	-0.095825	-0.095825	4.23E-17*	-0.095825
	ST*	-0.095825	-0.095825	-0.095825	4.23E-17*	-0.095825
g09	NST	680.776	681.380	681.454	4.23E-01	682.616
	ST	680.776	**681.311**	**681.383**	**3.76E-01**	**682.357**
g10	NST**	7073.306	7656.054	7623.999	4.29E+02	8281.761
	ST	**7050.513**	**7510.483**	**7594.795**	**4.04E+02**	**8153.756**
g11	NST	0.750	0.750	0.750	8.00E-05*	0.750
	ST	0.750	0.750	0.750	1.43E-04*	0.750
g13	NST	0.137313	0.416643	0.455166	1.69E-01	0.999374
	ST	**0.129549**	0.421717	**0.422981**	2.00E-01	**0.998991**
B01	NST	-47.764	-47.764	-47.764	0.00E+00	-47.764
	ST	-47.764	-47.764	-47.764	0.00E+00	-47.764
B02	NST	961.870	964.849	965.128	2.35E+00	968.546
	ST	**961.863**	**962.963**	**964.330**	2.65E+00	**967.907**
B03	NST	×	×	×	×	×
	ST	×	×	×	×	×
B04	NST	×	×	×	×	×
	ST	×	×	×	×	×

NST= Algorithm without SSRT, ST = Algorithm with SSRT, ×'= feasible solution were not found, **Bold** font indicates the best result achieved by algorithm with SSRT. , * indicates though the best, worst, median, and mean results are the same, standard deviation is positive due to rounding error. ** indicates algorithm achieved feasible solutions in only 10 run.

Form Table 6.7 we can see SSRT has improved the performance of MCA as well. However, with both approaches MCA could not find any feasible solutions for problem g06, B03, and B04. For the other 11 problems, the best results improve with SSRT in 45.45% of problems and remain the same in 45.45% of problems. The mean results are improved in 54.55% of problems and remain the same in 36.36% of problems. In 36.36% of problems the median results, and in 45.45% of problems the worst results, are improved with the help of SSRT. We should mention that in problem g10, MCA achieved feasible results in only 10 runs from the 30 runs. However with the help of SSRT it achieves feasible results in every run.

TC using SSRT achieved the same result for most of the problems as shown in Table 6.8. However, in g03, g05, g10, and B04, the results are mostly improved. This algorithm uses triangular crossover (Elfeky *et al.*, 2006) which chooses parents from both feasible and infeasible to generate offspring close to the boundary of the feasible region. Since the aim of this crossover is very similar to SSRT, that is to bring the population inside the feasible region, after the genetic process the results remain the same.

In Table 6.9 the comparison of the results (best, median, mean, st.dev and worst) achieved by the algorithms using SSRT and without SSRT are shown in the form of xxxxx = [Best, Median, Mean, St.Dev, Worst, results]. Symbol '+' indicates the algorithm using SSRT achieved a better result than without SSRT; '–' indicates algorithm achieved a worse result using SSRT; and '0' indicates same results.

Table 6.8: Performance of TC with and without SSRT from 30 independent runs

Fn	TC	Best	Median	Mean	StDev	Worst
g01	NST	-15.000	-15.000	-15.000	0.00E+00	-15.000
	ST	-15.000	-15.000	-15.000	0.00E+00	-15.000
g03	NST	-0.963	-0.886	-0.891	4.40E-02	-0.808
	ST	**-0.997**	**-0.991**	**-0.977**	**2.55E-02**	**-0.880**
g05	NST	5126.581	5215.653	5318.166	2.35E+02	5898.594
	ST	**5126.505**	5215.762	**5317.374**	2.37E+02	5954.586
g06	NST	-6961.814	-6961.814	-6961.814	3.70E-12*	-6961.814
	ST	-6961.814	-6961.814	-6961.814	3.70E-12*	-6961.814
g07	NST	24.566	25.832	25.855	4.80E-01	26.825
	ST	25.130	25.890	25.941	4.87E-01	26.944
g08	NST	-0.095825	-0.095825	-0.095825	4.23E-17*	-0.095825
	ST	-0.095825	-0.095825	-0.095825	4.23E-17*	-0.095825
g09	NST	680.634	680.654	680.660	1.98E-02	680.703
	ST	680.634	680.661	680.663	2.11E-02	680.703
g10	NST	7098.481	7862.190	8058.985	8.21E+02	10509.988
	ST	**7071.489**	7979.785	8097.473	8.47E+02	**9603.307**
g11	NST	0.750	0.750	0.750	0.00E+00	0.750
	ST	0.750	0.750	0.750	0.00E+00	0.750
g13	NST	0.055147	0.597604	0.481420	3.32E-01	1.000000
	ST	0.055147	0.597604	0.481420	3.32E-01	1.000000
B01	NST	-47.193	-46.199	-46.220	5.87E-01	-44.844
	ST	-47.193	-46.199	-46.220	5.87E-01	-44.844
B02	NST	961.724	962.661	963.293	1.86E+00	968.000
	ST	961.724	962.661	963.293	1.86E+00	968.000
B03	NST*	-1.905	-1.905	-1.905	3.29E-04	-1.903
	ST*	-1.905	-1.905	-1.905	3.66E-04	-1.903
B04	NST	-0.865939	-0.672910	-0.741236	1.05E-01	-0.500000
	ST	**-0.866011**	**-0.859673**	**-0.785920**	1.05E-01	-0.500000

NST= Algorithm without SSRT, ST = Algorithm with SSRT, ×'= feasible solution were not found, **Bold** font indicates the best result achieved by algorithm with SSRT. , * indicates though the best, worst, median, and mean results are the same, standard deviation is positive due to rounding error.

Table 6.9: Performance of SSRT on EAs in solving the benchmark problems

Fn	AMA	SGA	NSGA-II	MCA	TC
g01	0 0 + + +	− + + + +	0 + + + +	+ 0 + − −	0 0 0 0 0
g03	0 0 0 + 0	×	+ + + + +	0 0 0 − 0	+ + + + +
g05	− − + + +	+*	×	+ − − ++	+ − + − −
g06	0 + + + +	+ + + + +	+ + + − −	×	0 0 0 0 0
g07	+ + + + +	+ + + + +	+ + + + −	− + + − 0	− − − − −
g08	0 0 0 + 0	0 0 0 + 0	0 0 0 − 0	0 0 0 0 0	0 0 0 0 0
g09	+ + + + +	− + + + +	+ + + + 0	0 + + + +	0 − − − 0
g10	+ + + + +	+ + + + +	+ + + − −	+ + + + +	+ − − − +
g11	0 0 0 + 0	×	+ + + + +	0 0 0 − 0	0 0 0 0 0
g13	+ + + + +	+ + + + +	+ − + − +	+ − + − +	0 0 0 0 0
B01	+ − + + +	− − − + −	+ + + − −	0 0 0 0 0	0 0 0 0 0
B02	− + + + +	+ + + + −	+ − − + −	+ + + − +	0 0 0 0 0
B03	0 + + + +	0 + + + +	0 0 + + +	×	0 0 0 0 0
B04	− + + + +	+ + + − +	+ + + − +	×	+ + + 0 0

xxxxx = [Best, Median , Mean, St.Dev, Worst, results]. Symbols: +, better; 0, similar; −, worse, ×'= feasible solution were not found, * with SSRT, SGA found 1 feasible solution.

After analyzing the results, we can see for most of these benchmark problems SSRT has enhanced the performance of the evolutionary algorithms.

6.3.3 Solving a Real World Problem

The test problems considered in the previous section are smaller in size. Although the contribution of SSRT is positive in those problems, the real improvements may seem very little. To test the true performance of SSRT in solving a reasonable size problem, we use a real world crop planning problem (Sarker and Quaddus, 2002; Sarker and Ray, 2005; Sarker and Ray, 2009) in this section. As our objective is to test the performance of SSRT, we ignore the description of the problem here. However, the interested readers can find the details in (Sarker and Ray, 2005; Sarker and Ray, 2009) and in Appendix C. Here a

constrained non-linear single objective model of the crop planning problem is solved. The original model consists of 68 variables and 45 constraints. By applying variable / constraint reduction technique, the model can be reduced to 39 variables and 15 constraints. A given instance of the problem is considered, where the ratio of the feasible region and search space ρ is almost 0.00%.

As with the benchmark problems, two sets of experiments have been made for each of the algorithms. The results on the improvements of the 30 independent runs of the algorithms are summarized in Table 6.10. Using SSRT, most of the algorithms achieve remarkably better fitness value, for example, 1.24% better fitness value by AMA, 7.61% by SGA, 0.25% by NSGA-II, 1.38% by MCA than without SSRT. If we consider in terms of profit gain, AMA gains \$0.28 million, SGA gains \$0.63 million when NSGA-II and MCA gain \$0.06 and \$ 0.33 million respectively. SSRT also provides better mean, standard deviation, worst, and median values with these algorithms. The results of TC is not mentioned here in Table 6.10, since its result is not improved, like most other test problems, for the reason discussed earlier.

Table 6.10: Improvement of performance of different EAs using SSRT in solving crop problem.

Algorithms	**Best**	**Median**	**Mean**	**Worst**
AMA	1.24%	3.32%	1.64%	3.65%
SGA	7.61%	42.41%	23.01%	0.58%
NSGA-II	0.25%	0.68%	0.75%	6.51%
MCA	1.38%	4.31%	4.19%	3.89%

The convergence curve on this problem using AMA and AMA with SSRT is given in Figure 6.2. It is clear that AMA with SSRT provides faster convergence towards better results than AMA without SSRT.

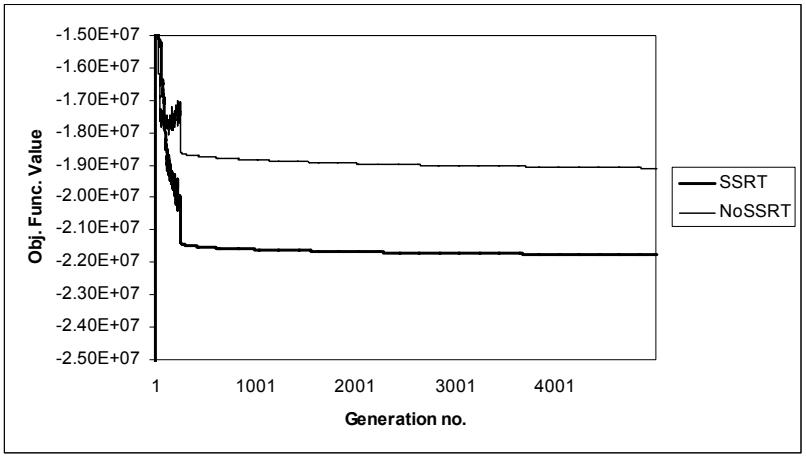


Figure 6.2: Convergence Curve for Crop problem using AMA with and without SSRT.

6.4 Chapter Summary

This chapter presents a simple search space reduction technique (SSRT) for population-based evolutionary algorithms to solve constrained optimization problems with tiny feasible region. The proposed SSRT allows certain infeasible solutions in the initial population to move slowly towards the feasible region. The performance of SSRT is investigated by solving a set of test problems and a real world case problem with AMA, simple genetic algorithm and three well-known algorithms found in the literature. This approach usually improves the performance of the algorithms in terms of either solution quality or computational time or both, at the cost of an additional step with $O(M^2)$ complexity. From the results of the real world problem, it is evident that the method is more appreciable for large scale problems with tiny feasible space. Although the idea of SSRT is very simple, the results justify the use of SSRT with evolutionary algorithms.

Regardless of the other limitations of the constraints it has been noticed that the existence of equality constraints reduces the size of the feasible space. In all the problems involving equality constraints in the test set the ratio of feasible space over the search space is almost zero. In the next chapter, a method is proposed to deal with equality constraints.

Chapter 7

Handling Equality Constraints

In addition to inequality constraints, many mathematical models require equality constraints to represent the practical problems appropriately. The existence of equality constraints reduces the size of the feasible space significantly, which makes it difficult to locate feasible and optimal solutions. This chapter presents a new Equality Constraint Handling Technique (ECHT) which enhances the performance of AMA in solving constrained optimization problems with equality constraints. The technique is basically used as an agent learning process in AMA. The performance of AMA with ECHT is tested on a set of well-known benchmark problems. The experimental results confirm the improved performance of the proposed technique.

7.1 Introduction

Many mathematical optimization models involve a set of equality, inequality or both types of constraints. The size of feasible space of these problems depends on the type of constraints and their interactions. In any optimization problems with equality constraints, each feasible solution point must lie on each and every equality constraint. The existence of equality constraints reduces the size of the feasible space drastically. In the previous chapter, we have seen in all the problems with equality constraints that the ratio of feasible space over the search space is almost zero. It is not easy to find the feasible points while solving such equality constrained problems. As a consequence, EAs have inherent difficulty in dealing with equality constraints when solving

constrained problems. As in Mezura-Montes and Coello (2002), it is very hard for traditional EAs to find feasible and optimal solutions for such problems.

Many traditional EAs (Deb, 2000; Elfeky *et al.*, 2006) convert equality constraints $h_j(\boldsymbol{X}) = 0$ into inequality constraints $-\delta \leq h_j(\boldsymbol{X}) \leq \delta$ (where δ is a small tolerance value) to increase the feasible space temporarily. Still they may fail to achieve either feasible or good quality solutions, for example, Koziel and Michalewicz (1999) have not found good quality solutions for problem g05 which involves equality constraints.

In solving problem g13 (which involves equality constraints), the mean and worst results achieved by the stochastic ranking algorithm (Runarsson and Yao, 2000) were 25.20% and 302.07% from the optimum respectively. Runarsson and Yao (2000) also reported that they failed to solve some equality constrained problems such as g03 and g05 using the dynamic penalty method as in Joines and Houck (1994). Although Deb (2000) has solved one optimization problem with equality constraints optimally, the percentage variation of median from the optimal was 346.29%. Mezura-Montes and Coello (2002) reported a comparison of several well-known multi-objective-based techniques to handle constraints such as Constrained Optimization by Multi-Objective Genetic Algorithms (COMOGA) by Surry and Radcliffe (1997), Vector Evaluated Genetic Algorithms (VEGA) (Schaffer, 1985) used by Coello (2000b), Multi-Objective Genetic Algorithms (MOGA) (Fonseca and Fleming, 1993) applied by Coello (2000a), Niched-Pareto Genetic Algorithms (NPGA) (Horn *et al.*, 1994) implemented by Coello and Mezura-Montes (2002). Most of these algorithms successfully solved the problems with inequality constraints. However, for 75% of the problems with equality constraints, the algorithms could not achieve optimal. Interestingly, they were unable to find any feasible solutions for 50% of the problems. This demonstrates the difficulties in solving constrained optimization problems with equality constraints, which motivates to design a new equality constraint handling technique.

In this chapter, a new technique is presented to handle the equality constraints. In any optimization problems with equality constraints, to satisfy the condition of feasibility and optimality, the solution points must lie on each and every equality constraint. That means it might be possible to find an optimal solution by simply

exploring an equality constraint function where the constraint function contains all the variables. This common knowledge encourages to design a new ECHT for dealing with equality constraints. The basic idea is to reach a point on the equality constraint from the current position of an individual solution, and then explore on the constraint landscape. That means an individual would explore only a portion of the entire search space. In practical problems, an equality constraint may not contain all the variables which require exploring the landscapes of other equality constraints. To the best of our knowledge, this approach for handling equality constraints has not appeared in the literature.

The proposed ECHT is added as a new LSLP in AMA. The new LSLP is particularly valuable during the early stage of the algorithm, to speed the search towards the feasible space. For that reason it is used exclusively for the first N generations, and thereafter other LSLP's are used to refine the solution. Chapter 6 was all about speeding the search towards the feasible space by calculating an approximate centroid from better initial randomly generated solutions. A percentage of worse solutions are then allowed to follow the centroid. Here the new LSLP tries to bring any individual solution to an equality constraint or to explore on that.

The new version of AMA is capable of dealing with the equality constraints more efficiently. To test the performance of the algorithm, a set of ten benchmark problems with equality constraints is selected, and the results are compared with different algorithms. The comparisons show that the results are of improved quality with low computational time. This chapter also analyzes the effect of the new learning process for handling the equality constraints.

The rest of this chapter is organized as follows. Section 7.2 describes the proposed equality constraint handling technique. Section 7.3 presents the extended Agent-based memetic algorithm. Section 7.4 provides computational experience, results of the proposed approach on benchmark problems. Another approach of incorporating ECHT in AMA and its performance is discussed in section 7.5. Finally, Section 7.6 concludes the chapter and provides future research directions.

7.2 Equality Constraint Handling Technique (ECHT)

The size of the feasible space within the search space may depend on the type of the constraints involved in the problem. For example, the existence of equality constraints significantly reduces the size of the feasible space, which makes it difficult for the algorithms to find the feasible points.

Consider a simple numerical example as follows.

Maximize $f(\boldsymbol{X}) = x_1 + x_2$;

Subject to

$1 \le x_1 \le 2$;

$1 \le x_2 \le 2$;

For simplicity, assume that up to two decimal points are allowed for x_1 and x_2 (i.e. their values can be 1.00, 1.01, 1.02 etc). If there are no other constraints, there are 10201 solution points in the discrete search space, and the optimal solution is 4.00 (when $x_1 = 2.00$, $x_2 = 2.00$). If we add an inequality constraint $g(\boldsymbol{X}) = x_1 - x_2 \le 0.80$, 9991 points (97.94%) in the discrete search space still satisfy the constraint, and the optimal result remains the same 4.00 (when $x_1 = 2.00$, $x_2 = 2.00$). However if we replace the inequality constraint $g(\boldsymbol{X})$ with an equality constraint $h(\boldsymbol{X}) = x_1 - x_2 = 0.80$, the feasible space becomes very tiny in comparison to the search space. Now all the feasible points must lie on the equality constraint $h(\boldsymbol{X})$. If we consider up to two decimal points, only 21 points satisfy the constraints among 10201 possible candidates and the optimal objective is now 3.20 (with $x_1 = 2.00$, $x_2 = 1.20$). The feasible space is now reduced to 0.21% of the original search space which is very hard to locate.

The complexity can be greatly increased by the number of constraints and the interference among them. However the existence of the equality constraints can be useful for solving COPs, as it might be possible to find an optimal solution by simply exploring the equality constraint functions. In this chapter, a new search process for handling equality constraints is introduced. To explain the technique, let us consider an

optimization model with all or a subset of constraints of equality type. Three propositions related to equality constraints are presented next.

Proposition 1: For a given constrained optimization model, where all or a subset of the constraints are of equality type, a feasible solution cannot be found without satisfying any of the equality constraints. We assume that the solution point under consideration satisfies all inequality constraints (if any).

Proof: By definition, a feasible solution point must satisfy all the constraints. To satisfy an equality constraint, the point must be on that constraint.

Proposition 2: A feasible and optimal solution point must lie on each and every equality constraint.

Proof: To satisfy all the equality constraints, a feasible point must lie on all the equality constraints. By definition, the best feasible point is the optimal solution.

Proposition 3: It is possible to find a feasible and optimal solution point by simply searching on an equality constraint function landscape when the function contains all the variables and is continuous.

Proof: As the feasible and optimal point must lie on all equality constraints, by simply moving on a constraint (i.e. points which satisfy the constraint), which involves all the variables, one may be able to reach the optimal solution.

As all variables may not exist in a certain equality constraint and there is no guarantee of having a continuous function, finding a better solution by simply exploring on the equality constraint may not work. The above arguments are used to design a new technique for handling equality constraints in solving optimization problems, as follows:

Choose an equality constraint randomly, and a randomly selected individual.

1. If the equality constraint is not satisfied in this individual, change only one variable so that the constraint is satisfied.

2. If the individual satisfies the constraint, choose two variables and modify them in such a way that the resulting point still satisfies the constraint and the total constraint violation reduces.

The first move would help to reduce overall constraint violation, and the second move would help to increase diversity of individuals in the population. The second move also considers the total constraint violations that include all the equality and inequality constraints.

It is not always possible to satisfy the constraints by changing a single variable. The variable may violate its bound. For simplicity of the algorithms in that case the variable is restricted to the boundary. Sometimes the required value of the variable might be an imaginary number (because of nonlinearity), in that case a random number is assigned between its bounds. The method of calculating change of fitness value is somehow similar to the Generalized Reduced Gradient (GRG) algorithm (Lasdon *et al.*, 1978). However, it has been done here numerically.

It is not always easy to move on an equality constraint function landscape. To explore on the constraint landscape it may be needed to change several variables involved in that constraint. We may need to increase the value of some variables while decreasing the value of some other variables in order to remain on the same equality constraint but at a different point. This is very simple when the constraint involves one or two variables, but becomes complex with more variables. To reduce this complexity in that case, only two variables are randomly selected, that are involved in the selected equality constraint and then a small random change is made on the first variable. The second variable is then modified in such a way that the solution remains on the constraint surface.

Pseudo code: Equality Constraint Handling Technique.

Let $\boldsymbol{X}=[x_1,x_2,\ldots,x_n]$ be the current solution vector, CV_j = constraint violation for $h_j(\boldsymbol{X})=0$; $j=1,2,\ldots,q$ with solution vector $\boldsymbol{X}$ and $\boldsymbol{X}_j$ presents a vector that contains the variables involved in constraint $h_j(\boldsymbol{X})=0$. *rnd*(.,.) is a uniform random number generator, *TCV*=Total Constraint Violation.

Select any of the equality constraints $h_j(\boldsymbol{X})=0;\ j=1,2,...,q$ randomly.

If $(CV_j \neq 0)$

Select a random variable $\{x_a | x_a \in \boldsymbol{X}_j\}$;

Calculate x'_a so that $CV_j=0$;

If x'_a is an imaginary number, set $x'_b = rnd\ (\underline{x_a}, \overline{x_a})$;

If $(x'_a > \overline{x_a})$ set $x'_a = \overline{x_a}$;

If $(x'_a < \underline{x_a})$ set $x'_a = \underline{x_a}$;

Else

Select a random variable $\{x_a | x_a \in \boldsymbol{X}_j\}$;

Calculate $\varphi = rnd\ (-0.1,0.1) \times x_a$;

Set $x'_a = x_a \pm \varphi$;(add/subtract, based on which direction the *TCV* is reduced);

Select a random variable $\{x_b | x_b \in \boldsymbol{X}_j, a \neq b\}$.Calculate x'_b such that $C_1 x'_a \oplus C_2 x'_b = C_1 x_a \oplus C_2 x_b$, where $\oplus$ represents any mathematical operators (e.g. $+, -, \times, /$), C_1, C_2 are constants.

End.

Figure 7.1 and 7.2 show an example of how the new ECHT works. Consider a nonlinear optimization problem, consisting of two equality constraints, which can be defined as follows:

Minimize $f(\boldsymbol{X})$, where $\boldsymbol{X} = [x_1, x_2]$, $\boldsymbol{X} \in R^n$ is a set of 2 variables of the solution.

Subject to,

$h_1(\boldsymbol{X}) = 0;$

$h_2(\boldsymbol{X}) = 0;$

$\boldsymbol{X} \geq 0;$

For an example of the first type of move, assume the graphical representation of the problem is like Figure 7.1. The two equality constraints $h_1(X) = 0$ and $h_2(X) = 0$ intersect at two points. Since the two intersection points of the constraints satisfy both of the constraints, we have only two feasible solution points for this problem. As it is a minimization problem, the objective function value $f(\boldsymbol{X})$ is optimal on the lower intersection point. The new ECHT randomly selects an equality constraint, suppose here $h_1(\boldsymbol{X}) = 0$, for an individual. If the solution (the black dot in Figure 7.1) does not satisfy the constraint (i.e. it does not lie on the arc satisfying the equality constraint), then select a random variable involved in that constraint, e.g. x_2. Then x_2 shall be changed so that $h_1(\boldsymbol{X}) = 0$.

Figure 7.2 shows the second type of move. If the constraint is satisfied, the ECHT will choose two variables x_1 and x_2 and modify them in a way that the resulting point is still on the constraint. This helps to increase the diversity and move towards the optimal solution.

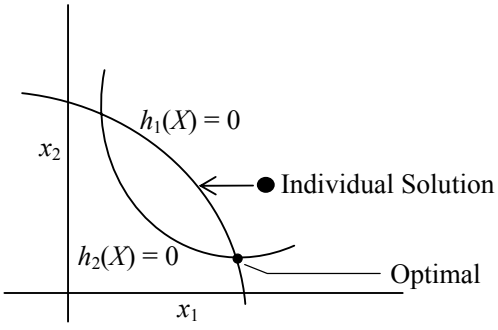


Figure 7.1: ECHT (when the solution does not satisfy the equality constraint).

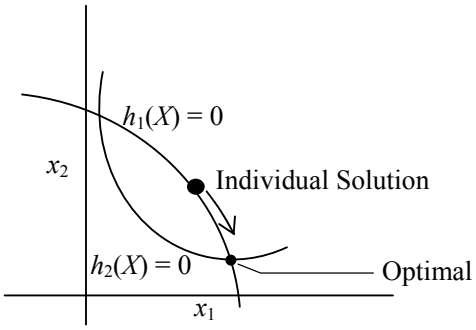


Figure 7.2: ECHT (when the solution satisfies the equality constraint).

7.3 Extended AMA (AMA-II)

In chapter 4, AMA is presented, where the agent concept is incorporated with memetic algorithms. The goal of each agent is to improve its fitness while satisfying constraints. Following the natural adaptation process, in the proposed AMA the agents improve their fitness by adaptively selecting a life span learning process from the designed set, together with the evolutionary adaptation of the population.

In this chapter, to enhance the performance of AMA in solving equality constrained problems, ECHT introduced in the previous section is used as a new life span learning process (LSLP). Through this LSLP an agent moves from its current position towards the curvature of an equality constraint by changing one or two variables. In AMA, after performing the crossover, a certain percentage of the agents are selected for LSLP. Here in extended AMA (indicated as AMA-II), the agents select only the new LSLP in the early stage of the evolution process e.g. first N generations. This LSLP directs the agents towards the feasible space, which speeds up the search process. Then for the later generations, the agents apply the other four LSLPs (described in chapter 4) self-adaptively for refining the solutions.

The main steps of AMA-II are given below.

Step 1. Create a random population, which consists of $M \times M$ agents.

Step 2. Arrange the agents in a lattice-like environment.

Step 3. Evaluate the agents individually. If the stopping criterion has been met, go to step 7, otherwise go to step 4.

Step 4. For each agent examine its neighborhood. Select an agent from its neighborhood and perform crossover.

Step 5. Select a certain percentage of agents; During the initial generations apply only the LSLP designed for equality constraint, then switch to other LSLPs.

Step 6. Go to step 3.

Step 7. Stop.

In the algorithm, the equality constraints are given special preference using the proposed learning process of the agents. At the same time, the constraints are also handled indirectly while comparing the individuals. Details of this constraint handling have been already discussed in section 4.5.

7.4 Experimental Studies

In this section, first the benchmark problems are presented, then the initial design experience for the new LSLP with the proposed ECHT is discussed. The performance of AMA with the new LSLP in solving COPs with equality constraints is analyzed next. A set of experiments is also carried to investigate the effect of the LSLP to the performance of the algorithm.

7.4.1 Benchmark Problems

To test the performance of AMA with the new LSLP, ten benchmark problems that involve equality constraints have been selected. The first four problems (g03, g05, g11, g13) are taken from 13 well known benchmark problems (g-series) (Runarsson and Yao, 2000). Mezura-Montes and Coello (2002) categorized some test problems based on difficulty and reported problems g05 and g13 as "Very difficult" and g03 as "Difficult". Two problems (B01 and B02) used in the previous chapters are also included, as they involve equality constraints. The other four problems (indicated here as B06-B09) are taken from (Floudas, 1999; Himmelblau, 1972; Hock and Schittkowski, 1981).

The benchmark problems involve different forms of objective functions and different number of variables (n). The maximization problems are transformed into equivalent minimization problems. The characteristics of the test problems are given in Table 7.1, and the detailed mathematical representations are provided in Appendix A and B.

Table 7.1: Characteristics of the test problems.

Fn	(n)	Obj. Fuc.	LI	NI	LE	NE	AC	Optimal
g03	10	Polynomial	0	0	0	1	1	-1.000
g05	4	Cubic	2	0	0	3	3	5126.498
g11	2	Quadratic	0	0	0	1	1	0.750
g13	5	Nonlinear	0	0	0	3	3	0.053950
B01	10	Nonlinear	0	0	3	0	3	-47.765
B02	3	Quadratic	0	0	1	1	2	961.715
B06	7	Linear	0	1	0	5	6	193.724
B07	5	Quadratic	0	0	2	0	1	0.000
B08	6	Nonlinear	0	0	6	0	6	6.334
B09	4	Linear	0	0	0	2	2	-1.000

LI = Linear Inequalities, NI = Nonlinear Inequalities, LE = Linear Equalities, NE = Nonlinear Equalities, AC = Active Constraints.

7.4.2 Initial Design Experience

During the initial design of the LSLP for equality constraints I have experienced some interesting situations. I would like to share some of them with the readers.

According to proposition 2, a feasible and optimal solution point must lie on each and every equality constraint. So if we explore any of the equality constraints it is possible to reach the optimal solution. In the initial design of the new LSLP for equality constraints, only a single equality constraint function is examined, by ignoring all other functions, for finding a good quality solution. However, this approach was not consistently providing benefit, as a single equality constraint may not contain all the variables involved in the model.

In the second attempt, one of the equality constraints is selected randomly and then one variable is changed by assigning it a value to reach the constraint. For example let us consider problem g13 constraint $h_2(x) = x_2 x_3 - 5x_4 x_5 = 0$.

According to the first move of the new ECHT, the LSLP needs to select a random variable (e.g. x_2). The required value of x_2 to reach to the equality constraint is $x_2 = \frac{5x_4 x_5}{x_3}$. If the target constraint cannot be reached, due to the variable bounds, the variable is assigned its upper/lower bound value (whichever is closest to the calculated value). This works fine for most of the problems. However in some test problems it raises a different issue e.g. in problem B01 if any x_i=0 (lower bound) then objective function value $f(x) = \sum_{i=1}^{10} x_i \left(c_i + \ln \frac{x_i}{\sum_{j=1}^{10} x_j} \right)$ will be infinity. To avoid this, in this case a very low value (e.g. x_i=1E-20) is assigned to x_i.

Sometimes the required value of the variable might be an imaginary number, e.g. in problem g11 where $h(x) = x_2 + x_1^2 = 0$, for any negative value of x_2, the required value of $x_1 = \sqrt{-x_2}$ must be an imaginary number. In that case a random number between its boundary ranges is assigned.

Furthermore when trying to explore on an equality constraint the values of only two variables are changed, to keep it computationally simple. For example in $h_2(x) = x_2 x_3 - 5x_4 x_5 = 0$, suppose we select randomly the variables x_2 and x_5. Then we change the value of x_2 as $x_2' = x_2 + rnd(-0.1,0.1) \times x_2$ and we calculate the new value of x_5' as $x_2 x_3 - 5x_4 x_5 = x_2' x_3 - 5x_4 x_5'$. However this process was not simple after all, as the variable may contain any type of nonlinearity. In particular, a variable with a power of 2 could have negative or positive value and have the same impact on satisfying the constraint. For example in problem g03 for any value x_i (positive or negative value) the result is same as $h_1(x) = \sum_{i=1}^{n} x_i^2 - 1 = 0$. If we always assign a positive value, or always a negative value, we could find it impossible to find the global optimum. The initial LSLP is changed so that in cases of variables with even powers, it is assigned randomly either a positive or negative value. This process consistently provided better solutions.

In third attempt, the new LSLP is used for only the first 100 generations (arbitrarily chosen) and the other four LSLPs afterward. The reason for this choice was due to the fact that the new LSLP increases the diversity of the population. It is observed that this process provided even better results.

7.4.3 Experimental Results and Discussions

First the performances of three algorithms in solving the test problems are investigated:

- SGA: a simple genetic algorithm with tournament selection, SBX crossover and parameter based mutation operator (Deb, 2000) as described in chapter 3.
- AMA-I: AMA as described in chapter 4.
- AMA-II: AMA with the new LSLP for equality constraints.

The initial solution vectors for the agents were randomly generated within the boundary of each decision variable. For AMA-I and AMA-II the agents are arranged in a lattice-like environment of size $M \times M$, so the number of the agents (i.e. the population

size) must be a square number. In AMA-II the new LSLP is used only in the first 100 generations then the other four LSLPs are used. For each algorithm the same population size (100), and probability of learning/LSLP (P_L= 0.2) are used. For SGA the probability of crossover is P_C = 0.90 and the probability of mutation used is P_M=0.2. The maximum number of generations considered was 3500.

The best, median, mean, standard deviation (st.dev.), and worst results, as well as execution time, for the well-known problems (g03, g05, g11, and g13) from 30 independent runs are given in Table 7.2. The results for remaining problems (B01, B02 B06-B09) are given in Table 7.3. An '×' in the Tables indicates that the algorithm did not find any feasible solution.

Form Tables 7.2 and 7.3 we can see that SGA could not find any feasible solution for problems g03, g05, and g11. Although SGA is able to find feasible solutions for the other problems, the best solution obtained is far from the known best. Both AMA-I and AMA-II have solved those problems successfully. AMA-I achieves the optimum in four problems (g03, g11, B07, and B09). AMA-II achieves the optimum in six problems (g03, g11, g13, B02, B07, and B09) and very close to optimum in the other four problems. For example, the achieved best results of AMA-II in g05 and B01 are within 0.00002% and 0.15912% of the optimum respectively. B06 is a special type of problem which involves a set of equality and inequality constraints with several variables, and the objective function depends on only one variable which is involved in the inequality constraint. Though the algorithm could not achieve the optimum for this problem the achieved result is within 3.97266% of optimum. The algorithm could not find the optimum (only 3.91538% far from optimum) for B08 which involves a set of linear equality problems.

Table 7.2: Statistics for 30 independent runs of different algorithms for problems with equality constraints in g-Series.

Fn	Optimal	App.	Best	Median	Mean	St.Dev	Worst	Time(s)
		SGA	×	×	×	×	×	4.14
g03	-1.000	AMA-I	-1.000	-1.000	-1.000	2.76E-06*	-1.000	80.86
		AMA-II	-1.000	-1.000	-1.000	**1.70E-06***	-1.000	86.54
		SGA	×	×	×	×	×	3.52
g05	5126.498	AMA-I	5127.388	5186.761	5208.999	8.20E+01	5463.927	27.62
		AMA-II	**5126.499**	**5126.985**	**5129.054**	**3.69E+00**	**5136.921**	24.13
		SGA	×	×	×	×	×	2.55
g11	0.750	AMA-I	0.750	0.750	0.750	3.35E-03*	0.750	14.92
		AMA-II	0.750	0.750	0.750	**4.00E-07***	0.750	11.11
		SGA	0.457442	0.922264	1.031979	6.59E-01	3.854752	4.29
g13	0.053950	AMA-I	0.054212	0.065926	0.193989	2.07E-01	0.811584	40.72
		AMA-II	**0.053950**	**0.055993**	**0.055616**	**8.63E-04**	**0.056540**	35.80

Opt= Optimal, '×'= feasible solution were not found, * indicates though the best, worst, median, and mean results are the same, standard deviation is positive due to rounding error. '–'= No result found in the literature. **Bold** fonts indicate the best result achieved.

The mean results achieved by AMA-II are also of good quality. In six problems (g03, g05, g11, B02, B07, and B09) the mean results are within 0.05% of the optimum. The results are within 4% of optimum in 3 problems (g13, B01, and B08) and only in one problem (B06) is the result more than 10% away (13.28%) from optimum. If we compare the mean results, AMA-II achieves better mean results than AMA-I in five problems (g05, g13, B01, B02, and B06). Although the mean results of AMA-I and AMA-II are same for the other five problems, AMA-II achieves better standard deviation for all those problems. The qualities of the other results of AMA-II are also remarkable.

Table 7.3: Statistics for 30 independent runs of different algorithms for the test problems.

Fn	Optimal	App.	Best	Median	Mean	St.Dev	Worst	Time(s)
		SGA	-44.775	-41.773	-41.553	1.81E+00	-37.910	3.47
B01	-47.765	AMA-I	-46.542	-44.950	-44.652	1.63E+00	-39.808	85.48
		AMA-II	**-47.688**	**-47.145**	**-47.155**	**3.09E-01**	**-46.696**	77.93
		SGA	964.124	975.253	973.961	3.76E+00	978.575	2.82
B02	961.715	AMA-I	961.716	964.836	965.085	2.39E+00	970.210	17.98
		AMA-II	**961.715**	**961.715**	**961.716**	1.21E-03	**961.720**	15.73
		SGA	218.009	661.675	653.398	2.05E+02	960.191	3.52
B06	193.724	AMA-I	**196.970**	**215.719**	229.847	3.07E+01	300.080	41.14
		AMA-II	201.420	216.121	**219.455**	**1.36E+01**	**237.181**	46.09
		SGA	0.076	0.754	0.797	4.50E-01	1.717	2.84
B07	0.000	AMA-I	0.000	0.000	0.000	8.40E-17	0.000	17.28
		AMA-II	**0.000**	**0.000**	**0.000**	0.00E+00	**0.000**	**11.83**
		SGA	**6.567**	6.583	6.582	5.15E-03	6.593	2.46
B08	6.334	AMA-I	6.582	6.582	6.582	5.39E-09	6.582	13.56
		AMA-II	6.582	6.582	6.582	**2.15E-09**	**6.582**	19.81
		SGA	-0.935	0.465	0.314	6.74E-01	1.282	3.71
B09	-1.000	AMA-I	-1.000	-1.000	-1.000	1.08E-08	-1.000	20.79
		AMA-II	-1.000	-1.000	-1.000	**8.43E-10***	-1.000	18.25

Opt= Optimal, '×'= feasible solution were not found, * indicates though the best, worst, median, and mean results are the same, standard deviation is positive due to rounding error. '–'= No result found in the literature. **Bold** fonts indicate the best result achieved.

After analyzing the results it is very clear that SGA is very simple and quick in solving the problems (on average it needs 3.33 seconds to solve each problem). However SGA could not solve three problems and the results for the other problems are not of high quality. Only in two problems (B02 and B08) the best results are within 5% of the optimum. Compared to SGA, AMA-II has a trade-off of speed against solution quality: AMA-II is slower but gives better results.

In AMA-II, during the initial generations it only uses the new LSLP process and so

the improvement index calculation and other four LSLPs (involving iterative process) are not used in that period. As a result the new LSLP reduces the execution time of AMA-II by 3.64% compared to AMA-I. The new AMA-II took on average 34.72 seconds to solve each problem. The results achieved by AMA-II show that the inclusion of the new LSLP has enhanced the ability of AMA in solving constrained optimization problems with equality constraints.

7.4.4 Effect of the new LSLP

The test results show that with the inclusion of the new LSLP, AMA-II always performs better than AMA-I. To visualize the internal effect of the new LSLP, two convergence curves are presented from the results with AMA-I and AMA-II, on problem g13. The objective function of this problem is nonlinear and involves equality constraints. As this problem is very difficult to solve (according to Mezura-Montes and Coello (2002)), this problem is chosen to show the effect of the new LSLP. Figure 7.3 shows the best objective function value achieved by the two algorithms at every generation for 1000 generations. To see the effect on the whole population, Figure 7.4 shows the average objective function of the whole population.

As the test problems include equality constraints, the feasible space is very tiny. During the initial generation, several individuals in the agent population may have better objective function values but with huge constraint violations. This is why the convergence curves drop from initial high values: although the initial objective function values may be good, the constraints are not satisfied. With time the constraint violation is reduced and objective function values move back towards optimal values, now with low constraint violation.

In most problems, AMA-II converges faster towards the best objective function value. It exploits the advantages of both the new LSLP early in the evolutionary process, and the other four LSLPs afterwards. Usually when the new LSLP is applied, it tries to move the agent's existing solution towards the randomly selected equality

constraints. As a result the agents may reach (or nearly reach) different points of different equality constraints. Hence it ensures enough diversity in the population with good quality solutions. After that AMA-II uses the other LSLPs, as did AMA-I. That is why AMA-II converges faster than AMA-I in Figure 7.3. However in some problems, like g03, g11, B07, and B09 both versions of AMA show similar behavior, since both versions achieve the optimal solution in their early generations and solved the problems easily.

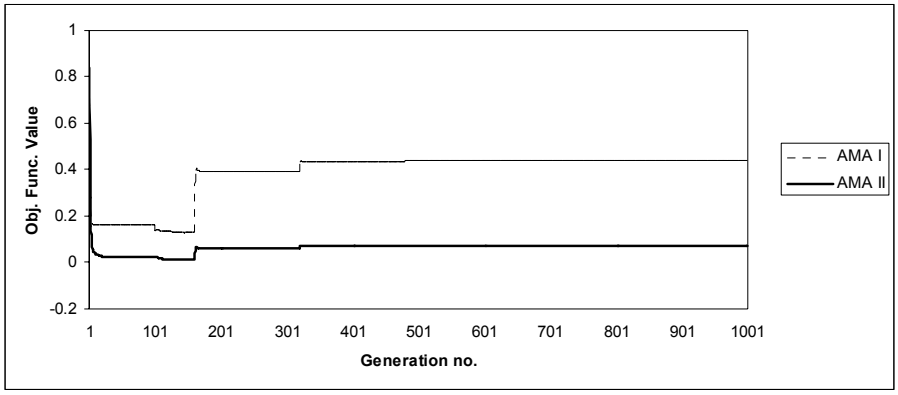


Figure 7.3: Convergence Curve from the best objective function of AMA-I and AMA-II for problem g13.

In Figure 7.4, for AMA-II, during the first 100 generations (while the new LSLP is working) the average of the objective function values of the population is very high with a high rate of change. This is due to the effect of the increased diversity by the new LSLP. When the new LSLP stops (after 100 generations) the average objective function improves slowly with the other four LSLPs. On the other hand AMA-I has lower diversity and converges slowly. The same behavior is seen for most other problems. So the new LSLP helps AMA-II to converge faster, with better quality solutions, than AMA-I.

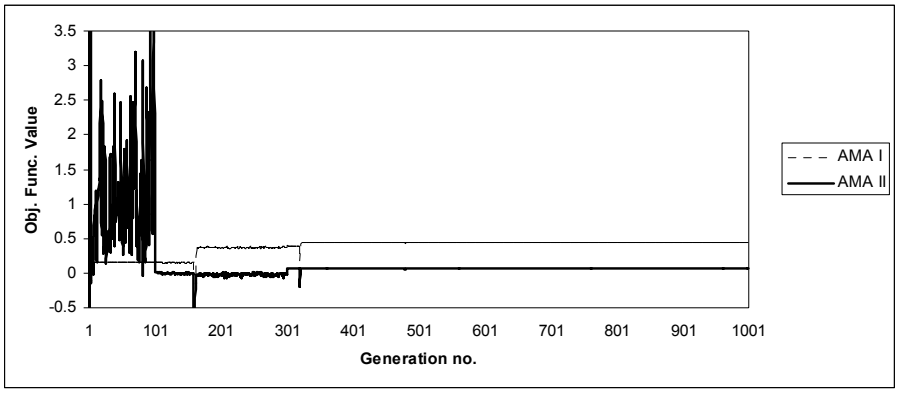


Figure 7.4: Convergence Curve from the average objective function of AMA-I and AMA-II for problem g13.

7.4.5 Effect of Probability of using LSLP

The probability of LSLP (P_L) is another important parameter of the algorithm. For low value of P_L a small number of agents are selected for LSLP. As discussed earlier the purposes of the LSLPs are to bring the agents' solutions towards different equality constraints (with the new LSLP) and explore the search space (through the other LSLPs). So a low value P_L allows lower diversity. On the other hand a higher value of P_L allows more agents to use LSLP. Figure 7.5 shows the diversity of the population, for the first 500 generations for problem g13, with different probability of LSLP (only the diversity using P_L values of 0.05, 0.1, 0.15 are shown to see the effects clearly). The average Euclidian distance among all solution vectors in the population is used to measure the diversity. In the first 100 generations the agents use the LSLP designed for handling equality constraints, which ensures enough diversity during that period. Afterwards we can see that higher P_L allows higher diversity in the population. The behavior is same for the other test problems.

Diversity is an important issue for the performance of any population-based search algorithm. Lack of diversity means the algorithm may not perform well; on the other hand, over-diversification causes slower convergence. To see the effect of P_L on the performance of AMA-II, the algorithm is run 30 times on each problem with different

values of P_L (0.05, 0.1, 0.15, 0.20, 0.25, and 0.3). For some problems, the algorithm may achieve the optimal solution as an outlier with any of these values of P_L. However if we consider the mean and standard deviation of the results we can see the effect of P_L. From the experiments we realize that for most problems the mean and standard deviations improve with an increase of P_L up to a point, then the quality degrades due to the over diversification. For g05, g13, and B01, AMA-II achieves the better mean results with P_L = 0.20. With P_L = 0.25 it has achieved better mean for B02. For some problems like g03, g11, B07 and B09 the algorithm achieves almost the same results with any P_L.

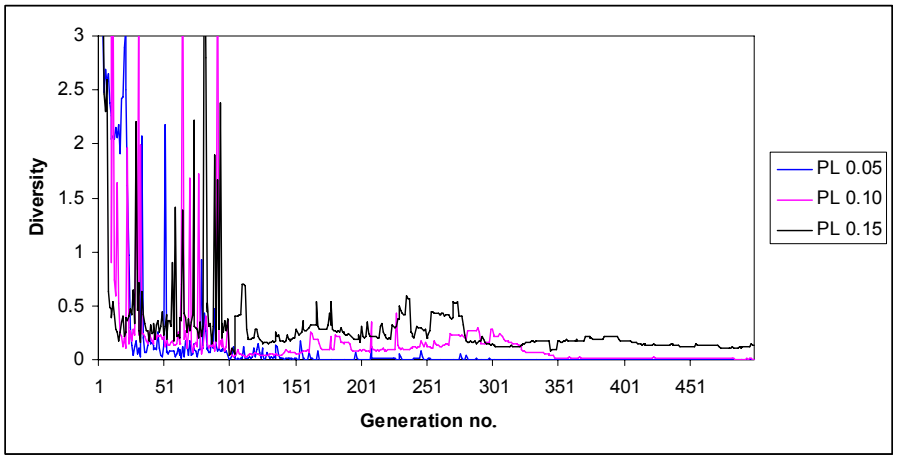


Figure 7.5: Effect of Probability of LSLP (P_L) on population diversity for problem g13.

Figure 7.6 shows the best and the mean performance of AMA-II on problem g13 as P_L varies; Figure 7.7 shows standard deviation, and Figure 7.8 shows average execution time, as P_L varies. It shows by increasing P_L both the mean and standard deviations improve gradually, but once P_L exceeds 0.2 the results are not improving. This is because higher P_L causes slow convergence in the population and so the performance degrades at the same level of generations. That indicates that increasing P_L can improve the performance of the algorithm, however, after a certain point there will be no significant improvement.

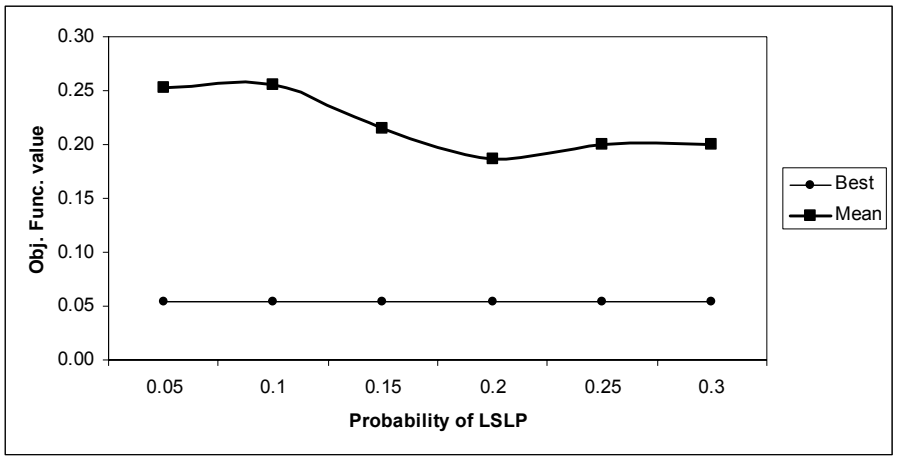


Figure 7.6: Effect of Probability of LSLP (P_L) on problem g13. Probability of LSLP vs. achieved best and mean results.

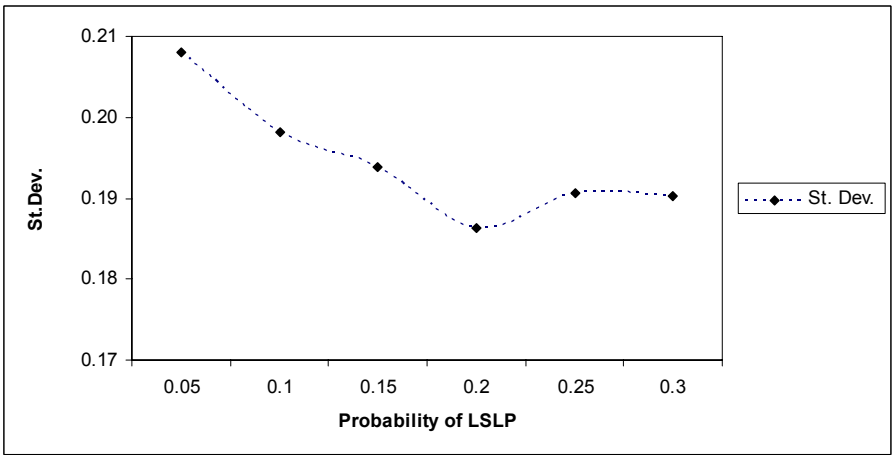


Figure 7.7: Effect of Probability of LSLP (P_L) on problem g13. Probability of LSLP vs. St.Dev. of achieved results.

More agents are allowed to apply LSLP by increasing the probability of P_L. This causes extra time to perform the additional task. The required time is increasing in proportion to the increase of P_L as shown in Figure 7.8.

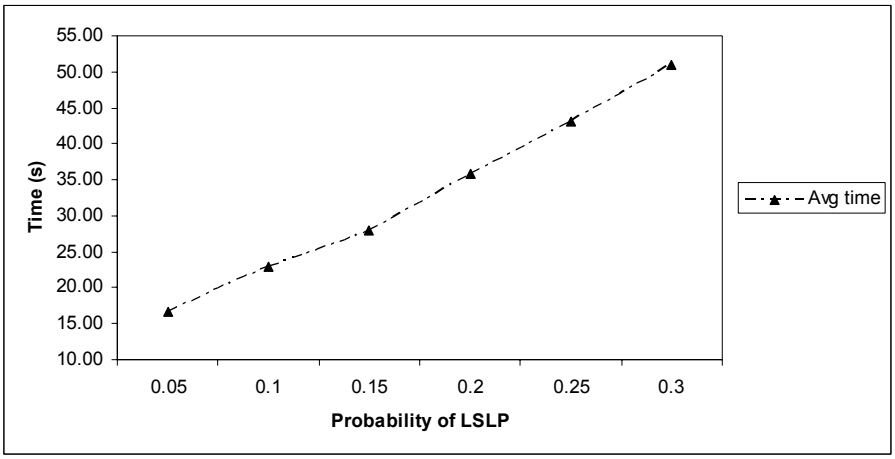


Figure 7.8: Effect of Probability of LSLP (P_L) on problem g13. Probability of LSLP vs. Average time required.

7.4.6 Summary

The experimental studies presented in this section demonstrate the value of ECHT in solving COPs. It is established that:

- AMA-II achieves faster convergence than AMA with the incorporation of ECHT while solving COPs with equality constraints, and enhances the quality of solutions in faster computational time.
- It is best to use a value of 0.20 for P_L, as it provides a balance between diversity in the population, quality of solutions and computational cost.

7.5 AMA with only the new LSLP (AMA-III)

So far the performance and issues of the new LSLP with AMA have been discussed. The new LSLP is used during the initial generations. After that the AMA uses other

LSLPs. As discussed earlier, although the existence of equality constraints reduces the feasible space size, it provides an advantage in finding the feasible solutions by exploring on the equality constraints. Since the new LSLP is designed to utilize this advantage, we are interested to see the performance if AMA uses only the new LSLP through the whole evolutionary process. To test this, AMA is modified. Here after the crossover, a percentage of agents only use the new LSLP, that means, at Step 5 the agents only use the new LSLP rather using all the LSLPs. This algorithm is called AMA-III, and the main steps of it are as follows:

Step 1. Create a random population, which consists of $M \times M$ agents.

Step 2. Arrange the agents in a lattice-like environment.

Step 3. Evaluate the agents individually. If the stopping criterion has been met, go to step 7, otherwise go to step 4.

Step 4. For each agent examine its neighborhood. Select an agent from its neighborhood and perform crossover.

Step 5. Select a certain percentage of agents; Apply only the LSLP designed for equality constraint

Step 6. Go to step 3.

Step 7. Stop.

To test the performance of the modified algorithm a set of experiments is carried out. For this experiment all the parameters are used as before i.e. population size (100), and probability of learning/LSLP (P_L= 0.2). The maximum number of generations considered is 3500. For each problem Table 7.4 shows the results from 30 independent runs for the new algorithm.

From the results in Table 7.4, we can see AMA-III is also very efficient in solving the test problems involving equality constraints. From the 10 problems it has achieved optimum results in 5 problems (g03, g11, B02, B07, and B09). The performance of the algorithm is also quite good in solving the other five problems. The achieved best results are very close to the optimum. For example the achieved best results for 3 problems are within 1% of the optimum results e.g. g05 (0.0003%), g13 (0.01%), and

B06 (0.89%), and for the other 2 problems the results are within 4% of the optimum results e.g. B01 (1.69%) and B08 (3.06%). Other results such mean, median, and worst results of AMA-III are also of good quality.

Table 7.4: Experimental results of AMA-III for the test problems (30 runs).

Prob	Optimal	Best	Median	Mean	StDev	Worst	Avg time
g03	-1.000	-1.000	-1.000	-1.000	9.75E-07*	-1.000	15.93
g05	5126.498	5126.511	5129.637	5133.582	8.51E+00	5155.896	18.33
g11	0.750	0.750	0.750	0.750	0.00E+00	0.750	14.44
g13	0.053950	0.053956	0.059302	0.060656	5.50E-03	0.070242	19.42
B01	-47.765	-46.957	-45.478	-45.589	6.27E-01	-44.748	17.27
B02	961.715	961.715	961.726	961.731	1.52E-02	961.773	12.05
B06	193.724	195.443	217.201	227.783	3.38E+01	355.225	14.66
B07	0.000	0.000	0.000	0.000	0.00E+00	0.000	14.50
B08	6.334	6.528	6.544	6.544	9.90E-03	6.564	15.67
B09	-1.000	-1.000	-1.000	-1.000	0.00E+00	-1.000	17.73

* indicates though the best, worst, median, and mean results are the same, standard deviation is positive due to rounding error.

The execution time of AMA-III is faster than AMA-I and AMA-II. On average it needs only 16.00 seconds to solve each problem. This is 55.60% less than AMA-I and 53.92% less than AMA-II. As AMA-III is not using the other four LSLPs and the calculation of improvement index, it is quite a bit faster than the previous versions.

As the execution of AMA-III is much faster than AMA-II, we are interested to see how good is the solution quality of AMA-III compared to AMA-II. In achieving best results both algorithms achieved optimum in five problems (g03, g11, B02, B07 and B08) and only AMA-II in g13. AMA-III has improved the performance in two problems in achieving best results e.g 2.967% in B06 and 0.8204% in B08. With the help of five LSLPs including the new LSLP for handling equality constraints, AMA-II performs better in three problems (g05, g13, and B01). However, the improvement with AMA-II

is not very high e.g. g05 (0.0002%), g13 (0.0111%) and B01 (1.5329%).

The mean results are same for the both algorithms in four problems (g03, g11, B07 and B09). However AMA-III shows better performance by achieving better standard deviation in g03, g11, and B09. In B08 AMA-III achived 3.9154% better mean result than AMA-II. For the other five problems the mean results of AMA-II are on average 3.2536% better than AMA-III. However for those five problems the best and mean results of AMA-III are on average only 0.5177% and 6.9409% from the optimum results.

Although AMA-II achieves better results with more LSLPs and more computational time than AMA-III, the achieved results of AMA-III are also of good quality. If we use student's *t*-test (95% confidence level, 60 degrees of freedom, and *t*-tabulated value 2), we can see in five problems (g03, g11, B6, B7 and B9) there is no significant difference between the performance of the two algorithms. In one problem (B08) AMA-III is significantly better than AMA-II. For the other four problems AMA-II is better.

Compared to AMA-III, AMA-II has a trade-off of speed against solution quality: AMA-II is slower but gives better results. However AMA-III is more than 50% faster than AMA-II and achieves good quality of solutions.

7.6 Chapter Summary

This chapter has introduced a new technique to handle equality constraints. The technique is used as a learning process in AMA. The extended AMA is now capable of solving nonlinear optimization problems with equality constraints more efficiently. The results show the enhanced performance achieved by the new AMA in terms of solution quality and computational time. The constraint handling techniques used here do not need any penalty functions or additional parameters. The performance of the proposed algorithms is tested using benchmark problems and the experimental results show promising performance.

The effect of the new learning process and probability of using the learning process

is also analyzed. The experimental results show that faster convergence can be achieved with the new learning process. The probability of learning (P_L) is also an important parameter. The performance of the algorithm improves with the increase of P_L up to certain point; beyond that point, high values of P_L may over diversify the population and slow down the convergence.

As the proposed equality constraint handling technique is experimentally successful with AMA, we are interested to see its performance with other EAs. In the next chapter, an investigation is made with a simple genetic algorithm.

Chapter 8

ECHT with Genetic Algorithms

In the last chapter, a new equality constraint handling technique was proposed. The technique was tested with AMA (presented in Chapters 4 and 5), and the revised AMA (known as AMA-II and AMA-III) performs better than AMA. It would be interesting to see how the technique performs with simple GA for equality constrained problems. The revised GA is tested on a set of standard benchmark problems. The results show that the proposed technique works very well on the benchmark problems considered in this research. This reinforces that the ECHT is useful in its own right, and not just as an addition to AMA.

8.1 Introduction

As discussed in earlier chapters, traditional GA has difficulty in dealing with equality constraints. In the previous chapter, a new equality constraint handling technique has been proposed and implemented successfully with AMA. In this chapter, it would be interesting to investigate the performance of the proposed equality constraint handling technique in conjunction with a simple traditional genetic algorithm to solve the equality constrained problems.

Here, the GA used in Chapter 3 is modified to incorporate ECHT. The revised GA is labeled as the Modified Genetic Algorithm (MGA) in this thesis. The modified algorithm is tested on the set of ten benchmark problems considered in the last chapter. The results are compared with the simple genetic algorithm presented in Chapter 3. Further comparison is made with a well-known ES-based based algorithm and with

AMA-II. The experimental results show that MGA has successfully solved all ten problems, achieving the optimum for six of them. The overall results are of acceptable quality with reasonable computational time.

The rest of this chapter is organized as follows. The next section describes the proposed MGA and its components. The effects of different components of the algorithm are studied in section 8.3. The outcome from these experiments is an understanding of how the performance of the proposed MGA is affected by these decisions, and thus how best to configure it. Then the performance of the proposed approach is described in section 8.4. Finally, section 8.5 concludes the chapter.

8.2 Modified Genetic Algorithm

In this section, the Modified Genetic Algorithm (MGA), which incorporates the new ECHT into GA, is introduced. In the proposed algorithm, the initial population $P^{t=0}$ is randomly generated. The individuals are evaluated and ranked based on their fitness. A set of individuals is selected as parents to produce offspring using crossover. This new population is called C^t. The selection process used here is based on tournament selection. From C^t certain percentages of the individuals apply the new ECHT, wherein the individuals try to reach and explore on the equality constraints. After applying ECHT, population C^t is called $C^{t'}$. The population for the next generation P^{t+1} is created from P^t and $C^{t'}$. The process is continued until the termination condition is reached. The main steps of the proposed algorithm are follows.

Pseudo code: Modified Genetic Algorithm.

Set generation no. $t = 0$;
Generate the initial population P^t at random;
REPEAT
  Evaluate the fitness of each individual in P^t and rank them;
  Apply modified tournament selection on P^t to select the parents, then apply crossover and generate C^t;
  Apply ECHT on C^t and generate $C^{t'}$;
  Produce generation P^{t+1} from P^t and $C^{t'}$;
  Set $t = t + 1$;
UNTIL the terminating condition is reached.

The representation of chromosome, ranking, and crossover operator used in MGA are as in chapter 3. The ECHT operator is as in chapter 7. The fitness evaluation, selection process, and the design of MGA are discussed below.

8.2.1 Fitness Evaluation

In solving constrained optimization problems it is necessary to evaluate the solution, particularly in the context of the underlying objective and constraint functions. For each individual the objective function value and Total Constraint Violation (TCV) are calculated. The TCV of an individual is the sum of absolute values by which the constraints are violated. A small tolerance δ on the total constraint violation is used i.e. if the TCV of an individual is less than δ, it is considered as feasible. As the evolutionary process continues the value of δ is gradually reduced. The equality constraints make the feasible region very small compared to the search space, which makes it hard for evolutionary algorithms to find feasible solutions. By using a positive δ, the algorithms can find some near feasible solutions, which are treated as feasible at

that stage. The value of δ is then gradually decreased, after every few generations, until it equals zero. Details of this process are discussed in section 8.3.1.

8.2.2 Selection

In the proposed selection process a parent is randomly selected from the elite individuals (e.g. top 30% of individuals). Another parent is the winner from a tournament selection between two individuals that are randomly selected from the rest of the individuals (i.e. non elites). These two parents then produce offspring after crossover. The elite parent plays an important role since the feasible space of this type of problem is very tiny.

8.2.3 Creating New Population

To generate a new population P^{t+1}, the current parent population P^t and the evolved child population $C^{t'}$ are merged and ranked. Based on experiments described below in section 8.3.1, the method for creating the new population is as follows. The top ranked 95% of individuals are selected for the next generation P^{t+1}. If all the individuals that have applied ECHT are not in P^{t+1} yet (as applying the ECHT may or may not improve the fitness of the individual), the remaining individuals are allowed to pass to the next generation. This allows individuals that are close to a particular equality constraint but have high TCV through to the next generation. Then some random individuals are allowed from P^t and $C^{t'}$ until the population P^{t+1} is filled up.

8.3 Experimental Studies

In this section, the experiments carried out to investigate the effect of different parameter values and design decisions are discussed. The outcome from these

experiments is an understanding of how the performance of the proposed MGA is affected by these decisions, and thus how best to configure it.

In the following few sub-sections, a number of experiments, to study the effects of different components of MGA on MGA's performance, is reported. In all these experiments, a fixed number of fitness evaluations is used for a fairer comparison.

8.3.1 Effect of ECHT

In MGA, after crossover a certain percentage of the individuals are randomly selected to apply ECHT (with probability P_E). The performance of the algorithm (over 30 runs each) with different values for P_E of (0.05, 0.1, 0.15, 0.20, 0.25, and 0.3) is tested while keeping the other parameters the same (P_C=0.90, dynamic tolerance in CV etc.).

Unlike traditional GAs, which use mutation to maintain diversity in the population, in the proposed MGA there is no mutation operator. However this purpose can be served by the ECHT. As discussed earlier (chapter 7), the proposed ECHT tries to bring the individuals towards the equality constraints or explore on the equality constraints. Up to a point, with lower value of P_E, ECHT may increase the diversity of the population. However, a higher value of P_E forces many individuals towards the equality constraints. As the feasible space is very tiny, the population loses diversity. Note that the effect of P_E is different from the effect of P_L described in chapter 7. AMA-II used P_L as a probability of LSLPs, which includes the new LSLP for equality constraints (only in the initial generations) and the other four LSLPs (in later generations). The last four LSLPs increase the diversity with the increases of P_L.

The effect of P_E on diversity is shown in Figure 8.1 for problem g13. The problem g13 is chosen since according to Mezura-Montes and Coello (2002) it is very difficult to solve. The objective function of this problem is nonlinear and involves equality constraints. To measure the diversity of the population the average Euclidian distance of the individuals is used. Figure 8.1 shows that the diversity of the population rises for a

while as P_E increases; then, as higher values of P_E drive lots of individuals towards the equality constraints, the population starts losing diversity.

If the performance is considered, for some problems such as g03, g11, and B09, MGA achieves the same result regardless of the value of P_E. In these problems MGA achieves the optimum easily. MGA achieves better results with lower values of P_E in problems B08 (P_E=0.05) and B06 (P_E=0.15). MGA achieves better results with a higher value of P_E (0.25) in B02. However in most problems (e.g. g05, g13, B01, and B08) MGA achieves better mean results with medium value of P_E = 0.20. This shows for better performance of the algorithm one should consider medium values for P_E, which ensures a balanced diversity in the population.

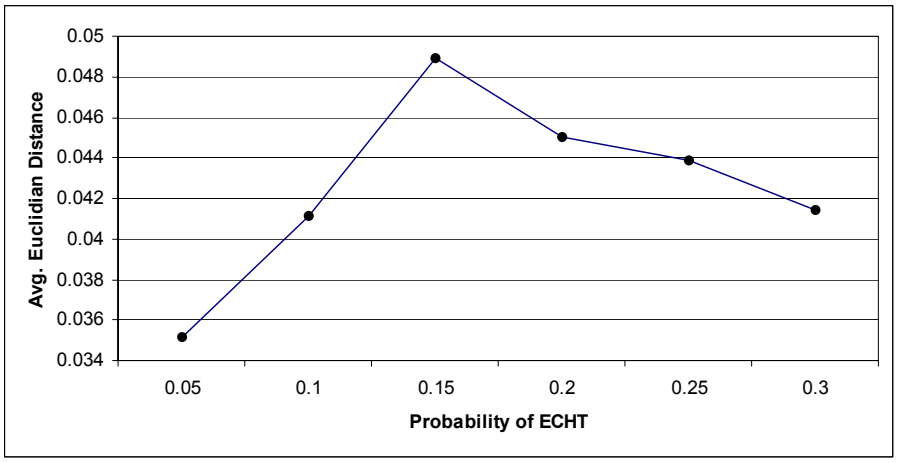


Figure 8.1: Effect of Probability of ECHT (P_E) on problem g13. Probability of ECHT vs. Average Euclidian distance in the population.

Figures 8.2 and 8.3 show the best, mean, and standard deviation of MGA in solving g13. They show by increasing P_E both the mean and standard deviations improve gradually, but after 0.2 the mean results are losing quality. This echoes the result that diversity increases as P_E increases up to a certain level, then the diversity decreases. For better performance of the algorithm we need a balanced diversity, neither too low nor too high, that is achieved here at P_E = 0.20.

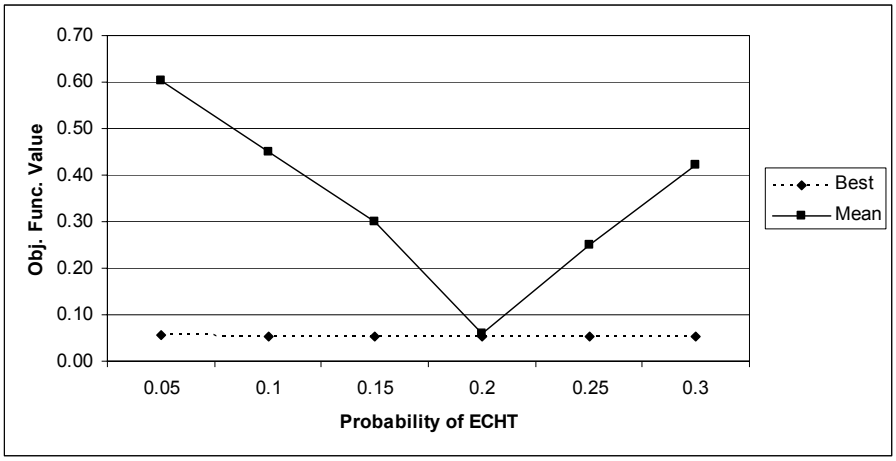


Figure 8.2: Effect of Probability of ECHT (P_E) on problem g13. Probability of ECHT vs. achieved best and mean results.

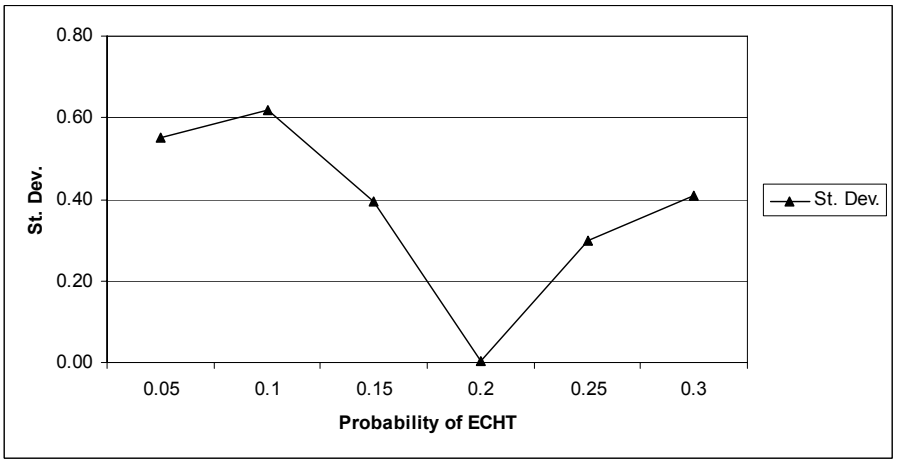


Figure 8.3: Effect of Probability of ECHT (P_E) on problem g13. Probability of ECHT vs. St.Dev. of achieved results.

8.3.2 Tolerance in Constraint Violation

During the fitness evaluation, a small tolerance δ on the total constraint violation is used. If the TCV is of an individual is less than δ, the individual is considered as feasible. To see the effect of δ, two set of experiments are carried out for 30 runs. In the

first set, the value of δ is used equal to 0. The objective function value and the TCV was calculated normally. In the second set, the value of $\delta > 0$. Since the equality constraints make the feasible region very small, by using $\delta > 0$, the algorithm allows some good quality infeasible solutions to be considered as feasible. Initially δ is assigned to 1. After every 16% of the maximum generation number, δ is divided by 10. Finally after 80% of the generations δ is left fixed at 0.0. During this experimentation the other parameters remain the same (e.g. P_C=0.90, P_M=0.20).

In problems g03 and g11, both approaches achieved the optimal result, since they are easy for MGA to solve. For the rest of the problems MGA achieves better results with the use of the small tolerance.

Figure 8.4 presents a convergence curve to show the effect of using δ in equality constraint for problem g13, up to 350 generations. It is clear that the algorithm, with a positive δ, converges to a better solution compared to the algorithm without δ.

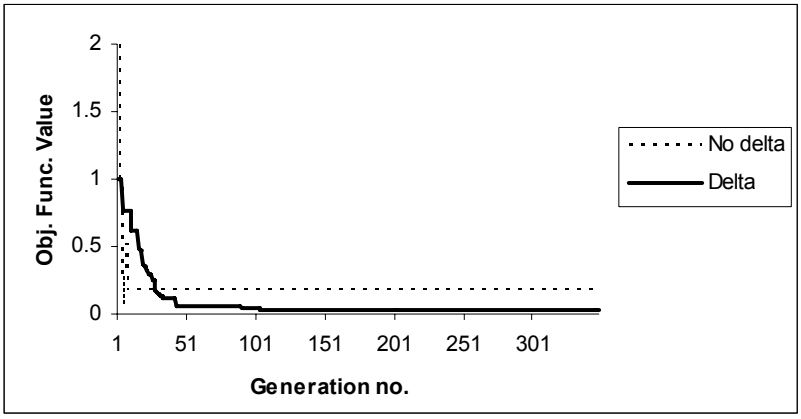


Figure 8.4: Convergence Curve from the objective function with and without using delta (δ) for problem g13.

8.3.3 Selection Process

As discussed earlier, a group of best individuals is defined as the elite group. In our selection process for reproduction, one parent is directly chosen from the elite group and

another parent is chosen via a tournament of two individuals. The elite parent plays an important role since the feasible space of this type of problem is very tiny. Selection of an elite as a parent will help to produce good quality offspring quickly. In this sub-section, the effect of the size of the elite group on the algorithm's performance is examined.

Five sets of experiments are carried out, considering the top 10%, 20%, 30%, 40%, and 50% of individuals as the elite group. The other parameters remain the same during this experimentation (e.g. P_C=0.90, P_M=0.20). In each case, the algorithm is executed 30 times with different seeds. The experimental results show that in problems (g03, g11) all the approaches achieve the optimal result. In problem B02, MGA achieves better results with the elite consisting of the top 20% of individuals. For B08 and B09, it achieves better results with the top 40% of individuals as the elite. In most other problems (e.g. g05, g13, B01, B06), performance was better with the top 30% of individuals as the elite.

In problems with equality constraints the feasible space is very tiny, and it is very difficult to find feasible solutions. If only a low percentage (e.g. the top 10%) of individuals is considered as elite, it always selects a parent from a small group of individuals. The diversity of the population will be reduced, since every offspring inherits genetic properties from these elites. However it needs the help of the top elites to guide the evolutionary process. If a high percentage (e.g. the top 50%) of individuals is considered as elite, the performance also suffers. In this case, a significant portion of the population is considered as elite, but many of the individuals may not be feasible since the feasible space is very tiny. It needs a balanced percentage of the top ranked individuals in the elite group. That is why MGA performs better for most problems with the top-ranked 30% of individuals as the elite.

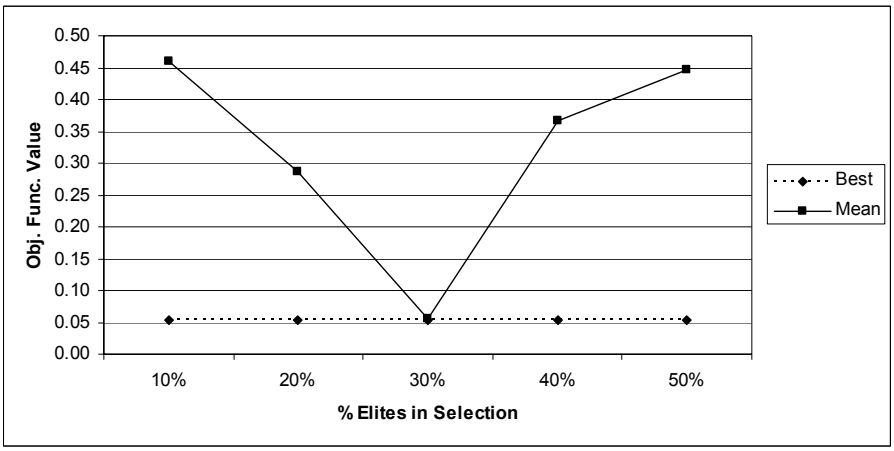


Figure 8.5: Effect of elites in selection process on problem g13. Percentage of Elites vs. achieved best and mean results.

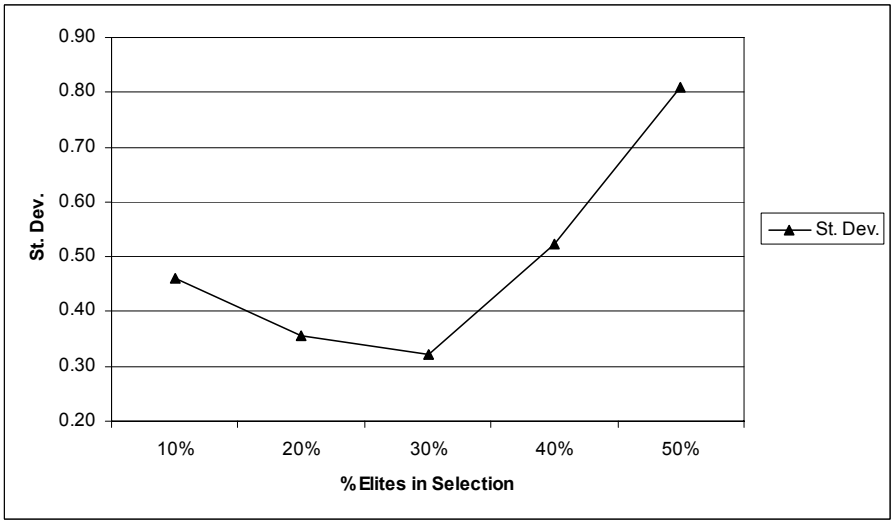


Figure 8.6: Effect of elites in selection process on problem g13. Percentage of Elites vs. St.Dev. of achieved results.

Figures 8.5 and 8.6 show the effect of the percentage of top ranked individuals as the elite in best, mean, and standard deviations for problem g13. The best achieved results of all approaches are very close to the optimal. As the best results could be an outlier in the case of evolutionary algorithms, we need to consider the mean and standard deviation. The performance (mean results, st.dev.) of MGA is improving with increasing

the elite zone up to 30%, then it starts decreasing.

8.3.4 Design of Next Generation

Diversity is a critical issue in population based algorithms. Lack of diversity may lead the algorithm to be trapped in local optima. On the other hand, over diversification may cause slow convergence. The design of the next generation plays an important role in this regard. Now the question arises how the next generation will be created?

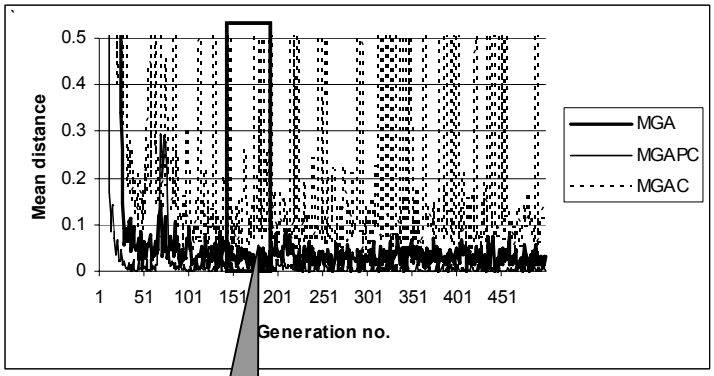
In this section, experiments are carried out to investigate the performance with three different approaches.

The first approach (called MGA-C below) is to consider the child generation (generated after crossover and ECHT) as the next generation population. It will ensure enough diversity in the population since the offspring from ECHT try to reach or explore on different equality constraints.

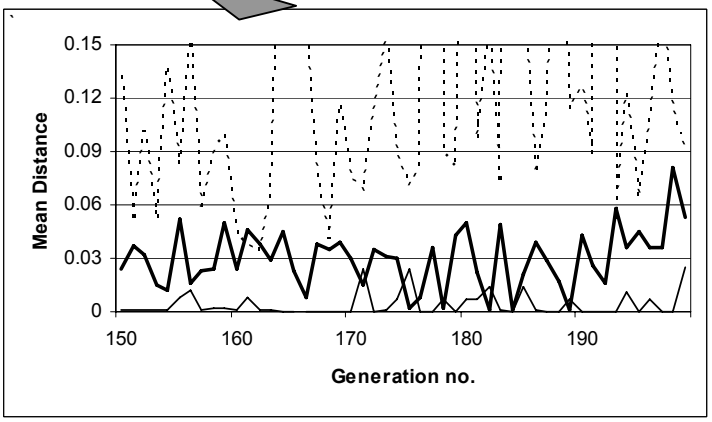
The second approach (MGA-PC) is to rank the parent and child population, and take the top ranked individuals from them to form the next generation. As this approach is only giving preference to good individuals, it may decrease the diversity.

In the third approach (MGA), most (e.g. 95%) of the next generation is chosen from the top ranked individuals from the parent and child population. The rest of the population is filled up by lower ranked individuals that applied ECHT (and so are close to an equality constraint) and random individuals.

After running each set of experiments for 30 runs the best, median, mean, standard deviation, and worst results are compared. It is found that MGA-C achieved the optimum in one problem (B02), MGA-PC achieved the optimum in three problems (g03, g11, B08), and MGA achieved the optimum in six problems (g03, g11, g13, B02, B07, and B09). In addition, for most problems (g05, g13, B01, B02, B06-B09) the results (mean, standard deviation, median) with MGA were better than with the other two approaches.



(8.7.1)



(8.7.2)

Figure 8.7: Diversity of population with different design of Next generation.

(8.7.2 is an enlargement of 8.7.1)

The reason for MGA's superiority can be found from Figure 8.7. Figure 8.7.1 shows the average Euclidian distance of the individuals (i.e. Diversity of population) for the first 500 generations when solving problem g13, with each of MGA, MGA-C and MGA-PC. To visualize the effect more clearly Figure 8.7.2 shows the expanded version of Figure 8.7.1 from generation number 150 to 200. Figure 8.7 shows that the diversity of the population is very high in MGA-C, since the next generation is formed from the child population. On the other hand, in MGA-PC the diversity is very low, since only the best individuals from the parent and child population are allowed to move to the next generation and the feasible space is very tiny (the top ranked individuals may be very close to each other). In the case of MGA, it maintains balanced trade-off between

the diversity and convergence, and its diversity lies between MGA-C and MGA-PC.

8.3.5 Section Summary

In this section, experiments are described which show how the performance of the proposed MGA is affected by various parameter settings and design decisions, and provided explanations from the observations. It is established that:

- It is best to use a value of 0.20 for P_E, and an elite consisting of the top 30% of individuals. These are both medium settings, which provide a balance between diversity in the population, quality of solutions, and rate of convergence;
- Allowing a small tolerance for violation of equality constraints helps the quality of solutions and the speed of convergence. By gradually reducing the tolerance to zero it can ensure that the equality constraints are met;
- The third approach to forming the next generation, based mainly on elitism but also including lower ranked individuals that are likely to be close to equality constraints, is best.

In the experiments described in the next section, the proposed MGA is configured with these "best" settings.

8.4 Evaluation of the Proposed MGA

In this section, the proposed MGA is evaluated on the selected benchmark problems, and compared to SGA, a well-known algorithm from the literature, and AMA-II.

8.4.1 Experimental Results and Discussions

Initially, a simple genetic algorithm (SGA) (presented in chapter 3) using tournament selection, SBX crossover (Deb and Agrawal, 1995) and parameter based mutation

operator (Deb, 2000) was used to solve the test problems. The aim was to see the performance of SGA in solving constrained optimization problems with equality constraints. The initial solution vectors are randomly generated within the boundary of each decision variable. As crossover is the major instrument of variation and innovation in GAs, with mutation insuring the population against permanent fixation at any particular locus and thus playing more of a background role (Holland, 1975), a high probability for crossover ($P_C = 0.90$) and with a low probability for mutation ($P_M = 0.2$) is used. The number of fitness evaluations is set to 350,000 as in (Elfeky *et al.*, 2006; Runarsson and Yao, 2000), which allows a maximum of 3500 generations with a population size of 100.

The best, median, mean, standard deviation (st.dev.), and worst results, as well as execution time, for the well-known problems (g03, g05, g11, and g13) are given in Table 8.1. The results for the other test problems (B01, B02, B06-B09) are given in Table 8.2. An '×' in the Tables indicates that the algorithm did not find any feasible solution.

From Table 8.1 and 8.2, it can be seen that SGA is very fast considering the execution time. However it could not find feasible solutions for 30% of the test problems (g03, g05, and g11). For the other test problems, the results are not convincing, even the best results achieved are far from the known optimum.

The MGA is then run with the same parameters. From the results in Table 8.1 and 8.2, it is clear that MGA overcomes the shortcomings of SGA while solving the COPs with equality constraints. MGA obtains the optimum for six of the ten benchmark problems (g03, g11, g13, B02, B07, and B09), and for the other four problems the results are extremely close (within 0.30%) to the known best values. For g05 and B06, which involve both equality and inequality constraints, MGA achieved very close to the optimal results. B06 can be considered as a special type of problem. In this problem the objective function depends only on one variable and this variable is involved in only the inequality constraint. Since the proposed algorithm is mainly designed to exploit the advantage of equality constraints, the other genetic operators used in MGA help to solve this problem.

8.4.2 Comparison with Other Algorithms

In this section, the performance of MGA is compared with another well-known algorithm (Runarsson and Yao, 2000) and AMA-II. Runarsson and Yao (2000) (abbreviated as RY) used an interesting ranking procedure (known as stochastic ranking) in their ES-based algorithm, and solved all the *g*-series problems. As the ES-based algorithm has produced the best results known so far for all the test problems (g01-g13) its results have been included for comparison.

Table 8.1: Statistics for 30 independent runs of different algorithms for problems with equality constraints in g-series

Fn	Optimal	App.	Best	Median	Mean	St.Dev	Worst	Time(s)
		SGA	×	×	×	×	×	4.14
g03	-1.000	RY	-1.000	-1.000	-1.000	1.90E-04	-1.000	–
		MGA	-1.000	-1.000	-1.000	**0.00E+00**	-1.000	4.47
		SGA	×	×	×	×	×	3.52
g05	5126.498	RY	**5126.497**	5127.372	5128.881	3.50E+00	5142.472	–
		MGA	5126.459	**5126.907**	**5128.144**	5.21E+00	**5137.793**	4.85
		SGA	×	×	×	×	×	2.55
g11	0.750	RY	0.750	0.750	0.750	8.00E-05	0.750	–
		MGA	0.750	0.750	0.750	**4.52E-09***	0.750	3.77
		SGA	0.457442	0.922264	1.031979	6.59E-01	3.854752	4.29
g13	0.053950	RY	0.053957	0.057006	0.067543	3.10E-02	0.216915	–
		MGA	**0.053950**	**0.056655**	**0.057864**	**3.81E-03**	**0.065063**	4.71

Opt= Optimum, '×'= feasible solution were not found, * indicates though the best, worst, median, and mean results are the same, standard deviation is positive due to rounding error. '–'= No result found in the literature. **Bold** font indicates the best result achieved.

Since the six test problems (B01, B02, and B06-B09) are new in the literature, no results for RY are available. RY successfully solved the first four problems, and are well known for their performance in solving the *g* series problems.

For g05, none of the algorithms has achieved the optimum. Due to the use of tolerance in equality constraint, RY's best result is lower than the optimal. However the mean, median, and worst results of MGA are better than RY.

For problems g11 and g03, both RY and MGA achieved the optimal result and the same mean and median results. However MGA achieved better standard deviation. For g13, only MGA has achieved the optimal result. The achievement of MGA is better than RY. The difference is statistically significant (for the statistical test Student's *t*-test is used and degrees of freedom is considered here 60 and *t*-tabulated value is 2).

Table 8.2: Statistics for 30 independent runs of different algorithms for other test problems

Fn	Optimal	App.	Best	Median	Mean	St.Dev	Worst	Time(s)
B01	-47.765	SGA	-44.775	-41.773	-41.553	1.81E+00	-37.910	3.47
		MGA	**-47.627**	**-47.022**	**-46.988**	**3.50E-01**	**-46.497**	6.04
B02	961.715	SGA	964.124	975.253	973.961	3.76E+00	978.575	2.82
		MGA	**961.715**	**961.717**	**961.717**	**1.71E-03**	**961.721**	3.80
B06	193.725	SGA	218.009	661.675	653.398	2.05E+02	960.191	3.52
		MGA	**194.113**	**225.487**	**205.921**	7.87E+01	**288.427**	5.24
B07	0.000	SGA	0.076	0.754	0.797	4.50E-01	1.717	2.84
		MGA	**0.000**	**0.000**	**0.000**	**0.00E+00**	**0.000**	3.55
B08	6.333	SGA	6.567	6.583	6.582	**5.15E-03**	6.593	2.46
		MGA	**6.338**	**6.324**	**6.316**	3.60E-02	**6.366**	3.49
B09	-1.000	SGA	-0.935	0.465	0.314	6.74E-01	1.282	3.71
		MGA	**-1.000**	**-1.000**	**-1.000**	**1.78E-05***	**-1.000**	5.06

Opt= Optimal, '×'= feasible solution were not found, * indicates though the best, worst, median, and mean results are the same, standard deviation is positive due to rounding error. '–'= No result found in the literature. **Bold** fonts indicates best result achieved.

The comparison of the execution time against RY can not be done, because their code was not available. For SGA a fair comparison of execution time can be made because it is being used in the same computing environment.

SGA is very simple and on average it needs only 3.33 seconds to solve each of these test problems. However SGA could not solve three problems (g03, g05, and g11). In B02 and B08 the achieved mean results of SGA is within 5% of optimum; in all other problems it could not achieve mean results within 10% of optimum.

MGA achieved mean results within 10% of optimum in all the test problems. In

seven problems the achieved means of MGA are within 1% of optimum. For the other three problems the achieved mean results are within 10% of optimums (1.27% in B01, 6.3% in B06 and 7.25% in g13). In spite of better performance, MGA only took on average 4.45 seconds to solve each problem.

The incorporation of ECHT radically enhanced the performance of MGA. If we compare the result of it with AMA-II (presented in chapter 7), both algorithms are now able to handle the equality constraints efficiently and achieve good quality solutions. In 60% of the test problems both algorithms achieve optimum results. In some problems like g03, g11, B07, and B09, the algorithms achieve the optimum as the best, mean, median, and worst results. However in two problems (g03 and g13) MGA achieved better standard deviation than AMA-II. In one problem (B09) AMA-II achieves better standard deviation. MGA achieves better mean results than AMA-II in three problems (g05, B06 and B08). In those problems the best and mean results of AMA-II are only on average 2.629% and 5.749% from optimum. AMA-II also achieves better mean results than MGA in three problems (g13, B01, and B02). However the achieved results of MGA are also very convincing in those problems. On average the best and mean results are only 0.095% and 2.96% far from the optimum. On average AMA-II needs 34.72 seconds to solve the test problems. So MGA provides competitive quality solutions with faster execution time.

Form the results we can see with help of ECHT, the proposed algorithm not only achieves better quality solutions, but is also time efficient in solving the COPs with equality constraints.

8.4.3 The Effect of more Fitness Evaluations

So far, SGA is executed up to 350,000 fitness evaluations. One question that may arise is whether SGA could find as good solutions as MGA if it were allowed to run for enough generations (i.e. allowed for more fitness evaluations). To investigate this SGA is executed for 500,000 fitness evaluations.

It is found that SGA improved its performance, by achieving feasible solutions in

problems g03, g11, B02, and B09. The new mean results for B02 and B09 are 0.55% and 3.99% respectively from the optimum. However the mean results of g03 and g11 are still 98.26% and 15.46% away from the optimum.

This shows that SGA is still so unlikely to find an optimal solution (even with the additional fitness evaluations), because the feasible space of these problems is so small, that an idea like this new ECHT is essential in order to have a realistic chance of finding optimal solutions in practice.

8.5 Chapter Summary

This chapter has introduced a modified genetic algorithm which combined simple GA with the proposed equality constraint handling technique. The constraint handling technique used here does not need any penalty functions or parameters, and provides good quality solutions in less computation time compared to other algorithms considered in this thesis.

The performance of the algorithm is investigated in solving a set of ten test problems. The results show that the proposed algorithm is robust in its handling of both linear and nonlinear equality and inequality constraints. The algorithm shows very impressive performance by achieving optimal results in six problems. The performance of the proposed algorithm is compared with genetic algorithm, AMA-II and one ES-based algorithm. The results show that the proposed approach gives mostly improved or comparable results to other algorithms.

The effects of the proposed ECHT and other design components of the algorithm have been analyzed. The experimental results show probability of ECHT is an important parameter for MGA. The performance of the algorithm increases with the increase of P_E, up to a certain level. The algorithm uses a small tolerance on the total constraint violation, which allows finding some near feasible solutions easily at that stage. The experimental studies justify this use of dynamic relaxation of the total constraint violation.

Chapter 9

Conclusions and Future Research Directions

This chapter briefly describes the research carried out in this thesis. It also discusses the findings and conclusions, and indicates some possible future research directions.

9.1 Summary of Research Done and Conclusions

In this thesis, genetic algorithms for solving constrained optimization are studied and analyzed. To enhance the performance of the algorithms, first, a new Agent-based Memetic Algorithm (AMA) is designed. In the new algorithm, a number of local search techniques are proposed for agent learning. Then a Search Space Reduction Technique (SSRT) is proposed to improve the quality of randomly generated initial solutions, while sacrificing very little in diversity of the population. After successfully implementing the SSRT with different evolutionary algorithms including AMA, an investigation is made to handle the equality constraints efficiently. The equality constraint handling technique is used as a new learning process for AMA. The superior performance of AMA, for problems with equality constraints, inspires to test the technique with simple GA. With this aim, this thesis also proposes a modified genetic algorithm for solving COPs with equality constraints.

The details of these developments, experimental results, and findings are briefly discussed below.

9.1.1 Genetic Algorithms in Solving COPs

In Chapter 3, a simple genetic algorithm (SGA) is implemented for solving a set of state-of-the-art constrained optimization problems. To design the genetic algorithm, suitable crossover, mutation and constraint handling techniques are used. Although SGA is fast in execution time, the quality of the results are not convincing enough for use of this algorithm for solving constrained optimization problems. In only 15.38% of problems has it achieved optimum results. However, for the rest of the test problems, the results are not pleasing. In 23.07% of problems it could not find any feasible solutions at all. The average deviation of the best result achieved by SGA for the rest of the problems is 107.89% from the optimum results. The study also shows that SGA suffers in solving problems with tiny feasible space. Especially when the test problems involve equality constraints, the performance of SGA is very poor. For the equality constrained problems, the algorithm could not solve 75% of them and the performance is also not satisfactory for the remaining problem.

The effect of different parameters is investigated in pursuit of better performance of SGA. Though the results can be improved slightly by increasing the number of generations, still SGA is unlikely to find good quality solutions for several problems.

This demonstrates that an improved algorithm is required to solve constrained optimization problems.

9.1.2 Agent-based Evolutionary Algorithms

Chapter 4 discusses agent-based evolutionary algorithms, and different issues that differentiate an agent-based EA from an independent EA. The chapter has also introduced a new agent-based memetic algorithm for solving COPs, by tailoring multi-agent concepts into a memetic algorithm. The individual candidate solutions of problems are represented as agents with additional characteristics. The agents have the ability to independently select a suitable life span learning process as an agent learning approach. Evolutionary operators consist of only crossover and one of the self-

adaptively selected LSLPs. To the best of our knowledge, solving constrained optimization problems using an agent-based memetic algorithm is new in the literature.

In chapter 5, the performance of AMA is investigated in solving a set of test problems, which includes five new problems plus 13 existing well-known problems. The results show that the proposed algorithm is robust in solving different types of COPs. As the agent exchanges information with its neighbors, AMA does not need any ranking for the whole population. The agent selects a neighborhood agent by using pair-wise comparison to mate, which handles constraints indirectly. Also in the self-adaptation process of learning, while calculating the improvement index, the constraints are indirectly handled. These two levels of constraint handling, with appropriate neighborhood size, SBX crossover, and LSLP, ensure the superior performance of AMA in handling constraints.

The algorithm shows very impressive performance by achieving optimal results in 13 problems. The performance of the AMA is compared with five GA-based and one ES-based algorithms. The comparisons show that the proposed approach gives mostly improved or comparable results to other algorithms. Statistical significance tests show that the proposed algorithm's performance is better than the well-known ES-based algorithms for the well-known 13 problems.

The effect of the proposed LSLPs is analyzed, showing that adaptively selecting one of the LSLPs achieves better results ensuring both diversity and convergence. The effect of Probability of Learning (P_L) is also analyzed. The performance of the algorithm increases with the increase of P_L, but after a certain level it causes over diversification. While analyzing the effect of neighborhood size, the experimental studies show the combined approach (i.e. applying 4 neighbors and 8 neighbors interchangeably) performs better than the other types of neighborhood. The effect of population size is also analyzed, which shows a low population size is not able to achieve good results. With the increase of population size the performance improves, however after a certain population size there is no significant improvement in the solutions.

9.1.3 Problems with Tiny Feasible Space

In many practical optimization problems, the feasible spaces are very tiny. These problems are very challenging, as they require searching a huge variable space in order to locate feasible points with acceptable quality. Chapter 6 proposes a simple method to improve the quality of randomly generated initial solutions, while sacrificing very little in diversity of the population, for solving COPs with tiny feasible space. The proposed method, which is termed the search space reduction technique in this thesis, directs the selected low quality infeasible solutions towards the feasible space.

The performance of SSRT is investigated, in conjunction with AMA, simple genetic algorithm and three well-known algorithms found in the literature, by solving a set of test problems and a real world case problem. This approach usually improves the performance of the algorithms in terms of either solution quality or computational time or both, at the cost of an additional step with $O(M^2)$ complexity (where M is the number of infeasible solutions in the initial population). From the results of the real world problem, it is evident that the method is more appreciable for large scale problems with tiny feasible space. Although the idea of SSRT is very simple, the results justify the use of SSRT with evolutionary algorithms.

9.1.4 Handling Equality Constraints

Chapter 7 presents a new equality constraint handling technique, which enhances the performance of AMA in solving constrained optimization problems with equality constraints. The technique is basically used as an agent learning process in AMA. The ECHT brings an individual solution to a point on an equality constraint from the current position of an individual solution, and then explores on the constraint landscape. The extended AMA is capable of solving nonlinear optimization problems with equality constraints more efficiently. The constraint handling techniques used here do not need any penalty functions or additional parameters. The performance of the proposed algorithms is tested using benchmark problems. The experimental results show the

enhanced performance achieved by the new AMA in terms of solution quality and computational time.

The effect of the new learning process and probability of using the learning process is also analyzed. The experimental results show that faster convergence can be achieved with the new learning process. The probability of learning is an important parameter in the process. The performance of the algorithm improves with the increase of P_L up to a certain point; beyond that point, high values of P_L may over diversify the population which slows down the convergence.

9.1.5 ECHT with Genetic Algorithms

The superior performance of AMA (in chapter 7), for problems with equality constraints, inspires to test the same technique with GA. Chapter 8 proposes a modified genetic algorithm for solving COPs with equality constraints. In MGA, only crossover operator and the ECHT are used. The results show that the proposed approach overcomes the limitations of SGA discussed in Chapter 3, and provides significantly improved results compared to SGA reported in this thesis. The algorithm shows very impressive performance by achieving optimal results for 60% of the problems, whereas SGA could not solve 75% of these equality constrained optimization problems. The achieved best results are on average 0.05668% from the optimum. In 70% of problems the achieved mean results of MGA are within 1% of optimum. For the other three problems the achieved mean results are within 10% of optimums. In spite of better performance, MGA only took on average 4.45 seconds to solve each problem. This shows the proposed algorithm not only achieves better quality solutions, but also is time efficient in solving COPs with equality constraints.

The effects of the proposed ECHT and other design components of the algorithm have been analyzed. Probability of ECHT is an important parameter; the performance of the algorithm increases with the increase of P_E, up to a certain level.

The algorithm uses a small tolerance on the total constraint violation, which allows

finding some near feasible solutions easily at the earlier stage of evolution. The experimental results justify the use of dynamic relaxation of the total constraint violation.

9.1.6 Summary

The key contributions in this thesis are the AMA architecture and the ECHT. In problems with only inequality constraints, the AMA architecture helps a lot, while the ECHT is not relevant. The algorithm can solve all the test problems efficiently, with suitable parameter settings. In problems with equality constraints, the ECHT helps a lot with both AMA and SGA. With ECHT, AMA-II can solve the problems with better quality solutions and even with lower computational cost such as a 3.64% reduction of execution time compared to AMA-I. With ECHT, the proposed MGA is able to solve all the test problems, which was not at all possible with SGA.

One other important contribution of this research is the design of SSRT and its implementation with different EAs in solving COPs with tiny feasible space. The technique shows its value by improving the solutions when it is used with any of a variety of different techniques such as AMA, SGA, NSGA-II and MCA.

The detailed experiments have shown in the individual chapters how each of the algorithm performance behaves as various important parameters are varied. This demonstrates the robustness of the algorithms – small changes in the values of parameter do not cause large changes in performance – and it helps to identify the optimum settings for some parameters.

Some of the key experimental results are briefly indicated below.

- Adaptively selecting one of the LSLPs achieves better results, as that ensures both diversity and convergence (see Chapter 5).
- For the neighborhood size of the agents, the combined approach performs better than the other types of neighborhood (see Chapter 5).

- With the increase of population size the performance of AMA improves, however after a certain population size there is no significant improvement in the solutions (see Chapter 5).
- In the design process of SSRT, the number (or percentage) of infeasible solutions used in calculating the centroid, and a stopping criterion for SSRT, are very important factors. As the diversity of the population decreases with application of SSRT, the diversity measure is considered as a stopping criterion.
- The ECHT, which is considered as a learning process in AMA, achieves faster convergence.
- A small tolerance for violation of equality constraints helps the quality of solutions and the speed of convergence.

In conclusion: with the help of the algorithms proposed in this thesis, COPs can be solved more efficiently. For a new constrained optimization problem, if the size of the feasible space is not tiny, AMA (see Chapter 5 and 6) can be used. However for problems with tiny feasible space, SSRT (see Chapter 7) can be applied before the evolutionary process. With the presence of equality constraints, the use ECHT (see Chapter 8) with AMA is recommended. For faster performance in solving such problems, MGA (see Chapter 8) is highly recommended. The experimental study shows the strength of the algorithms, and the achieved results of the algorithms presented in the thesis are as good or better than others in the existing literature.

9.2 Future Research Directions

Various avenues of further research stem from the work carried out in this thesis. The current research can be extended in a number of different ways. The performance of proposed AMA and SSRT can be tested on more test and practical problems. The performance of the equality constraint handling technique can be extended for more equality constrained optimization problems.

In addition, some other aspects can also be introduced in conjunction with our proposed techniques, which are described below.

- In the design of AMA, more intelligent characteristics can be incorporated in the agents. The agents may have their own information storage, information retrieval, and decision support system to analyze the characteristics and fitness landscape of the problems and make their own judgment accordingly.
- In AMA, the individual agents select a LSLP from their parents LSLPs based on their improvement index. However, a different approach could be chosen to select the LSLPs. For example, an individual could select a LSLP, based on its fitness or random.
- In designing the SSRT (chapter 6), to calculate the centroid, the arithmetic mean of the participating solutions of respective variables is considered, that gives equal priority to all the individuals under consideration. However, weighted approach or other techniques may be considered to calculate the centroid.
- For the stopping criterion of SSRT, diversity reduction is considered in this thesis. Other measures such as the feasibility of the whole population could be used.
- In the design of ECHT (presented in Chapter 7), to explore on the equality constraint, only two variables are considered to be changed. More than two variables could be considered.
- In solving equality constrained problems, it could be possible to find the optimum by searching only one constraint that involves all the variables. This could be considered in the design of ECHT.
- The methodologies proposed in this research can be incorporated easily with different population-based evolutionary algorithms to improve their performance. The proposed search space reduction technique can be used with any population-based algorithms, even with multi-objective based evolutionary algorithms for constraints handling. After applying SSRT, the algorithms can

start the evolutionary process with a set of improved quality solutions. The multi-objective based constraints handling algorithms can also use the ECHT proposed in this research while solving optimization problems with equality constraints. In those problems these algorithms can easily replace the traditional mutation operator with the proposed ECHT to handle the equality constraints efficiently.

- Many real world optimization problems may involve a wide range of uncertainties and dynamically changing environments. The objective function and/or constraint functions may be changed with time and conditions. These functions may also be noisy in some cases. The proposed algorithms can be extended on these types of problems. that are dynamic and noisy.
- This research considers single objective optimization problems. However, the application of evolutionary algorithms in multi-objective optimization is currently receiving growing interest from the researchers and practitioners. The proposed algorithms can be extended to solving multi-objective problems.

Appendix A

Test Problem Suite I

All benchmark functions (g1-g13) are described in (Runarsson and Yao, 2000). They are summarized here for completeness. The original sources of the functions are also cited.

a. g01 (Floudas and Pardalos, 1987)

Minimize

$$f(x) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$$

subject to

$$g_1(x) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \le 0,$$
$$g_2(x) = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \le 0,$$
$$g_3(x) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \le 0,$$
$$g_4(x) = -8x_1 + x_{10} \le 0,$$
$$g_5(x) = -8x_2 + x_{11} \le 0,$$
$$g_6(x) = -8x_3 + x_{12} \le 0,$$
$$g_7(x) = -2x_4 - x_5 + x_{10} \le 0,$$
$$g_8(x) = -2x_6 - x_7 + x_{11} \le 0,$$
$$g_9(x) = -2x_8 - x_9 + x_{12} \le 0,$$

where the bounds are $0 \le x_i \le 1\ (i = 1,...,9)$, $0 \le x_i \le 100\ (i = 10,11,12)$, and $0 \le x_{13} \le 1$.

The global minimum is at x^* = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1) where six constraints are active $(g_1, g_2, g_3, g_7, g_8,$ and $g_9)$ and $f(x^*) = -15.000$.

b. g02 (Koziel and Michalewicz, 1999)

Maximize

$$f(x) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=i}^{n} i x_i^2}} \right|$$

subject to

$$g_1(x) = 0.75 - \prod_{i=1}^{n} x_i \le 0,$$

$$g_2(x) = \sum_{i=1}^{n} x_i - 7.5n \le 0,$$

where $n = 20$ and $0 \le x_i \le 10\ (i = 1,...,n)$. The global maximum is unknown; However the best reported global minimum (to the best of our knowledge) reported at x^* = 3.61246061572185, 3.12833142812967, 3.09479212988791, 3.06145059523, 3.0279291588, 2.993826067, 2.958668717, 2.9218422731, 0.494825114566933, 0.48835711005490, 0.4823164271186, 0.476644750927, 0.47129550835, 0.46623099264167, 0.46142004984199, 0.45683664767, 0.45245876903267, 0.448267622418, 0.44424700958760, 0.44038285956317), where $f(x^*) = 0.8036191041\ 2559$. Constraint g_1 is close to being active $(g_1 = -10^{-8})$.

c. g03 (Michalewicz *et al.*, 1996)

Maximize

$$f(x) = (\sqrt{n})^n \prod_{i=1}^{n} x_i$$

subject to

$$h_1(x) = \sum_{i=1}^{n} x_i^2 - 1 = 0$$

where $n = 10$ and $0 \le x_i \le 1\ (i = 1,...,n)$. The global maximum is at $x^* = 1/\sqrt{n}\ (i = 1,...,n)$ where $f(x^*) = 1.000$.

d. g04 (Himmelblau, 1972)

Minimize

$$f(x) = 5.3578547\,x_3^2 + 0.8356891\,x_1x_5 + 37.293239\,x_1 - 40792.141$$

subject to

$$g_1(x) = 85.334407 + 0.0056858\,x_2x_5 + 0.0006262\,x_1x_4 - 0.0022053\,x_3x_5 - 92 \le 0,$$

$$g_2(x) = -85.334407 - 0.0056858\,x_2x_5 - 0.0006262\,x_1x_4 + 0.0022053\,x_3x_5 \le 0,$$

$$g_3(x) = 80.51249 + 0.0071317\,x_2x_5 + 0.0029955\,x_1x_2 + 0.0021813\,x_3^2 - 110 \le 0,$$

$$g_4(x) = -80.51249 - 0.0071317\,x_2x_5 - 0.0029955\,x_1x_2 - 0.0021813\,x_3^2 + 90 \le 0,$$

$$g_5(x) = 9.300961 + 0.0047026\,x_3x_5 + 0.0012547\,x_1x_3 + 0.0019085\,x_3x_4 - 25 \le 0,$$

$$g_6(x) = -9.300961 - 0.0047026\,x_3x_5 - 0.0012547\,x_1x_3 - 0.0019085\,x_3x_4 + 20 \le 0,$$

where $78 \le x_1 \le 102$, $33 \le x_2 \le 45$, and $27 \le x_i \le 45\ \ (i = 3,4,5)$.

The optimum solution is x^* = (78, 33, 29.995256025682, 45, 36.775812905788) where

$f(x^*) = -30665.539$. Two constraints are active (g_1 and g_6).

e. g05 (Hock and Schittkowski, 1981)

Minimize

$$f(x) = 3x_1 + 0.000001x_1^3 + 2x_2 + (0.000002/3)x_2^3$$

subject to

$$g_1(x) = -x_4 + x_3 - 0.55 \le 0,$$
$$g_2(x) = -x_3 + x_4 - 0.55 \le 0,$$
$$h_3(x) = 1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) + 894.8 - x_1 = 0,$$
$$h_4(x) = 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0,$$
$$h_4(x) = 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) + 1294.8 = 0,$$

where $0 \le x_1 \le 1200$, $0 \le x_2 \le 1200$, $-0.55 \le x_3 \le 0.55$, and $-0.55 \le x_4 \le 0.55$.

The best known solution is x^* = (679.9453, 1026.067, 0.1188764, -0.3962336) where $f(x^*) = 5126.4981$.

f. g06 (Floudas and Pardalos, 1987)

Minimize

$$f(x) = (x_1 - 10)^3 + (x_2 - 20)^3$$

subject to

$$g_1(x) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \le 0,$$
$$g_2(x) = (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \le 0,$$

where $13 \le x_1 \le 100$, and $0 \le x_2 \le 100$.

The optimum solution is x^* = (14.095, 0.84296) where $f(x^*) = -6961.81388$. Both constraints are active.

g. g07 (Hock and Schittkowski, 1981)

Minimize

$$f(x) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2$$
$$+2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45$$

subject to

$$g_1(x) = -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \le 0,$$
$$g_2(x) = 10x_1 - 8x_2 - 17x_7 + 2x_8 \le 0,$$
$$g_3(x) = -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \le 0,$$
$$g_4(x) = 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \le 0,$$
$$g_5(x) = 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \le 0,$$
$$g_6(x) = x_1^2 + 2(x_2 - 2)^2 - 2x_1x_2 + 14x_5 - 6x_6 \le 0,$$
$$g_7(x) = 0.5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 \le 0,$$
$$g_8(x) = -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \le 0,$$

where $-10 \le x_i \le 10$ $(i = 1,...,10)$.

The optimum solution is x^* = (2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927) where $f(x^*) = 24.3062091$. Six constraints are active $(g_1, g_2, g_3, g_4, g_5,$ and $g_6)$.

h. g08 (Koziel and Michalewicz, 1999)

Minimize

$$f(x) = \frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1 + x_2)}$$

subject to

$$g_1(x) = x_1^2 - x_2 + 1 \le 0,$$
$$g_2(x) = 1 - x_1 + (x_2 - 4)^2 \le 0,$$

where $0 \le x_1 \le 10$, and $0 \le x_2 \le 10$.

The optimum solution is x^* = (1.2279713, 4.2453733) where $f(x^*) = -0.095825$. The solution lies within the feasible region.

i. g09 (Hock and Schittkowski, 1981)

Minimize

$$f(x) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$

subject to

$$g_1(x) = -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \le 0,$$
$$g_2(x) = -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \le 0,$$
$$g_3(x) = -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \le 0,$$
$$g_4(x) = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \le 0,$$

where $-10 \le x_i \le 10$ $(i = 1,...,7)$.

The optimum solution is x^* = (2.330499, 1.951372, –0.4775414, 4.365726, –0.6244870, 1.038131, 1.594227) where $= f(x^*) = 680.6300573$. Two constraints are active $(g_1$ and $g_4)$.

j. g10 (Hock and Schittkowski, 1981)

Minimize

$$f(x) = x_1 + x_2 + x_3$$

subject to

$$g_1(x) = -1 + 0.0025(x_4 + x_6) \le 0,$$
$$g_2(x) = -1 + 0.0025(x_5 + x_7 - x_4) \le 0,$$
$$g_3(x) = -1 + 0.01(x_8 - x_5) \le 0,$$
$$g_4(x) = -x_1x_6 + 833.33252x_4 + 100x_1 - 83333.333 \le 0,$$
$$g_5(x) = -x_2x_7 + 1250x_5 + x_2x_4 - 1250x_4 \le 0,$$
$$g_6(x) = -x_3x_8 + 1250000 + x_3x_5 - 2500x_5 \le 0,$$

where $100 \le x_1 \le 10000$, $1000 \le x_i \le 10000$ $(i = 2,3)$ and $10 \le x_i \le 1000$ $(i = 4,...,8)$.

The optimum solution is x^* = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979) where $f(x^*) = 7049.3307$. Three constraints are active $(g_1, g_2$ and $g_3)$.

k. g11 (Koziel and Michalewicz, 1999)

Minimize

$$f(x) = x_1^2 + (x_2 - 1)^2$$

subject to

$$h(x) = x_2 - x_1^2 = 0,$$

where $-1 \le x_1 \le 1$ and $-1 \le x_2 \le 1$.

The optimum solution is $x^* = \left(\pm 1/\sqrt{2}, 1/2\right)$ where $f(x^*) = 0.75$.

l. g12 (Koziel and Michalewicz, 1999)

Maximize

$$f(x) = \left(100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2\right)/100$$

subject to

$$g(x) = (x_1 - p)^2 - (x_2 - q)^2 - (x_3 - r)^2 - 0.0625 \le 0$$

where $0 \le x_i \le 10$ $(i = 1,2,3)$ and $p, q, r = 1,2,...,9$.

The feasible region of the search space consists of 9^3 disjointed spheres. A point (x_1, x_2, x_3) is feasible if and only if there exist (p, q, r) such that the above inequality holds. The optimum is located at $x^* = (5,5,5)$ where $f(x^*) = 1$. The solution lies within the feasible region.

m. g13 (Hock and Schittkowski, 1981)

Minimize

$$f(x) = e^{x_1 x_2 x_3 x_4 x_5}$$

subject to

$$h_1(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0,$$

$$h_2(x) = x_2 x_3 - 5 x_4 x_5 = 0,$$

$$h_3(x) = x_1^3 + x_2^3 + 1 = 0,$$

where $-2.3 \le x_i \le 2.3$ (i = 1, 2) and $-3.2 \le x_i \le 3.2$ (i = 3, 4, 5).

The best known solution is at

$x^* = (-1.717142, 1.595721, 1.827250, -0.763659, -0.763659)$ where $f(x^*) = 0.0539498$.

Appendix B

Test Problem Suite II

a. B01 (Himmelblau, 1972)

Minimize

$$f(x) = \sum_{i=1}^{10} x_i \left(c_i + \ln \frac{x_i}{\sum_{j=1}^{10} x_j} \right)$$

Subject to

$$h_1(x) = x_1 + 2x_2 + 2x_3 + x_6 + x_{10} - 2 = 0,$$
$$h_2(x) = x_4 + 2x_5 + x_6 + x_7 - 1 = 0,$$
$$h_3(x) = x_3 + x_7 + x_8 + 2x_9 + x_{10} - 1 = 0,$$

where $0 \le x_i \le 10$ (i = 1,…,10) and $c_1 = -6.089$, $c_2 = -17.164$, $c_3 = -34.054$, $c_4 = -5.914$, $c_5 = -24.721$, $c_6 = -14.986$, $c_7 = -24.1$, $c_8 = -10.708$, $c_9 = -26.662$, $c_{10} = -22.179$.

The best known solution is at x^* = (0.0406684113, 0.147721240, 0.783205732, 0.0014143393, 0.4852936367, 0.0006931383, 0.0274052040, 0.0179509660, 0.0373268186, 0.0968844604) where $f(x^*) = -47.764888459$.

b. B02 (Himmelblau, 1972)

Minimize

$$f(x) = 1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1 x_2 - x_1 x_3$$

subject to

$$h_1(x) = x_1^2 + x_2^2 + x_3^2 - 25 = 0,$$
$$h_2(x) = 8x_1 + 14x_2 + 7x_3 - 56 = 0,$$

where the bounds are $0 \le x_i \le 10 (i = 1, 2, 3)$.

The best known solution is at

x^*=(3.5121281261, 0.2169875104, 3.5521785492), where $f(x^*) = 961.715022$.

c. B03 (Himmelblau, 1972)

Minimize

$$f(x) = 0.000117y_{14} + 0.1365 + 0.00002358y_{13} + 0.000001502y_{16} + 0.0321y_{12} + 0.004324y_5 + 0.0001\frac{c_{15}}{c_{16}} + 37.48\frac{y_2}{c_{12}} - 0.0000005843y_{17}$$

Subject to

$$g_1(x) = \frac{0.28}{0.72}y_5 - y_4 \le 0,$$
$$g_2(x) = x_3 - 1.5x_2 \le 0,$$
$$g_3(x) = 3496\frac{y_2}{c_{12}} - 21 \le 0,$$
$$g_4(x) = 110.6 + y_1 - \frac{62212}{c_{17}} \le 0,$$
$$g_5(x) = 213.1 - y_1 \le 0,$$
$$g_6(x) = y_1 - 405.23 \le 0,$$
$$g_7(x) = 17.505 - y_2 \le 0,$$
$$g_8(x) = y_2 - 1053.6667 \le 0,$$
$$g_9(x) = 11.275 - y_3 \le 0,$$
$$g_{10}(x) = y_3 - 35.03 \le 0,$$
$$g_{11}(x) = 214.228 - y_4 \le 0,$$
$$g_{12}(x) = y_4 - 665.585 \le 0,$$
$$g_{13}(x) = 7.458 - y_5 \le 0,$$
$$g_{14}(x) = y_5 - 584.463 \le 0,$$
$$g_{15}(x) = 0.961 - y_6 \le 0,$$
$$g_{16}(x) = y_6 - 265.916 \le 0,$$
$$g_{17}(x) = 1.612 - y_7 \le 0,$$
$$g_{18}(x) = y_7 - 7.046 \le 0,$$
$$g_{19}(x) = 0.146 - y_8 \le 0,$$
$$g_{20}(x) = y_8 - 0.222 \le 0,$$
$$g_{21}(x) = 107.99 - y_9 \le 0,$$
$$g_{22}(x) = y_9 - 273.366 \le 0,$$
$$g_{23}(x) = 922.693 - y_{10} \le 0,$$
$$g_{24}(x) = y_{10} - 1286.105 \le 0,$$
$$g_{25}(x) = 926.832 - y_{11} \le 0,$$
$$g_{26}(x) = y_{11} - 1444.046 \le 0,$$
$$g_{27}(x) = 18.766 - y_{12} \le 0,$$

$$g_{28}(x) = y_{12} - 537.141 \le 0,$$
$$g_{29}(x) = 1072.163 - y_{13} \le 0,$$
$$g_{30}(x) = y_{13} - 3247.039 \le 0,$$
$$g_{31}(x) = 8961.448 - y_{14} \le 0,$$
$$g_{32}(x) = y_{14} - 26844.086 \le 0,$$
$$g_{33}(x) = 0.063 - y_{15} \le 0,$$
$$g_{34}(x) = y_{15} - 0.386 \le 0,$$
$$g_{35}(x) = 7108433 - y_{16} \le 0,$$
$$g_{36}(x) = -140000 + y_{16} \le 0,$$
$$g_{37}(x) = 2802713 - y_{17} \le 0,$$
$$g_{38}(x) = y_{17} - 12146108 \le 0,$$

where

$$y_1 = x_2 + x_3 + 41.6,$$
$$c_1 = 0.024x_4 - 4.62,$$
$$y_2 = \frac{12.5}{c_1} + 12,$$
$$c_2 = 0.0003535x_1^2 + 0.5311x_1 + 0.08705y_2x_1,$$
$$c_3 = 0.052x_1 + 78 + 0.002377y_2x_1,$$
$$y_3 = \frac{c_2}{c_3},$$
$$y_4 = 19y_3,$$
$$c_4 = 0.04782(x_1 - y_3) + \frac{0.1956(x_1 - y_3)^2}{x_2} + 0.6376y_4 + 1.594y_3,$$
$$c_5 = 100x_2,$$
$$c_6 = x_1 - y_3 - y_4,$$
$$c_7 = 0.950 - \frac{c_4}{c_5}$$
$$y_5 = c_6c_7,$$
$$y_6 = x_1 - y_5 - y_4 - y_3,$$
$$c_8 = (y_5 + y_4)0.995,$$
$$y_7 = c_8 y_1,$$
$$y_8 = \frac{c_8}{3798},$$
$$c_9 = y_7 - \frac{0.0663y_7}{y_8} - 0.3153,$$

$$y_9 = \frac{96.82}{c_9} + 0.321 y_1 \ ,$$
$$y_{10} = 1.29 y_5 + 1.258 y_4 + 2.29 y_3 + 1.71 y_6 ,$$
$$y_{11} = 1.71 x_1 - 0.452 y_4 + 0.580 y_3 ,$$
$$c_{10} = \frac{12.3}{752.3} \ ,$$
$$c_{11} = (1.75 y_2)(0.995 x_1) ,$$
$$c_{12} = 0.995 y_{10} + 1998 ,$$
$$y_{12} = c_{10} x_1 + \frac{c_{11}}{c_{12}} ,$$
$$y_{13} = c_{12} - 1.75 y_2 ,$$
$$y_{14} = 3623 + 64.4 x_2 + 58.4 x_3 + \frac{146312}{y_9 + x_5} ,$$
$$c_{13} = 0.995 y_{10} + 60.8 x_2 + 48 x_4 - 0.1121 y_{14} - 5095 ,$$
$$y_{15} = \frac{y_{13}}{c_{13}} ,$$
$$y_{16} = 148000 - 331000 y_{15} + 40 y_{13} - 61 y_{15} y_{13} ,$$
$$c_{14} = 2324 y10 - 28740000 y_2 ,$$
$$y_{17} = 14130000 - 1328 y_{10} - 531 y_{11} + \frac{c_{14}}{c_{12}} ,$$
$$c_{15} = \frac{y_{13}}{y_{15}} - \frac{y_{13}}{0.52} ,$$
$$c_{16} = 1.104 - 0.72 y_{15} ,$$
$$c_{17} = y_9 + x_5 ,$$

where the bounds are $704.4148 \le x_1 \le 906.3855$, $68.6 \le x_2 \le 288.88$, $0 \le x_3 \le 134.75$, $193 \le x_4 \le 287.0966$, and $25 \le x_5 \le 84.1988$.

The best known solution is at x^* = (705.1745370700, 68.599999999, 102.8999999999, 282.3249315936, 37.5841164258) where $f(x^*) = -1.9051552585$.

d. B04 (Himmelblau, 1972)

Minimize

$$f(x) = -0.5(x_1 x_4 - x_2 x_3 + x_3 x_9 - x_5 x_9 + x_5 x_8 - x_6 x_7)$$

subject to

$$g_1(x) = x_3^2 + x_4^2 - 1 \le 0 ,$$
$$g_2(x) = x_9^2 - 1 \le 0 ,$$
$$g_3(x) = x_5^2 + x_6^2 - 1 \le 0 ,$$

$$g_4(x) = x_1^2 + (x_2 - x_9)^2 \le 0,$$
$$g_5(x) = (x_1 - x_5)^2 + (x_2 - x_6)^2 - 1 \le 0,$$
$$g_6(x) = (x_1 - x_7)^2 + (x_2 - x_8)^2 - 1 \le 0,$$
$$g_7(x) = (x_3 - x_5)^2 + (x_4 - x_6)^2 - 1 \le 0,$$
$$g_8(x) = (x_3 - x_7)^2 + (x_4 - x_8)^2 - 1 \le 0,$$
$$g_9(x) = x_7^2 + (x_8 - x_9)^2 - 1 \le 0,$$
$$g_{10}(x) = x_2 x_3 + x_1 x_4 \le 0,$$
$$g_{11}(x) = -x_3 x_9 \le 0,$$
$$g_{12}(x) = x_5 x_9 \le 0,$$
$$g_{13}(x) = x_6 x_7 - x_5 x_8 \le 0,$$

where the bounds are $-10 \le x_i \le 10$ $(i = 1,...,8)$ and $0 \le x_9 \le 20$.

The best known solution is at $x^* = (-0.6577761924, -0.1534187734, 0.3234138716, -0.9462576116, -0.6577761943, -0.7532134346, 0.3234138741, -0.3464629479, 0.5997946628)$ where $f(x^*) = -0.8660254037$.

e. **B05** (Floudas, 1999)

Minimize

$$f(x) = -x_1 - x_2$$

subject to

$$g_1(x) = -2x_1^4 + 8x_1^3 - 8x_1^2 + x_2 - 2 \le 0,$$
$$g_2(x) = -4x_1^4 + 32x_1^3 - 88x_1^2 + 96x_1 + x_2 - 36 \le 0,$$

where the bounds are $0 \le x_1 \le 3$ and $0 \le x_2 \le 4$.

The feasible global minimum is at $x^* = (2.3295201974, 3.1784930741)$.

where $f(x^*) = -5.50801327156$. This problem has a feasible region consisting on two disconnected sub-regions.

f. **B06** (Himmelblau, 1972)

Minimize

$$f(x) = x_1$$

subject to

$$g_1(x) = -x_1 + 35x_2^{0.6} + 35x_3^{0.6} - 25 \le 0,$$
$$h_1(x) = -300x_3 + 7500x_5 - 7500x_6 - 25x_4x_5 + 25x_4x_6 + x_3x_4 = 0,$$

$$h_2(x) = 100x_2 + 155.365x_4 + 2500x_7 - x_2x_4 - 25x_4x_7 - 15536.5 = 0,$$
$$h_3(x) = -x_5 + ln(-x_2 + 900) = 0,$$
$$h_4(x) = -x_6 + ln(-x_4 + 300) = 0,$$
$$h_5(x) = -x_7 + ln(-2x_4 + 700) = 0,$$

where the bounds are $0 \le x_1 \le 1000$, $0 \le x_2, x_3 \le 40$, $100 \le x_4 \le 300$, $6.3 \le x_5 \le 6.7$, $5.9 \le x_6 \le 6.4$, $4.5 \le x_7 \le 6.25$.

The best known solution is at x^*= 193.7245100700, 5.5694413155E-27, 17.3191887294, 100.0478978013, 6.6844518536, 5.9916842844, 6.2145164888; where $f(x^*) = 193.724510070$.

g. B07 (Hock and Schittkowski, 1981)

Minimize

$$f(x) = (x_1 - 1)^2 + (x_2 - x_3)^2 + (x_4 - x_5)^2$$

Subject to

$$h_1(x) = x_1 + x_2 + x_3 + x_4 + x_5 - 5 = 0,$$
$$h_2(x) = x_3 + 2(x_4 + x_5) + 3 = 0,$$

where $0.0 \le x_i \le 2.0$ $(i = 1,\ldots,5)$.

The optimum solution is $x^* = (1.000, 1.000, 1.000, 1.000, 1.000)$ where $f(x^*) = 0.0000$.

h. B08 (Hock and Schittkowski, 1981)

Minimize

$$f(x) = x_1 + 2x_1 + 4x_5 + e^{x_1x_4}$$

Subject to

$$h_1(x) = x_1 + 2x_2 + 5x_5 - 6 = 0,$$
$$h_2(x) = x_1 + x_2 + x_3 - 3 = 0,$$
$$h_3(x) = x_4 + x_5 + x_6 - 2 = 0,$$
$$h_4(x) = x_1 + x_4 - 1 = 0,$$
$$h_5(x) = x_2 + x_5 - 2 = 0,$$
$$h_6(x) = x_3 + x_6 - 2 = 0,$$

where $0.0 \le x_i \le 2.0$ $(i = 1,\ldots,6)$.

The optimum solution is

$x^* = (0.000000, 1.333334, 1.666667, 1.000000, 0.666667, 0.333334)$ where $f(x^*) = 6.3334$.

i. **B09** (Hock and Schittkowski, 1981)

Minimize

$$f(x) = -x_1$$

Subject to

$$h_1(x) = x_2 - x_1^3 - x_3^2 = 0\,,$$

$$h_2(x) = x_1^2 - x_2 - x_4^2 = 0\,,$$

where $0.0 \le x_i \le 2.0$ ($i = 1,\ldots,4$).

The optimum solution is $x^* = (1.000000, 1.000000, 0.000000, 0.000000)$ where $f(x^*) = -1.000000$.

Appendix C

Test Problem Suite III

Crop Planning Model

a. A Linear Crop Planning Model (Sarker and Ray, 2005; Sarker and Ray, 2009)

Index:

i for a crop which can be considered for production

j a crop combination made up from i

k land type

Set:

CE set of crops that can be imported

CAL set of crops having area limitation

CIL set of crops having import limitation

Parameters:

n_1 number of alternative crops for single-cropped land

n_2 number of crop combinations for double-cropped land

n_3 number of crop combinations for triple-cropped land

$N1_j$ a crop in each j for single-cropped land, $j = 1, \ldots, n_1$

$N2_j$ the j-th crop pair of the possible crop combinations of double-cropped land, $j = 1, \ldots, n_2$

$N3_j$ the j-th crop triple of the possible crop combinations of triple-cropped land, $j = 1, \ldots, n_3$

YR_{ijk} yield rate that is the amount of production per unit area for crop i of crop combination j in land type k.

CP_{ijk} variable cost required per unit area for crop i of crop combination j in land type k.

P_i market price of crop i per metric ton

B_{ijk} gross margin that is the benefit that can be obtained per unit area of land from crop i of crop combination j in land type $k = (P_i * YR_{ijk} - CP_{ijk})$

IC_i gross margin from import of crop i (=Market revenue - Import cost)

D_i yearly demand of crop i

L_k available area of land type k

LT_k land type coefficient for land type k (=1, 1/2 or 1/3)

C_a working capital available, this indicates the total amount of money that can be used for covering variable costs.

A area suitable and available for crop i when $k = 1$

IL upper limit of total crop import

Variables

X_{ijk} Area of land to be cultivated for crop i of crop combination j in land type k.

I_i Amount of crop i that should be imported.

Objective function 1: The first objective is to maximize the total gross margin (from cultivated plus imported crops) that can be obtained from cropping in a single crop year.

$$\text{Maximize} \quad Z_1 = \sum_{j=1}^{n_1} \sum_{i \in N1_j} B_{ij(k=1)} X_{ij(k=1)} + \sum_{j=1}^{n_2} \sum_{i \in N2_j} B_{ij(k=2)} X_{ij(k=2)} + \sum_{j=1}^{n_3} \sum_{i \in N3_j} B_{ij(k=3)} X_{ij(k=3)} + \sum_{i \in CE} IC_i I_i \quad (1)$$

The first, second, third and fourth terms represent the gross margin from single crop land, double crop land, triple crop land and imported crop respectively. Note that there is only one crop for each j in single crop land, two crops in doubled crop land and three crops in triple crop land.

Constraints:

Demand constraint: The sum of local production and the imported quantity of crop i in a

year must be greater than or equal to the total requirements in the country.

$$\sum_j \sum_k YR_{ijk} X_{ijk} + I_{i \in CE} \geq D_i \qquad \forall i \tag{3}$$

Land constraint: The total land used for a given type of land must be less than or equal to the total available land of that type.

$$\sum_i \sum_j LT_k X_{ijk} \leq L_k \qquad \forall k \tag{4}$$

Here, for k = 1, 2 and 3, the coefficients (LT_k) are 1, ½ and 1/3 respectively. If a piece of land is used by two crops (in a double cropped land) one after another (consecutive production) in a given year, it is assumed equivalent to the use of half the land for one of these two crops in a year - that is LT_k= ½. This assumption makes the constraint (4) simpler and it is required only for land constraint.

Capital constraint: The total amount of money that can be spent for covering the variable costs in crop production must be less than or equal to the working capital available. Note that minimization of capital requirements is one of our two objectives formulated above. This additional constraint basically sets the upper bound of capital availability.

$$\sum_{j=1}^{n_1} \sum_{i \in N1_j} CP_{ij(k=1)} X_{ij(k=1)} + \sum_{j=1}^{n_2} \sum_{i \in N2_j} CP_{ij(k=2)} X_{ij(k=2)} + \sum_{j=1}^{n_3} \sum_{i \in N3_j} CP_{ij(k=3)} X_{ij(k=3)} \leq C_a \tag{5}$$

Contingent constraint: The area used for any crop under a crop combination for double- or triple-cropped land must be equal for every crop. For example, in a double-cropped land, the area used by two crops belonging to any crop combination must be equal.

$$X_{(i_1 \in N2_j) j(k=2)} - X_{(i_2 \in N2_j) j(k=2)} = 0 \qquad \forall j \tag{6}$$

In double cropped land, for a given crop combination j there is only two crops: i_1 and i_2 where i_1 is the first crop and i_2 is the second crop in the combination. Both crops use the same area of land but one after another. In a triple-cropped land, the area used by three crops belonging to any crop combination must be equal.

$$X_{(i_1 \in N3_j) j(k=3)} - X_{(i_2 \in N3_j) j(k=3)} = 0 \qquad \forall j \tag{7}$$

$$X_{(i_2 \in N3_j) j(k=3)} - X_{(i_3 \in N3_j) j(k=3)} = 0 \qquad \forall j \qquad (8)$$

Here, i_1 is the first crop, i_2 is the second crop and i_3 is the third crop for combination j.

Area and import bound constraint: Due to soil characteristics and regional aspects, in some regions, the amount of area to be used for certain crops is restricted. For example, the unsuitability of certain lands for fruit cultivation needs to set an area limit for fruit. This is true only for single-cropped land. Similarly, a constraint needs to be set for import restriction as there is an upper limit on the importation of some crops.

$$\textit{Area bound: } \sum_{i \in CAL} X_{ijk} \le A \qquad \forall j = 1, k = 1 \qquad (9)$$

$$\textit{Import bound: } \sum_{i \in CIL} I_i \le IL \qquad (10)$$

Non-negativity constraint: The decision variables must be greater than or equal to zero.

$$X_{ijk} \ge 0 \qquad \forall i,j,k \quad \text{and} \qquad (11)$$

$$I_i \ge 0 \qquad \forall i$$

b. A Nonlinear Crop Planning Model (Sarker and Ray, 2005; Sarker and Ray, 2009)

It is interesting that, for a given crop, the yield rate in double- and triple- cropped land is little higher than the single-cropped land. This is due to frequent use of fertilizers and insecticides in double- and triple-cropped land. The difference is significant for triple-cropped land and a nonlinear relationship is established to reflect this change. The change is related to the triple crop decision variables by expressing as $(x_{ijk})^b$. The value of b varies from situation to situation. For higher yield rate, the value of b will be more than one. In situations, where fertilizers are not appropriately used, the yield rate may decrease for double- and triple-cropped lands. So the value of b is assumed that it will be less than one. In addition, the nonlinearity may arise due to soil characteristics and the level of agricultural inputs used. In the model, the triple crop variables for the first objective function and demand constraints will be assumed as nonlinear.

References

Adler, M. R., Davis, A. B., Weihmayer, R. and Worrest, R. W. (1989). Conflict resolution strategies for nonhierarchical distributed agents. In *Distributed Artificial Intelligence*, vol. 2, pages 139–16. Pitman Publishing: London and Morgan Kaufmann, San Mateo, CA.

Ahn, H. and Kim, K. J. (2009). Bankruptcy prediction modeling with hybrid case-based reasoning and genetic algorithms approach. *Applied Soft Computing*, **9**(2): 599-607.

Alba, E. and Dorronsoro, B. (2005). The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. *IEEE Transactions on Evolutionary Computation* **9**(2): 126-142.

Alkan, A. and Ozcan, E. (2003). Memetic algorithms for timetabling. In *The 2003 Congress on Evolutionary Computation*, Vol. 3, pp. 1796-1802

Alkemade, F., Van Bragt, D. D. B. and La Poutre, J. A. (2005). Stabilization of tag-mediated interaction by sexual reproduction in an evolutionary agent system. *Information Sciences*, **170**(1): 101-119.

Alyabysheva, T., Parfenov, L., Rudnev, A., Stepanov, V. and Tsvetkov, E. (1975). Optimizing the operating schedules of hydroelectric stations in power systems. *Power Technology and Engineering (formerly Hydrotechnical Construction)*, **9**(9): 818-822.

Anderson, D. R., Sweeney, D. J., Williams, T. A., Harrison, N. J. and Rickard, J. A. (1996). *Essentials of Statistics for Business and Economics* Harper educational (Australia) Pty. Limited.

Andreou, A. S., Georgopoulos, E. F. and Likothanassis, S. D. (2002). Exchange-Rates Forecasting: A Hybrid Algorithm Based on Genetically Optimized Adaptive Neural Networks. *Computational Economics*, **20**(3): 191-210.

Bäck, T., Fogel, D. B. and Michalewicz, Z., Eds. (2000). *Handbook of Evolutionary Computation*. Institute of Physics Publishing Ltd, Bristol and Philadelphia, UK.

Badiru, A. B. and Pulat, P. S. (1994). *Comprehensive Project Management: Integrating Optimization Models, Management Principles, and Computers*. Prentice Hall PTR.

Bajo, J. and Corchado, J. (2006). Multiagent Architecture for Monitoring the North-Atlantic Carbon Dioxide Exchange Rate. In *Current Topics in Artificial Intelligence*, pages 321-330. Springer, Berlin.

Bazaraa, M., Jarvis, J. and Sherali, H. (1990). *Linear Programming and Network Flows*. John Wiley & Sons, New York.

Bazaraa, M., Sherali, H. and Shetty., C. (2006). *Nonlinear Programming: Theory and Algorithms*. 3 ed., A John Wiley & Sons, Inc, New York.

Bean, J. C. and Hadj-Alouane, A. B. (1993). A dual genetic algorithm for bounded integer programs. Technical Report, Department of Industrial and Operations Engineering, The University of Michigan.

Ben Hadj-Alouane, A. and Bean, J. C. (1997). A Genetic Algorithm for the Multiple-Choice Integer Program. *Operations Research*, **45**(1): 92-101.

Bentall, M., Turton, B. C. H. and Hobbs, C. W. L. (1997). Benchmarking the restoration of heavily loaded networks using a two dimensional order-based genetic algorithm. In *Second International Conference On Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA 97*, pp. 151-156.

Bertsekas, D. (1995). *Nonlinear Programming*. Athens Scientific, Belmont.

Bingul, Z., Sekmen, A. S., Palaniappan, S. and Zein-Sabatto, S. (2000). Genetic algorithms applied to real time multiobjective optimization problems. In *Proceedings of the IEEE Southeastcon 2000*, pp. 95-103.

Buckley, M. J. (1996). Linear array synthesis using a hybrid genetic algorithm. In *Antennas and Propagation Society International Symposium, 1996. AP-S. Digest*, Vol. 1, pp. 584-587.

Burke, E. K., Dave, E. and Rupert, F. W. (1995). A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pp. 605-610, Morgan Kaufmann Publishers

Inc.

Burke, E. K. and Newall, J. P. (1999). A multistage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, **3**(1): 63-74.

Burke, E. K., Newall, J. P. and Weare, R. F. (1996). A Memetic Algorithm for University Exam Timetabling. In *Selected papers from the First International Conference on Practice and Theory of Automated Timetabling*, pp. 241 - 250, Springer-Verlag.

Burke, E. K. and Smith, A. J. (1999). A memetic algorithm to schedule planned maintenance for the national grid. *Journal of Experimental Algorithmics* **4**(6): Article No. 1 (1-13).

Burmeister, B., Haddadi, A. and Matylis, G. (1997). Application of multi-agent systems in traffic and transportation. In *IEE Proceedings Software Engineering*, Vol. 144, pp. 51-60.

Byrski, A. and Schaefer, R. (2009). Formal Model for Agent-Based Asynchronous Evolutionary Computation. In *IEEE Congress on Evolutionary Computation (CEC) 2009*, Norway, pp. 78-85.

Caglayan, A., Snorrason, M., Mazzu, J., Jacoby, J., Jones, R. and Kumar, K. (1997). Open sesame – a learning agent engine. *Applied Artificial Intelligence*, **11**(5): 393–412.

Cai, Z. and Wang, Y. (2006). A Multiobjective Optimization-Based Evolutionary Algorithm for Constrained Optimization. *IEEE Transactions on Evolutionary Computation*, **10**(6): 658-675.

Caldwell, C. and Johnston, V. S. (1991). Tracking a criminal suspect through face-space with a genetic algorithm. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 416-421.

Carlson, S. E. and Shonkwiler, R. (1998). Annealing a genetic algorithm over constraints. In *1998 IEEE International Conference on Systems, Man, and Cybernetics, 1998*, Vol. 4, pp. 3931-3936.

Cetnarowicz, K., Kisiel-Dorohinicki, M. and Nawarecki, E. (1996). The Application of Evolution Process in Multi-Agent World to the Prediction System. In *Proceedings of the 2nd International Conference on Multi-Agent Systems (ICMAS'96)*, pp. 26-32, AAAI Press.

Chaib-draa, B. (1995). Industrial applications of distributed AI. *Communications of the ACM*, **38**(11): 49-53.

Chavez, A. and Kasbah, P. M. (1996). An agent marketplace for buying and selling goods. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*, London, UK, pp. 75–90.

Chen, L. and Sycara, K. (1998). Webmate : A personal agent for browsing and searching. In *Proceedings of the Second International Conference on Autonomous Agents (Agents 98)*, Minneapolis/St Paul, MN, pp. 132 - 139.

Chen, Y. L., Liao, W. B., Yang, Y. R., Shen, K. Y., Wang, S. C. and Chang, Y. C. (2002). The interactive surrogate worth trade-off method for multi-objective decision-making in reactive power sources planning. In *International Conference on Power System Technology, 2002*, Vol. 2, pp. 863-866.

Cheng, R. and Gen, M. (1996). Parallel machine scheduling problems using memetic algorithms. In *IEEE International Conference on Systems, Man, and Cybernetics, 1996*, Vol. 4, pp. 2665-2670.

Chira, C., Gog, A. and Dumitrescu, D. (2008). Exploring population geometry and multi-agent systems: a new approach to developing evolutionary techniques. In *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, Atlanta, GA, USA, pp. 1953-1960, ACM.

Choi, S. P. M., Liu, J. and Chan, S. P. (2001). A genetic agent-based negotiation system. *Computer Networks*, **37**(2): 195 - 204.

Chootinan, P. and Chen, A. (2006). Constraint handling in genetic algorithms using a gradient-based repair method. *Computers & Operations Research*, **33**(8): 2263-2281.

Chryssolouris, G. and Subramaniam, V. (2001). Dynamic scheduling of manufacturing job shops using genetic algorithms. *Journal of Intelligent Manufacturing*, **12**(3): 281-293.

Clearwater, S. H., Costanza, R., Dixon, M. and Schroeder, B. (1996). Saving energy using market-based control. In Clearwater, S. H., editor, *Market Based Control*, pages 253–273. World Scientific, Singapore.

Coello, C. A. C. (2000a). Constraint-handling using an evolutionary multiobjective optimization technique. *Civil Engineering and Environmental Systems* **17**: 319–346.

Coello, C. A. C. (2000b). Treating constraints as objectives for single-objective evolutionary optimization. *Engineering Optimization*, **32**(3): 275–308.

Coello, C. A. C. (2000c). Use of a self-adaptive penalty approach for engineering optimization problems. *Computers in Industry*, **41**(2): 113-127.

Coello, C. A. C. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, **191**(11-12): 1245-1287.

Coello, C. A. C. and Mezura-Montes, E. (2002). Handling Constraints in Genetic Algorithms using Dominance-Based Tournaments. In *Proceeding of the Fifth International Conference on Adaptive Computing Design and Manufacture (ACDM 2002)*, Devon, UK, Vol. 5, pp. 273-284, Springer-Verlag.

Coello, C. A. C., Van Veldhuizen, D. A. and Lamont, G. B. (2002). *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York.

Coello Coello, C. A. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, **191**(11-12): 1245-1287.

Coit, D. W., Smith, A. E. and Tate, D. M. (1996). Adaptive Penalty Methods for Genetic Optimization of Constrained Combinatorial Problems. *Informs Journal on Computing*, **8**(2): 173-182.

Conover, W. J. (1980). *Practical Nonparametric Statistics*. John Wiley & Sons.

Corne, D. W., Ross, P. and Fang, H.-L. (1994). Fast practical evolutionary timetabling. In *Evolutionary Computing*, pages 250-263. Springer, Berlin / Heidelberg.

Cortés, U., Annicchiarico, R. and Urdiales, C. (2008). Agents and Healthcare: Usability and Acceptance. In Annicchiarico, R., Cortés, U. and Urdiales, C., editor, *Agent Technology and e-Health*, pages 1-4. Birkhäuser Basel.

Cox, L. A. J., Davis, L. and Qiu, Y. (1991). Dynamic anticipatory routing in circuit-switched telecommunications networks. In Davis, L., editor, *Handbook of Genetic*

Algorithms, pages 124-143. Van Nostrand Reinhold.

Crossley, W. A. and Williams, E. A. (1997). A study of adaptive penalty functions for constrained genetic algorithm based optimization. In *AIAA 35th Aerospace Sciences Meeting and Exhibit, AIAA-1997-83*, Reno, Nevada.

Cutkosky, M. R., Engelmore, R. S., Fikes, R. E., Genesereth, M. R., Gruber, T. R., Mark, W. S., Tenenbaum, J. M. and Weber, J. C. (1993). PACT: an experiment in integrating concurrent engineering systems. *Computer*, **26**(1): 28-37.

Dantzig, G. B. and Thapa, M. N. (2003). *Linear Programming: Theory and extensions*. vol. 2, Springer.

Darr, T. P. and Birmingham, W. P. (1994). Automated design for concurrent engineering. *IEEE Expert*, **9**(5): 35-42.

Darwin, C. R. (1859). *The Origin of Species: By Means of Natural Selection or the Preservation of Favoured Races in the Struggle for Life*. John Murray, London.

Dasgupta, D. and Michalewicz, Z., Eds. (1997). *Evolutionary Algorithms in Engineering Applications*. Springer, Berlin.

Davidor, Y. (1991). A genetic algorithm applied to robot trajectory generation. In Davis, L., editor, *Handbook of Genetic Algorithms*, pages 144–165. Van Nostrand Reinhold, New York.

Davidsson, P., Persson, J. and Holmgren, J. (2007). On the Integration of Agent-Based and Mathematical Optimization Techniques. In *Agent and Multi-Agent Systems: Technologies and Applications*, pages 1-10. Springer-Verlag, Berlin / Heidelberg.

Davis, L. (1991). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York.

Davis, L., Orvosh, D., Cox, A. and Qiu, Y. (1993). A Genetic Algorithm for Survivable Network Design. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 408 - 415, Morgan Kaufmann Publishers Inc.

Dawkins, R. (1976). *The selfish gene*. Oxford University Press, New York.

De Jong, K. A. (2008). Evolving intelligent agents: A 50 year quest. *Computational Intelligence Magazine, IEEE*, **3**(1): 12-17.

Deb, K. (1999). An Introduction to Genetic Algorithms. *Sadhana*, **24**(4): 293-315.

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

Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc.

Deb, K. and Agrawal, R. B. (1995). Simulated Binary Crossover for Continuous Search Space. *Complex Systems*, **9**: 115-148.

Deb, K. and Beyer, H.-g. (2001). Self-Adaptive Genetic Algorithms with Simulated Binary Crossover. *Evolutionary Computation*, **9**(2): 197-221.

Deb, K. and Goldberg, D. E. (1989). An investigation of niche and species formation in genetic function optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, George Mason University, CA, pp. 42–50.

Deb, K. and Goyal, M. (1996). A combined genetic adaptive search (GeneAS) for engineering design. *Computer Science and Informatics*, **26**(4): 30-45.

Deb, K., Padhye, N. and Neema, G. (2007). Interplanetary Trajectory Optimization with Swing-Bys Using Evolutionary Multi-objective Optimization. In *Advances in Computation and Intelligence*, pages 26-35. Springer, Berlin / Heidelberg.

Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, **6**(2): 182.

Dobrowolski, G., Kisiel-Dorohinicki, M. and Nawarecki, E. (2001). Evolutionary multiagent system in multiobjective optimisation. In *Proceedings of the IASTEDI nternational Symposium: Applied Informatics*, IASTED/ACTA Press.

Dreżewski, R. and Kisiel-Dorohinicki, M. (2006). Maintaining Diversity in Agent-Based Evolutionary Computation. In *Computational Science – ICCS 2006*, pages 908-911.

Eiben, A. E. and Van Der Hauw Hauw, J. K. (1998). Adaptive penalties for evolutionary graph coloring. In *Artificial Evolution*, pages 95-106. Springer, Berlin / Heidelberg.

Eiselt, H. A., Pederzoli, G. and Sandblom, C. L. (1987). *Continuous Optimization Models*. Walter de Gruyter, Berlin.

Elfeky, E. Z., Sarker, R. A. and Essam, D. L. (2006). A Simple Ranking and Selection for Constrained Evolutionary Optimization. In *Simulated Evolution and Learning*, Lecture Notes in Computer Science, Vol. 4247, pages 537-544. Springer-Verlag,

Berlin / Heidelberg.

Elfeky, E. Z., Sarker, R. A. and Essam, D. L. (2008). Analyzing the simple ranking and selection process for constrained evolutionary optimization. *Journal of Computer Science And Technology* **23** (1): 19-34.

Farmani, R. and Wright, J. A. (2003). Self-adaptive fitness formulation for constrained optimization. *IEEE Transactions on Evolutionary Computation*, **7**(5): 445.

Ferber, J. (1999). *Multiagent systems as introduction to distributed artificial intelligence*. Addision-Wesley.

Ferrolho, A., Crisostomo, M. and Wojcik, R. (2007). Job shop scheduling problems with Genetic Algorithms. In *International Conference on Computer Engineering & Systems, 2007. ICCES '07*, pp. 76-80.

Fletcher, R. (1990). *Practical Methods of Optimization*. 2 ed., Wiley, New York.

Floudas, C. (1999). *Handbook of Test Problems in Local and Global Optimization. Nonconvex Optimization and its Applications*. Kluwer Academic Publishers, The Netherlands.

Floudas, C. A. and Pardalos, P. M. (1987). *A Collection of Test Problems for Constrained Global Optimization*. vol. 455, Series: Lecture Notes in Computar Science, Springer-Verlag, Berlin, Germany.

Folino, G., Pizzuti, C. and Spezzano, G. (2001). Parallel hybrid method for SAT that couples genetic algorithms and local search. *IEEE Transactions on Evolutionary Computation*, **5**(4): 323-334.

Fonseca, C. M. and Fleming, P. J. (1993). Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, California University of Illinois at Urbana-Champaign, pp. 416-423, Morgan Kauffman Publishers.

Forrest, S., Javornik, B., Smith, R. E. and Perelson, A. S. (1993). Using Genetic Algorithms to Explore Pattern Recognition in the Immune System. *Evolutionary Computation*, **1**(3): 191-211.

Francois, F. I., Michael, P. G. and Anand, S. R. (1992). An architecture for Real-Time Reasoning and System Control. *IEEE Expert: Intelligent Systems and Their*

Applications, **7**(6): 34-44.

Gass, S. and Saaty, T. (2006). The computational algorithm for the parametric objective function. *Naval Research Logistics Quarterly*, **2**(1-2): 39-45.

Gen, M., Tsujimura, Y. and Kubota, E. (1994). Solving job-shop scheduling problems by genetic algorithm. In *1994 IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 2, pp. 1577-1582.

Gendreau, M. (2003). An Introduction to Tabu Search. In Glover, F. and Kochenberger, G. A., editor, *Handbook of Metaheuristics*, pages 37-54. Springer.

Geoffrion, A. M., Dyer, J. S. and Feinberg, A. (1972). An Interactive Approach for Multi-Criterion Optimization, with an Application to the Operation of an Academic Department. *Management Science*, **19**(4): 357-368.

Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers, London.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Goldberg, D. E. and Deb, K. (1991). A comparison of selection schemes used in genetic algorithms. In Rawlins, G. J. E., editor, *Foundations of Genetic Algorithms*, pages 69–93.

Goldberg, D. E. and R. Lingle, J. (1985). Alleles, Loci, and Traveling Salesman Problem. In *Proceedings of 1st International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, pp. 154-159.

Goldberg, D. E. and Voessner, S. (1999). Optimizing Global-Local Search Hybrids. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 220–228.

Gonzalez, T. F. (2007). *Handbook of Approximation Algorithms and Metaheuristics* Chapman & Hall/Crc Computer and Information Science Series.

Grefenstette, J. J., Gopal, R., Rosmaita, B. and VanGucht, D. (1985). Genetic algorithms for the traveling salesman problem In *Proceedings International Conference Genetic Algorithms and Their Applicaions* pp. 160–168.

Griffeth, N. D. and H.Velthuijsen (1994). The negotiating agents approach to run-time feature interaction resolution. In Bouma, L. G. and Velthuijsen, H., editor, *Feature Interactions in Telecommunications Systems*, pages 217–235. IOS Press.

Grimbleby, J. B. (1995). Automatic analogue network synthesis using genetic algorithms. In *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, 1995. GALESIA*, pp. 53-58.

Guimaraes, F. G., Wanner, E. F., Campelo, F., Takahashi, R. H. C., Igarashi, H., Lowther, D. A. and Ramirez, J. A. (2006). Local Learning and Search in Memetic Algorithms. In *IEEE Congress on Evolutionary Computation, CEC 2006*, pp. 2936-2943.

Gupta, H. and Deb, K. (2005). Handling Constraints In Robust Multi-Objective Optimization. In *The 2005 IEEE Congress on Evolutionary Computation, 2005*, Edinburgh, Scotland, Vol. 1, pp. 25-32.

Haimes, Y. Y., Lasdon, L. S. and Wismer, D. A. (1971). On a Bicriterion Formulation of the Problems of Integrated System Identification and System Optimization. *IEEE Transactions on Systems, Man and Cybernetics*, **1**(3): 296-297.

Handa, H., Chapman, L. and Xin, Y. (2006). Robust route optimization for gritting/salting trucks: a CERCIA experience. *IEEE Computational Intelligence Magazine*, **1**(1): 6-9.

Hanjie, C. and Baldick, R. (2007). Optimizing Short-Term Natural Gas Supply Portfolio for Electric Utility Companies. *IEEE Transactions on Power Systems*, **22**(1): 232-239.

Hanshar, F. T. and Ombuki-Berman, B. M. (2007). Dynamic vehicle routing using genetic algorithms. *Applied Intelligence*, **27**(1): 89-99.

Hart, W. E. (1994). Adaptive Global Optimization With Local Search. San Diego, CA, University of California, **PhD Thesis**.

Hasan, S. M. K., Sarker, R., Essam, D. and Cornforth, D. (2008). Memetic Algorithms for Solving Job-Shop Scheduling Problems. *Memetic Computing, Springer*, **1**(1): 69-83.

Hayes-Roth, B., Hewett, M., Washington, R., Hewett, R. and Seiver, A. (1989). Distributing intelligence within an individual. In Gasser, L. and Huhns, M., editor, *Distributed Artificial Intelligence* vol. II, pages 385–412. Pitman Publishing, CA.

He, L. and Mort, N. (2000). Hybrid Genetic Algorithms for Telecommunications

Network Back-Up Routeing. *BT Technology Journal*, **18**(4): 42-50.

Helsgaun, K. (2000). An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, **126**(1): 106–130.

Hillier, F. S. and Lieberman, G. J. (2005). *Introduction to Operations Research,* 8 ed., McGraw-Hill, Boston.

Himmelblau, D. M. (1972). *Applied Nonlinear Programming*. McGraw-Hill, New York.

Hock, W. and Schittkowski, K. (1981). *Test Examples for Nonlinear Programming Codes*. Springer-Verlag, New York, Inc., Secaucus, NJ, USA.

Hoffmeister, F. and Sprave, J. (1996). Problem-independent handling of constraints by use of metric penalty functions. In *Proceedings of the Fifth Annual Conference on Evolutionary Programming (EP'96)*, San Diego, CA, pp. 289–294, MIT Press.

Holland, J. H. (1975). *Adaption in Natural and Artificial Systems*. The University of Michigan Press.

Homaifar, A., Qi, C. X. and Lai, S. H. (1994). Constrained optimization via genetic algorithms *Simulation*, **62**(4): 242-254.

Horn, J., Nafploitis, N. and Goldberg, D. E. (1994). A niched Pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp. 82–87, IEEE Press.

Horrocks, D. H. and Khalifa, Y. M. A. (1994). Genetically derived filter circuits using preferred value components. In *IEE Colloquium on Analogue Signal Processing*, London, UK, pp. 4/1-4/5.

Houck, C. R., Joines, J. A. and Kay, M. G. (1996). Utilizing Lamarckian Evolution and the Baldwin Effect in Hybrid Genetic Algorithms. NCSU-IE Technical Report, Department of Industrial Engineering, North Carolina State University.

Hu, X., Huang, Z. and Wang, Z. (2003). Hybridization of the multi-objective evolutionary algorithms and the gradient-based algorithms. In *The 2003 Congress on Evolutionary Computation*, Vol. 2, pp. 870-877.

Huang, J., Jennings, N. R. and Fox, J. (1995). An Agent-based Approach to Health Care Management. *Int. Journal of Applied Artificial Intelligence*, **9**(4): 401-420.

Huaiqing, W. and Chen, W. (1997). Intelligent agents in the nuclear industry. Computer, IEEE Computer Society. **30:** 28-31.

Hung Dinh, N., Yoshihara, I., Yamamori, K. and Yasunaga, M. (2007). Implementation of an Effective Hybrid GA for Large-Scale Traveling Salesman Problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, **37**(1): 92-99.

Ishibuchi, H., Kaige, S. and Narukawa, K. (2005). Comparison Between Lamarckian and Baldwinian Repair on Multiobjective 0/1 Knapsack Problems. In *Evolutionary Multi-Criterion Optimization*, pages 370-385. Springer, Berlin / Heidelberg.

Jacobs, P. H. M., Verbraeck, A. and Mulder, J. B. P. (2005). Flight scheduling at KLM. In *Proceedings of the 37th conference on Winter simulation*, Orlando, Florida pp. 299 - 306.

Jennings, N. R., Sycara, K. and Wooldridge, M. (1998). A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*, **1**(1): 7-38.

Jennings, N. R. and Wooldridge, M. (1998). Applying agent technology. In Jennings, N. R. and Wooldridge, M., editor, *Agent Technology: Foundations, Applications, and Markets*. Springer-Verlag, Berlin, Germany.

Jensen, M. T. (2003). Generating robust and flexible job shop schedules using genetic algorithms. *IEEE Transactions on Evolutionary Computation*, **7**(3): 275-288.

Jeon, G., Leep, H. R. and Shim, J. Y. (2007). A vehicle routing problem solved by using a hybrid genetic algorithm. *Computers & Industrial Engineering*, **53**(4): 680-692.

Jeong, I. K. and Lee, J. J. (1997). Evolving multi-agents using a self-organizing genetic algorithm. *Applied Mathematics and Computation*, **88**(2-3): 293.

Jing, X., Michalewicz, Z. and Lixin, Z. (1996). Evolutionary Planner/Navigator: operator performance and self-tuning. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 366-371.

Jing, X., Michalewicz, Z., Lixin, Z. and Trojanowski, K. (1997). Adaptive evolutionary planner/navigator for mobile robots. *IEEE Transactions on Evolutionary Computation*, **1**(1): 18-28.

Joines, J. A. and Houck, C. R. (1994). On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, Vol. 2, pp. 579-584.

Kim, J. H. and Myung, H. (1997). Evolutionary programming techniques for constrained optimization problems. *IEEE Transactions on Evolutionary Computation*, **1**(2): 129-140.

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, **220**: 671–680.

Kisiel-Dorohinicki, M. (2002). Agent-Oriented Model of Simulated Evolution. In *SOFSEM 2002: Theory and Practice of Informatics*, pages 253-261.

Knowles, J. and Corne, D. W. (2000). M-PAES: a memetic algorithm for multiobjective optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation*, California, USA Vol. 1, pp. 325-332.

Knowles, J. and Corne, D. W. (2001). A comparative assessment of memetic, evolutionary and constructive algorithms for the multi-objective d-msat problem. In *GECCO-2001 Workshop Program*, pp. 162–167.

Knowles, J. and Corne, D. W. (2005). Memetic Algorithms for Multiobjective Optimization: Issues, Methods and Prospects. In Hart, W. E., Krasnogor, N. and Smith, J. E., editor, *Recent Advances in Memetic Algorithms*, pages 313-352. Springer.

Koziel, S. and Michalewicz, Z. (1999). Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evolutionary Computation*, **7**(1): 19-44.

Krasnogor, N. (2002). Studies on the Theory and Design Space of Memetic Algorithms, University of the West of England. **Ph.D. Thesis**.

Krasnogor, N. and Smith, J. (2005). A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, **9**(5): 474-488.

Kruiskamp, M. W. (1996). Analog design automation using genetic algorithms and polytopes. Dept. Elec. Engineering. Eindhoven, The Netherlands, Eindhoven University of Technology. **Ph.D. Thesis**.

Kuechler, W., Vaishnavi, V. K. and Kuechler, D. (2001). Supporting optimization of business-to-business e-commerce relationships. *Decision Support Systems*, **31**(3): 363-377.

Kuwahara, Y. (2005). Multiobjective optimization design of Yagi-Uda antenna. *IEEE Transactions on Antennas and Propagation*, **53**(6): 1984-1992.

Lae-Jeoung, P. and Cheol Hoon, P. (1995). Genetic algorithm for job shop scheduling problems based on two representational schemes. Electronics Letters, IET Periodicals **31:** 2051-2053.

Laleci, G. B., Dogac, A., Olduz, M., Tasyurt, I., Yuksel, M. and Okcan, A. (2008). SAPHIRE: A Multi-Agent System for Remote Healthcare Monitoring through Computerized Clinical Guidelines. In Annicchiarico, R., Cortés, U. and Urdiales, C., editor, *Agent Technology and e-Health*, pages 25-44. Birkhäuser Basel.

Lasdon, L. S., Waren, A. D., Jain, A. and Ratner, M. (1978). Design and Testing of a Generalized Reduced Gradient Code for Nonlinear Programming. ACM Transactions on Mathematical Software **4**(1): 34-50.

Leung, Y. W. (2001). An Orthogonal Genetic Algorithm with Quantization for Global Numerical Optimization Optimization. *IEEE Transactions on Evolutionary Computation*, **5**(1): 41-53.

Li, G., Shangping, D., Shijue, Z. and Guanxiang, Y. (2007). Using Genetic Algorithm for Data Mining Optimization in an Image Database. In *Fourth International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2007*, Vol. 3, pp. 721-723.

Liang, J. J., Runarsson, T. P., Mezura-Montes, E., Clerc, M., Suganthan, P. N., Coello, C. A. C. and Deb, K. (2006). Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-Parameter Optimization. In *Special Session on Constrained Real-Parameter Optimization, IEEE Congress on Evolutionary Computation, CEC 2006*. Singapore.

Liang, J. J. and Suganthan, P. N. (2006). Dynamic Multi-Swarm Particle Swarm Optimizer with a Novel Constraint-Handling Mechanism. In *IEEE Congress on Evolutionary Computation, CEC 2006*, pp. 9-16.

Lieberman, H. (1995). Letizia: An agent that assists web browsing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, Canada, pp. 924–929.

Lim, M. K. and Zhang, Z. (2002). Iterative multi-agent bidding and co-ordination based

on genetic algorithm. In *Proceeding of 3rd International Symposium on Multi-Agent Systems, Large Complex Systems, and E-Businesses, Erfurt*, pp. 682-689.

Linden, D. S. (1997). Automated Design and Optimization of Wire Antennas using Genetic Algorithms. Cambridge, MA, MIT. **Ph.D. Thesis**.

Liu, G. R. and Han, X. (2003). *Computational Inverse Techniques in Nondestructive Evaluation*, CRC Press, Washington, D.C.

Liu, G. R., Ma, W. B. and Han, X. (2002a). An inverse procedure for determination of material constants of composite laminates using elastic waves. *Computer Methods in Applied Mechanics and Engineering*, **191**(33): 3543-3554.

Liu, H. and Frazer, J. H. (2002). Supporting evolution in a multi-agent cooperative design environment. *Advances in Engineering Software*, **33**(6): 319-328.

Liu, J., Jing, H. and Tang, Y. Y. (2002b). Multi-agent oriented constraint satisfaction. *Artificial Intelligence*, **136**(1): 101-144.

Liu, J., Zhong, W. and Jiao, L. (2006). A multiagent evolutionary algorithm for constraint satisfaction problems. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, **36**(1): 54-73.

Ljunberg, M. and Lucas, A. (1992). The OASIS air traffic management system. In *Proceedings of the Second Pacific Rim International Conference on AI (PRICAI-92)*, Seoul, Korea.

Lluch-Ariet, M., Estanyol, F., Mier, M., Delgado, C., González-Vélez, H., Dalmas, T., Robles, M., Sáez, C., Vicente, J., Huffel, S., Luts, J., Arús, C., Silveira, A. P. C., Julià-Sapé, M., Peet, A., Gibb, A., Sun, Y., Celda, B., Bisbal, M. C. M., Valsecchi, G., Dupplaw, D., Hu, B. and Lewis, P. (2008). On the Implementation of HealthAgents : Agent-Based Brain Tumour Diagnosis. In *Agent Technology and e-Health*, pages 5-24. Birkhäuser Basel.

Loannou, P. A. (2008). *Intelligent Freight Transportation*. 1 ed., CRC Press.

Lohn, J. D. and Colombano, S. P. (1999). A circuit representation technique for automated circuit design. *IEEE Transactions on Evolutionary Computation*, **3**(3): 205-219.

Mahfoud, S. and Mani, G. (1996). Financial forecasting using genetic algorithms. *Applied Artificial Intelligence*, **10**(6): 543-565.

Mahlab, U., Shamir, J. and Caulfield, H. J. (1991). Genetic algorithm for optical pattern recognition. Optics Letters, **16**(9): 648-650.

Man, K. F., Tang, K. S. and Kwong, S. (1996). Genetic algorithms: concepts and applications. *IEEE Transactions on Industrial Electronics*, **43**(5): 519-534.

Marczyk, A. (2004). "Genetic Algorithms and Evolutionary Computation." TalkOrigins Archive, from http://www.talkorigins.org.

Melanie, M. (1998). *An Introduction to Genetic Algorithms*. The MIT Press.

Méndez, C. A., Grossmann, I. E., Harjunkoski, I. and Kaboré, P. (2006). A simultaneous optimization approach for off-line blending and scheduling of oil-refinery operations. *Computers & Chemical Engineering*, **30**(4): 614-634.

Meng, A., Ye, L., Roy, D. and Padilla, P. (2007). Genetic algorithm based multi-agent system applied to test generation. *Computers & Education*, **49**(4): 1205-1223.

Merz, P. and Freisleben, B. (1997). Genetic local search for the TSP: new results. In *IEEE International Conference on Evolutionary Computation, 1997*, Indianapolis, IN, USA, pp. 159-164.

Merz, P. and Freisleben, B. (1999). A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem. In *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 99*, Vol. 3, pp. 2063-2070.

Merz, P. and Freisleben, B. (2000). Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, **4**(4): 337-352.

Merz, P. and Freisleben, B. (2001). Memetic algorithms for the traveling salesman problem. *Complex Systems*, **13**(4): 297–345.

Mezura-Montes, E. and Coello, C. A. C. (2002). A Numerical Comparison of Some Multiobjective-Based Techniques to Handle Constraints in Genetic Algorithms. Technical Report EVOCINV-03-2002, Evolutionary Computation Group at CINVESTAV-IPN, Mexico.

Mezura-Montes, E. , Ed. (2009). *Constraint-Handling in Evolutionary Optimization*. Studies in Computational Intelligence. Springer, Berlin / Heidelberg.

Michalewicz, Z. (1994). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag.

Michalewicz, Z. (1995). Genetic algorithms, numerical optimization and constraints. In *The 6th International Conference on Genetic Algorithms*, University of Pittsburgh, Morgan Kaufmann, San Mateo, CA, pp. 151–158.

Michalewicz, Z. and Attia, N. F. (1994). Evolutionary optimization of constrained problems. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pp. 98–108, World Scientific, Singapore.

Michalewicz, Z. and Janikow, C. Z. (1996). GENOCOP: a genetic algorithm for numerical optimization problems with linear constraints. *Communications of the ACM*, **39**(12).

Michalewicz, Z. and Nazhiyath, G. (1995). Genocop III: a co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In *IEEE International Conference on Evolutionary Computation, 1995*, Vol. 2, pp. 647-651.

Michalewicz, Z., Nazhiyath, G. and Michalewicz, M. (1996). A Note on Usefulness of Geometrical Crossover for Numerical Optimization Problems. In *Proceeding of the 5th Annual Conference on Evolutionary Programming*, San Diego, CA, pp. 305-312, MIT Press, Cambridge, MA.

Michalewicz, Z. and Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, **4**(1): 1-32.

Miettinen, K. (1999). Nonlinear Multiobjective Optimization. In *Kluwer's International Series in OR/MS*. Kluwer Academic Publishers, Boston.

Miettinen, K. (2001). Some Methods for Nonlinear Multi-objective Optimization. In *Evolutionary Multi-Criterion Optimization*, pages 1-20. Springer, Berlin / Heidelberg.

Miranker, D. P. and Lofaso, B. J. (1991). The organization and performance of a TREAT-based production system compiler. *IEEE Transactions on Knowledge and Data Engineering*, **3**(1): 3-10.

Molina, D., Herrera, F. and Lozano, M. (2005). Adaptive local search parameters for real-coded memetic algorithms. In *The 2005 IEEE Congress on Evolutionary Computation* Edinburgh, UK, Vol. 1, pp. 888-895.

Morales, A. K. and Quezada, C. V. (1998). A universal eclectic genetic algorithm for

constrained optimization. In *The 6th European Congress on Intelligent Techniques and Soft Computing, EUFIT'98*, Verlag Mainz, Aachen, Germany, pp. 518–522.

Moscato, P. (1989). On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts Towards Memetic Algorithms. Caltech Concurrent Computation Program Report Pasadena, CA, U.S.A, California Institute of Technology.

Muruganandam, A., Prabhaharan, G., Asokan, P. and Baskaran, V. (2005). A memetic algorithm approach to the cell formation problem. *The International Journal of Advanced Manufacturing Technology*, **25**(9): 988-997.

Nakashima, T., Ariyama, T., Yoshida, T. and Ishibuchi, H. (2003). Performance evaluation of combined cellular genetic algorithms for function optimization problems. In *Proceedings of the 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Vol. 1, pp. 295-299.

Nemhauser, G. and Wolsey, L. (1999). *Integer and Combinatorial Optimization*. John Wiley & Sons, New York.

Nicholas, R. J. (2001). An agent-based approach for building complex software systems. *Communications of the ACM*, **44**(4): 35-41.

Nishibe, Y., Kuwabara, K., Suda, T. and Ishida, T. (1993). Distributed channel allocation in ATM networks. In *IEEE Global Telecommunications Conference, GLOBECOM '93*. Houston.

Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. 2 ed., Springer.

Obayashi, S. and Sasaki, D. (2004). Multi-objective optimization for aerodynamic designs by using ARMOGAs. In *Proceedings of the Seventh International Conference on High Performance Computing and Grid in Asia Pacific Region*, pp. 396-403.

Obayashi, S., Sasaki, D., Takeguchi, Y. and Hirose, N. (2000). Multiobjective evolutionary computation for supersonic wing-shape optimization. *IEEE Transactions on Evolutionary Computation*, **4**(2): 182-187.

Olariu, E. B. S. and Zomaya, A. Y., Eds. (2006). *Handbook of Bioinspired Algorithms and Applications*. Computer and Information Science Series. Chapman & HALL/CRC.

Oliveira, E., Fonseca, J. M. and Steiger-Garcao, A. (1997). MACIV: A DAI based

resource management system. *Applied Artificial Intelligence*, **11**(6): 525–550.

Ombuki-Berman, B. and Hanshar, F. (2009). Using Genetic Algorithms for Multi-depot Vehicle Routing. In *Bio-inspired Algorithms for the Vehicle Routing Problem*, pages 77-99. Springer, Berlin / Heidelberg.

Ong, Y. S. and Keane, A. J. (2004). Meta-Lamarckian learning in memetic algorithms. *IEEE Transactions on Evolutionary Computation*, **8**(2): 99-110.

Ong, Y. S., Lim, M. H., Neri, F. and Ishibuchi, H. (2009). Special issue on emerging trends in soft computing: memetic algorithms. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, **13**(8): 739-740.

Ong, Y. S., Lim, M. H., Zhu, N. and Wong, K. W. (2006). Classification of adaptive memetic algorithms: a comparative study. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, **36**(1): 141-152.

Oprea, M. (2004). Applications of Multi-Agent Systems. In *Information Technology*, pages 239-270. Springer, Boston.

Ortiz-Boyer, D., Hervas-Martınez, C. and Garcıa-Pedrajas, N. (2005). A Crossover Operator for Evolutionary Algorithms Based on Population Features. *Journal of Artificial Intelligence Research,* **24**: 1-48.

Overgaard, L., Petersen, H. G. and Perram, J. W. (1996). Reactive motion planning: a multi-agent approach. *Applied Artificial Intelligence*, **10**(1): 35–52.

Pack, D. J., Toussaint, G. J. and Haupt, R. L. (1996). Robot trajectory planning using a genetic algorithm. In *Adaptive Computing: Mathematical and Physical Methods for Complex Environments*, Denver, CO, USA, Vol. 2824, pp. 171-182, SPIE.

Paechter, B., Cumming, A. and Luchian, H. (1995). The use of local search suggestion lists for improving the solution of timetable problems with evolutionary algorithms. In *Evolutionary Computing*, pages 86-93. Springer, Berlin / Heidelberg.

Pan, H. and Wang, I. Y. (1991). The bandwidth allocation of ATM through genetic algorithm. In *Global Telecommunications Conference, GLOBECOM '91*, Vol. 1, pp. 125-129.

Parunak, H. V. D. (1999). Industrial and practical applications of DAI. In *Multiagent systems: a modern approach to distributed artificial intelligence*, pages 377-421.

MIT Press, Cambridge, MA, USA.

Pattie, M. (1994). Agents that reduce work and information overload. *Communications of the ACM*, **37**(7): 30-40.

Pendharkar, P. C. (2007). The theory and experiments of designing cooperative intelligent systems. *Decision Support Systems*, **43**(3): 1014-1030.

Potvin, J. Y. and Bengio, S. (1996). The vehicle routing problem with time windows—Part II: genetic search. *INFORMS Journal on Computing*, **8** (2): 165–172.

Potvin, J. Y., Dubé, D. and Robillard, C. (1996). A hybrid approach to vehicle routing using neural networks and genetic algorithms. *Applied Intelligence* **6**(3): 241–252.

Powell, D. and Skolnick, M. M. (1993). Using genetic algorithms in engineering design optimization with non-linear constraints. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 424 - 431, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA

Pursula, M. and Niittymäki, J., Eds. (2001). *Mathematical Methods on Optimization in Transportation Systems*. Springer.

Qiong, L., Tao, J., Yuchen, F., Quan, L. and Zhiming, C. (2007). Application of Genetic Algorithm in the Optimization of Water Pollution Control Scheme. In *Workshop on Intelligent Information Technology Application*, pp. 189-191.

Rasheed, K. (1998). An adaptive penalty approach for constrained genetic-algorithm optimization. In *Proceedings of the Third Annual Genetic Programming Conference*, Morgan Kaufmann, San Francisco, CA, pp. 584–590.

Ravindran, A. R., Ed. (2007). *Operations Research and Management Science Handbook*. CRC Press.

Reddy, P. C. P., Karimi, I. A. and Srinivasan, R. (2004). Novel solution approach for optimizing crude oil operations. *AIChE Journal*, **50**(6): 1177-1197.

Renders, J. M. and Bersini, H. (1994). Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways. In *Proceedings of the First IEEE Conference on Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence*, Orlando, FL, USA, Vol. 1, pp. 312-317.

Richard, J. B. (1994). *Genetic Algorithms and Investment Strategies*. John Wiley & Sons, Inc.

Rizki, M. M., Zmuda, M. A. and Tamburino, L. A. (2002). Evolving pattern recognition systems. *IEEE Transactions on Evolutionary Computation*, **6**(6): 594-609.

Roth, G. and Levine, M. D. (1992). Geometric primitive extraction using a genetic algorithm. In *Proceedings of the 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR '92*, pp. 640-643.

Runarsson, T. P. and Yao, X. (2005). Search biases in constrained evolutionary optimization. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, **35**(2): 233-243.

Runarsson, T. P. and Yao, X. (2000). Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, **4**(3): 284-294.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*, 2 ed., Prentice Hall.

Safe, M., Carballido, J., Ponzoni, I. and Brignole, N. (2004). On Stopping Criteria for Genetic Algorithms. In Bazzan, A. L. C. and Labidi, S., editor, *Advances in Artificial Intelligence – SBIA 2004*, pages 405-413. Springer-Verlag.

Sahin, C. S., Urrea, E., Uyar, M. U., Conner, M., Hokelek, I., Bertoli, G. and Pizzo, C. (2008a). Uniform distribution of mobile agents using genetic algorithms for military applications in MANETs. In *IEEE Military Communications Conference, 2008. MILCOM 2008*, pp. 1-7.

Sahin, C. S., Urrea, E., Uyar, M. U., Conner, M., Hokelek, I., Conner, M., Bertoli, G. and Pizzo, C. (2008b). Genetic algorithms for self-spreading nodes in MANETs. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, Atlanta, GA, USA, pp. 1141-1142, ACM.

Sait, S. M. and Youssef, H. (2000). *Iterative Computer Algorithms with Applications in Engineering : Solving Combinatorial Optimization Problems*. 1 ed., Wiley-IEEE Computer Society.

Sakawa, M. (1982). Interactive multiobjective decision making by the sequential proxy optimization technique: SPOT. *European Journal of Operational Research*, **9** (4): 386-396.

Sarimveis, H. and Nikolakopoulos, A. (2005). A line up evolutionary algorithm for solving nonlinear constrained optimization problems. *Computers & Operations*

Research, **32**(6): 1499-1514.

Sarker, R., Kamruzzaman, J. and Newton, C. (2003). Evolutionary optimization (EvOpt): a brief review and analysis. *International Journal of Computational Intelligence and Applications* **3**(4): 311-330.

Sarker, R. and Newton, C. S. (2007). *Optimization Modelling: A Practical Approach*. Taylor & Francis/CRC Press.

Sarker, R. and Quaddus, M. (2002). Modelling a Nationwide Crop Planning Problem Using a Multiple Criteria Decision Making Tool. *Computers and Industrial Engineering*, **42**(2-4): 541-553.

Sarker, R. and Ray, T. (2005). Multiobjective Evolutionary Algorithms for solving Constrained Optimization Problems. In *International Conference on Computational Intelligence for Modelling, Control and Automation (CIMCA2005)*, Vienna, Austria, pp. 197-202, IEEE Press-USA.

Sarker, R. and Ray, T. (2009). An improved evolutionary algorithm for solving multi-objective crop planning models. *Computers and Electronics in Agriculture*, **68**: 191-199.

Sasaki, D., Morikawa, M., Obayashi, S. and Nakahashi, K. (2001). Aerodynamic Shape Optimization of Supersonic Wings by Adaptive Range Multiobjective Genetic Algorithms. In Coello, C. A. C., Aguirre, A. H. and Zitzler, E., editor, *Evolutionary Multi-Criterion Optimization*, pages 639-652. Springer, Berlin / Heidelberg.

Schaffer, J. D. (1985). Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 93 - 100, L. Erlbaum Associates Inc.

Schoonderwoerd, R., Holland, O. and Bruten, J. (1997). Ant-like agents for load balancing in telecommunications networks. In *Proceedings of the First International Conference on Autonomous Agents (Agents 97)*, CA, pp. 209–216.

Shi, Y., Liu, R., Yan, N. and Chen, Z. (2008). A Family of Optimization Based Data Mining Methods. In *Progress in WWW Research and Development*, Lecture Notes in Computer Science, Vol. 4976, pages 26-38. Springer, Berlin / Heidelberg.

Shih, J. S., Russell, A. G. and McRae, G. J. (1998). An optimization model for

photochemical air pollution control. *European Journal of Operational Research*, **106**(1): 1-14.

Shintaku, E. (1999). Minimum energy trajectory for an underwater manipulator and its simple planning method by using a Genetic Algorithm. *Advanced Robotics*, **13**(2): 115–138.

Sinclair, M. C. (1999). Minimum cost wavelength-path routing and wavelength allocation using a genetic-algorithm/heuristic hybrid approach. In *IEE Proceedings-Communications*, Vol. 146, pp. 1-7.

Siwik, L. and Kisiel-Dorohinicki, M. (2006). Semi-elitist Evolutionary Multi-agent System for Multiobjective Optimization. In *Computational Science – ICCS 2006*, pages 831-838. Springer, Berlin / Heidelberg.

Slowinski, R., Jaszkiewicz, Andrzej (1999). The 'Light Beam Search' approach - an overview of methodology and applications. *European Journal of Operational Research*, **113**(2): 300-314.

Smith, B. M. and Gemperline, P. J. (2000). Wavelength selection and optimization of pattern recognition methods using the genetic algorithm. *Analytica Chimica Acta*, **423**(2): 167-177.

Smith, R. E., Kearney, P. J. and Merlat, W. (1999). Evolutionary Adaptation in Autonomous Agent Systems — A Paradigm for the Emerging Enterprise. *BT Technology Journal*, **17**(4): 157-167.

Snowdon, J. L. and Paleologo, G. (2007). Airline Optimization. In Ravindran, A. R., editor, *Operations Research and Management Science Handbook.* CRC Press.

Socha, K. and Kisiel-Dorohinicki, M. (2002). Agent-based evolutionary multiobjective optimisation. In *Proceedings of the 2002 Congress on Evolutionary Computation, 2002*, Vol. 1, pp. 109-114.

Soon, G. K., Anthony, P., Teo, J. and Chin Kim, O. (2008). The effect of mutation rate in the evolution of bidding strategies. In *International Symposium on Information Technology, ITSim 2008*, Vol. 1, pp. 1-8.

Soon, N. Y. (2003). Optimizing a Military Supply Chain in the Presence of Random, Non-Stationary Demands. Naval Postgraduate School. Monterey, CA, **Masters Thesis**.

Spalanzani, A. (2000). Lamarckian vs Darwinian Evolution for the Adaptation to Acoustical Environment Change. In *Artificial Evolution*, pages 136-144. Springer, Berlin / Heidelberg.

Sprumont, F. and P.Muller, J. (1997). Amacoia: A multi-agent system for designing flexible assembly lines. *Applied Artificial Intelligence*, **11**(6): 573–590.

Srinivas, N. and Deb, K. (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, **2**(3): 221–248.

Srinivasan, D. and Rachmawati, L. (2006). An efficient multi-objective evolutionary algorithm with steady-state replacement model. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, Seattle, Washington, USA, pp. 715 - 722, ACM Press.

Stan, F. and Art, G. (1997). Is It an agent, or just a program?: A taxonomy for autonomous agents. In *Intelligent Agents III Agent Theories, Architectures, and Languages*, pages 21-35. Springer, Berlin / Heidelberg.

Steuer, R. and Choo, E.-U. (1983). An interactive weighted Tchebycheff procedure for multiple objective programming. *Mathematical Programming*, **26**(3): 326-344.

Steuer, R. E. (1986). *Multiple Criteria Optimization: Theory, Computation, and Application*. John Wiley & Sons Inc.

Surry, P. D. and Radcliffe, N. J. (1997). The COMOGA method: Constrained optimization by multi-objective genetic algorithms. *Control Cybern.*, **26**(3): 391–412.

Sycara, K., Decker, K., Pannu, A., M.Williamson and Zeng, D. (1996). Distributed Intelligent Agents. *IEEE Expert*, **11**(6).

Sycara, K. P. (1998). Multiagent Systems. *The American Association for Artificial Intelligence*.

Takahama, T. and Sakai, S. (2006). Constrained Optimization by the ε Constrained Differential Evolution with Gradient-Based Mutation and Feasible Elites. In *IEEE Congress on Evolutionary Computation, CEC 2006*, pp. 1-8.

Takahama, T. and Sakai, S. (2009). Solving Difficult Constrained Optimization Problems by the ε Constrained Differential Evolution with Gradient-Based Mutation. In Mezura-Montes, E., editor, *Constraint-Handling in Evolutionary*

Optimization, pages 51-72. Springer, Berlin / Heidelberg.

Takahashi, R. H. C., Saldanha, R. R., Dias-Filho, W. and Ramirez, J. A. (2003). A new constrained ellipsoidal algorithm for nonlinear optimization with equality constraints. *Magnetics, IEEE Transactions on*, **39**(3): 1289-1292.

Tang, J., Lim, M. and Ong, Y. (2007). Diversity-adaptive parallel memetic algorithm for solving large scale combinatorial optimization problems. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, **11**(9): 873-888.

Tang, J., Lim, M. H., Ong, Y. S. and Er, M. J. (2005). Solving large scale combinatorial optimization using PMA-SLS. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, Washington DC, USA, pp. 621-628, ACM Press.

Tasgetiren, M. F. and Suganthan, P. N. (2006). A Multi-Populated Differential Evolution Algorithm for Solving Constrained Optimization Problem. In *IEEE Congress on Evolutionary Computation, CEC 2006*, pp. 33-40.

Thangiah, S. R. (1995). Vehicle routing with time windows using genetic algorithms. In Chambers, L., editor, *Application handbook of genetic algorithms: new frontiers*, vol. 2, pages 253–277. CRC Press, Boca Raton, FL.

Thangiah, S. R., Nygard, K. E. and Juell, P. L. (1991). GIDEON: a genetic algorithm system for vehicle routing with time windows. In *Proceedings of the Seventh IEEE Conference on Artificial Intelligence Applications, 1991*, pp. 322-328.

Thangiah, S. R. and Nygard, P. L. (1992). School bus routing using genetic algorithms. In *Proceedings of the SPIE Conference on Applications of Artificial Intelligence Knowledge Based Systems*, Orlando, FL, pp. 387–397.

Thangiah, S. R. and Salhi, S. (2001). Genetic Clustering: An Adaptive Heuristic For The Multidepot Vehicle Routing Problem. *Applied Artificial Intelligence*, **15**(4): 361-383.

Thornton, C. and Boulay, B. d. (1999). *Artificial Intelligence: Strategies, Applications, and Models Through SEARCH*. AMACOM, New York, USA.

Tian, L. (2001). The Nature of Crossover Operator in Genetic Algorithms. In *Rough Sets and Current Trends in Computing*, pages 619-623. Springer, Berlin/Heidelberg.

Tian, L. and Collins, C. (2004). An effective robot trajectory planning method using a

genetic algorithm. *Mechatronics*, **14**(5): 455-470.

Torn, A. and Zilinskas, A. (1989). *Global Optimization*. Lecture Notes in Computer Science, vol. 350. Springer- Verlag, New York.

Tsang, P. W. M. (1995). A genetic algorithm for affine invariant object shape recognition. In *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, pp. 293-298.

Turban, E. and Meredith, J. R. (1994). *Fundamentals of Management Science*. 6 ed., Irwin McGraw-Hill, Boston, MA.

Tzung-Pei, H., Chun-Hao, C., Yeong-Chyi, L. and Yu-Lung, W. (2008). Genetic-Fuzzy Data Mining With Divide-and-Conquer Strategy. *IEEE Transactions on Evolutionary Computation*, **12**(2): 252-265.

Van der Duyn Schouten, F. A. and Vanneste, S. G. (1995). Maintenance optimization of a production system with buffer capacity. *European Journal of Operational Research*, **82**(2): 323-338.

Van Laarhoven, P. J. M. and Aarts, E. H. L. (1987). *Simulated Annealing: Theory and Applications*. Kluwer Academic Publishers, Dordrecht.

Vasile, M. and Locatelli, M. (2008). A hybrid multiagent approach for global trajectory optimization. *Journal of Global Optimization*, **44**(4): 461-479.

Vatn, J., Hokstad, P. and Bodsberg, L. (1996). An overall model for maintenance optimization. *Reliability engineering & systems safety*, **51**(3): 227-354.

Vavak, F., Fogarty, T. and Jukes, K. (1996). A genetic algorithm with variable range of local search for tracking changing environments. In *Parallel Problem Solving from Nature - PPSN IV*, pages 376-385. Springer, Berlin / Heidelberg.

Villegas, F. J., Cwik, T., Rahmat-Samii, Y. and Manteghi, M. (2004). A parallel electromagnetic genetic-algorithm optimization (EGO) application for patch antenna design. *IEEE Transactions on Antennas and Propagation*, **52**(9): 2424-2435.

Vlassis, N. (2007). A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning 2007*, **1**(1): 1-71.

Weber, B., Bojduj, B. and Pohl, J. G. (2006). Tabu Search for Optimization of Military

Supply Distribution. In *The 18th International Conference on Systems Research, Informatics and Cybernetics*, Germany, pp. 87-91.

White, D. J. (1990). A Bibliography on the Applications of Mathematical Programming Multiple-Objective Methods. *The Journal of the Operational Research Society*, **41**(8): 669-691.

Whitley, D., Gordon, V. and Mathias, K. (1994). Lamarckian evolution, the Baldwin effect and function optimization. In *Lecture Notes in Computer Science: Parallel Problem Solving from Nature — PPSN III*, pages 5-15. Springer, Berlin / Heidelberg.

Whitley, D., Starkweather, T. and Shaner, D. (1991). The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination. In Davis, L., editor, *The Handbook of Genetic Algorithms*, pages 350-372. Van Nostrand, New York.

Whitley, L. D. (1993). Cellular Genetic Algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers Inc.

Wierzbicki, A. P. (1982). A mathematical basis for satisficing decision making. *Mathematical Modelling*, **3**(5): 391-405.

Wierzbicki, A. P. and Granat, J. (1999). Multi-objective modeling for engineering applications: DIDASN++ system. *European Journal of Operational Research*, **113**(2): 374-389.

Williams, E., Crossley, W. and Lang, T. (2001). Average and maximum revisit time trade studies for satellite constellations using a multiobjective genetic algorithm. *Journal of the Astronautical Sciences*, **49**(3): 385-400.

Wooldridge, M., Bussmann, S. and Klosterberg, M. (1996). Production sequencing as negotiation. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*, London, UK, pp. 709–726.

Wright, A. H. (1991). Genetic Algorithms for Real Parameter Optimization. In Rawlins, G. J. E., editor, *Foundations of Genetic Algorithms*, pages 205-218. Morgan Kaufmann.

Wu, C. L. (2006). Improving Airline Network Robustness and Operational Reliability by Sequential Optimisation Algorithms. *Networks and Spatial Economics*, **6**(3): 235-251.

Xing, C., Kama, H. and Xiao-Bang, X. (2005). Automated design of a three-dimensional fishbone antenna using parallel genetic algorithm and NEC. *IEEE Antennas and Wireless Propagation Letters*, **4**: 425-428.

Yan, C., Zeng-Zhi, L. and Zhi-Wen, W. (2004). Multi-agent based genetic algorithm for JSSP. In *Proceedings of the 2004 International Conference onMachine Learning and Cybernetics, 2004*, Vol. 1, pp. 267-270.

Yang, A., Abbass, H. and Sarker, R. (2006). Land Combat Scenario Planning: A Multiobjective Approach. In *Simulated Evolution and Learning*, pages 837-844. Springer, Berlin / Heidelberg.

Yano, F. and Toyoda, Y. (1999). Preferable movement of a multi-joint robot arm using genetic algorithm. In *Proceedings of SPIE, Intelligent Robots and Computer Vision XVIII: Algorithms, Techniques, and Active Vision* Boston, MA, USA, Vol. 3837, pp. 80–88.

Yueqin, Z., Jinfeng, L., Fu, D. and Jing, R. (2007). Genetic Algorithm in Vehicle Routing Problem. In *Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIHMSP 2007*, Vol. 2, pp. 578-581.

Zadeh, L. (1963). Optimality and non-scalar-valued performance criteria. *IEEE Transactions on Automatic Control*, **8**(1): 59-60.

Zahir, S., Sarker, R. and Al-Mahmud, Z. (2009). An interactive decision support system for implementing sustainable relocation strategies for adaptation to climate change: a multi-objective optimisation approach. *International Journal of Mathematics in Operational Research* **1**(3): 326 - 350.

Zarka, A. (2005). Project management optimization based on an optical cavity laser modelization. In *Proceedings of the 2005 IEEE International Engineering Management Conference*, Vol. 2, pp. 798-803.

Zhang, Z. and Zhang, C. (2004). *Agent-Based Hybrid Intelligent Systems*. Springer, Berlin / Heidelberg.

Zhong, W., Liu, J. and Jiao, L. (2005). Job-Shop Scheduling Based on Multiagent

Evolutionary Algorithm. In *Advances in Natural Computation*, Lecture Notes in Computer Science, Vol. 3612, pages 925-933. Springer, Berlin / Heidelberg.

Zhong, W., Liu, J., Xue, M. and Jiao, L. (2004). A multiagent genetic algorithm for global numerical optimization. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, **34**(2): 1128-1141.