Bayesian Optimization Algorithms for Multiobjective and Hierarchically Difficult Problems

Nazan Khan

IlliGAL Report No. 2003021
July, 2003

Illinois Genetic Algorithms Laboratory (IlliGAL)
Department of General Engineering
University of Illinois at Urbana-Champaign
117 Transportation Building
104 S. Mathews Avenue, Urbana, IL 61801
http://www-illigal.ge.uiuc.edu

BAYESIAN OPTIMIZATION ALGORITHMS FOR MULTIOBJECTIVE AND HIERARCHICALLY DIFFICULT PROBLEMS

BY

NAZAN KHAN

Bachelor of Technology, Indian Institute of Technology. 2001

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in General Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2003

Urbana, Illinois

Abstract

In the last two decades significant progress has been made in the theory and design of competent genetic algorithms—genetic algorithms (GAs) that solve hard problems quickly, reliably, and accurately (Goldberg, 2002). In contrast to the first generation GAs which use fixed recombination operators, competent GAs employ recombination operators that adapt linkages. Competent GAs have been shown to solve problems of bounded difficulty in polynomial (usually subquadratic) time.

Parallel to the progress in the theory of competent GAs, there has been a growing interest in multiobjective optimization in general and multiobjective genetic algorithms (MOGAs) in particular. However, similar to the first generation GAs, existing MOGAs use fixed genetic operators and there is a need to develop competent MOGAs—GAs that solve hard multiobjective problems quickly, reliably, and accurately.

This study combines the selection scheme of NSGA-II (Deb, Agrawal, Pratap, & Meyarivan, 2000), a multiobjective GA, with the linkage learning capabilities of two powerful single objective competent GAs, BOA (Pelikan, Goldberg, & Cantú-Paz, 1999) and hBOA (Pelikan & Goldberg, 2000), to form two competent MOGAs called multiobjective Bayesian optimization algorithm (mBOA) and multiobjective hierarchical Bayesian optimization algorithm (mhBOA), respectively. A test suite of additively decomposable problems with controllable difficulty has also been developed in a principled manner to test both mBOA and mhBOA. Results show that while mBOA is capable of solving boundedly difficult problems with variable interactions at a single level, mhBOA is also able to solve hierarchically difficult problems.

To Mehboob, Rifat, and Tipu

Acknowledgments

I am grateful to many people for helping me in completing my thesis. Firstly, I am extremely thankful to my parents and brother Tipu for their continuous support and blessings. I am also grateful to my friend Saurav for encouraging me at numerous points of time.

I am falling short of words while expressing my gratitude towards Professor Goldberg, my thesis advisor. He is the one to present me with the excellent opportunity of working with him. I found in him both a wonderful teacher and a great researcher. If today I feel that I have achieved something, I can say I did it because of him. He surely knows how to make people get the best out of themselves.

I am thankful to Professor Deb for introducing me to genetic algorithms and encouraging me to pursue further studies in the field.

I am thankful to all the members of IlliGAL, specifically, Laura Albert, Jacob Borgerson, Martin Butz, Jian-Hung Chen, Ying-Ping Chen, Alex Kosorukoff, Jeffrey Leesman, Xavier Llora, Prasanna V. Parthasarathy, Martin Pelikan, Kumara Sastry, Abhishek Sinha, Nathan Sis, Ravi Srivastava, Andy Vaughn and Tian-Li Yu. I am particularly thankful to Martin Pelikan and Kumara Sastry for their helpful suggestions to my research work. Kumara Sastry and Ravi Prakash guided and helped me in writing the thesis. In particular, I would like to especially thank Kumara Sastry for giving me his time, helping me in organizing this thesis, pointing out my mistakes, and correcting them to make this dissertation meet the standard of IlliGAL. Working with him, I have learned to convey my thoughts more effectively.

I am thankful to Loretta Auvil, Barbara Minsker, and Michael Welge for giving me an exposure to the National Center for Super Computing Applications (NCSA) excellent en-

vironment and the industry-oriented work. They gave me an opportunity to view genetic algorithms from a commercial point of view. I thank them for their cooperation, understanding, and encouragement. The author was financially supported by NCSA in completing her graduate studies at the University of Illinois at Urbana-Champaign.

I am also thankful to Donna Eiskamp, administrative aide, GE department, for helping me with the process of depositing the thesis.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-0163. Research funding for this work was also provided by a grant from the National Science Foundation under grant DMI-9908252. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, or the U.S. Government.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recently, there has been a significant development in the theory and design of competent genetic algorithms. A design decomposition methodology has been proposed and several competent GAs have been developed (Goldberg, 1999; Goldberg, 2002). By competent GAs, we mean the GAs that can solve hard problems quickly, reliably, and accurately. Competent GAs solve problems that were intractable by first generation GAs and often times do so in polynomial (usually subquadratic) number of fitness function evaluations (Pelikan, Goldberg, & Cantú-Paz, 1999; Harik, 1999; Pelikan & Goldberg, 2000). Most existing competent GAs focus only on single objective optimization, although many real-world problems contain more than one objective.

Independent to the development of competent GAs, a number of approaches to solve multiobjective problems have been proposed (Zitzler, Deb, & Thiele, 2000; Deb, 2001; Coello Coello, Van Veldhuizen, & Lamont, 2002). Goldberg (1989) suggested the use of the selection based on nondominated sorting to move the population to the Pareto optimal front, and niching to maintain a good spread of solutions over the front and this led to the development of various MOGAs (Fonseca & Fleming, 1993; Horn, Nafpliotis, & Goldberg, 1994; Srinivas & Deb, 1994; Deb, Agrawal, Pratap, & Meyarivan, 2000).

MOGAs have been extensively used in solving multiobjective real-world problems. However, there has been little or no effort to develop competent multiobjective operators that will efficiently identify, propagate, and combine important partial solutions of the problem

at hand. Such competent multiobjective operators should not only render problems that are intractable by simple MOGAs tractable, but also do so by requiring only a polynomial (hopefully subquadratic) number of function evaluations. Therefore, this thesis borrows the best of two worlds, namely, MOGAs and competent GAs, to develop competent MOGAs—GAs that solve hard multiobjective problems quickly, reliably, and accurately. Specifically, the selection scheme of a MOGA is combined with the linkage identification, exchange, and propagation capabilities of a competent GA.

The three objectives of the thesis are as follows:

1. **Design and Implement Competent MOGAs:** The thesis designs, implements, and tests two multiobjective competent GAs formed by incorporating the multiobjective selection scheme of NSGA-II (Deb, Agrawal, Pratap, & Meyarivan, 2000) into BOA (Pelikan, Goldberg, & Cantú-Paz, 1999) and hBOA (Pelikan & Goldberg, 2000). The developed algorithms are called multiobjective Bayesian optimization algorithm (mBOA) and multiobjective hierarchical Bayesian optimization algorithm (mhBOA), respectively.

2. **Testing:** In order to test the algorithms developed in this study, we need to develop multiobjective problems whose properties such as problem difficulty are tunable. Therefore, this thesis develops a test suite consisting of additively decomposable problems with tunable problem difficulty. Linkage identification, propagation, and exchange of important partial solutions are essential to obtain global solutions to these problems. Furthermore, the test suite also contains hierarchical problems that are not decomposable into subproblems of bounded order and involve variable interactions over multiple levels. Linkage learning, niching, and chunking capabilities are essential to solve these hierarchical problems.

3. **Population Sizing of NSGA-II:** There is very little or no guidance available to set the population size for a MOGA. This thesis attempts to explore the population

size requirement of NSGA-II by using the population-sizing model for fitness sharing (Mahfoud, 1995).

Having discussed the thesis objectives, the following is a description of the organization of the report.

1.1 Road Map

The next chapter briefly describes the mechanics and design of GAs with an explanation of the workings of NSGA-II, a multiobjective GA.

Chapter 3 provides a brief introduction to the probabilistic model-building genetic algorithms (PMBGAs) and describes two powerful PMBGAs, BOA and hBOA.

Chapter 4 describes the multiobjective test functions that are designed to test mBOA and mhBOA. Some of the test functions are deceptive with loose linkage, whereas others are hierarchical in nature.

Chapter 5 describes the workings of mBOA and shows its performance on the test functions. The performance of mBOA is compared with that of a simple MOGA. The results indicate that mBOA succeeds in solving boundedly difficult problems with variable interaction at a single level but fails on problems with variable interaction at multiple levels. A simple multiobjective GA performs worse than mBOA on the multiobjective deceptive test functions with loose linkage and it also fails on multiobjective hierarchical problems.

Chapter 6 discusses the reasons behind the failure of mBOA on hierarchical functions and designs, implements, and tests mhBOA. The performance of mhBOA is compared with that of mBOA. For the functions that are deceptive with loose linkage, but do not involve variable interaction over multiple levels, mBOA proved to be better than mhBOA. However, mhBOA was able to solve multiobjective hierarchical functions, whereas mBOA was unable to do so.

Chapter 7 gives suggestions for the future work in the field of hierarchical and multiobjective problem solving using Bayesian optimization algorithms.

Chapter 8 summarizes the thesis and presents the conclusions.

Chapter 2

Genetic Algorithms (GAs) and Multiobjective Genetic Algorithms (MOGAs)

This chapter gives a brief introduction to simple and competent genetic algorithms, and describes GA design decomposition theory. It also includes the key multiobjective optimization concepts and a concise background on MOGAs, in addition to the discussion on the workings of NSGA-II.

2.1 Genetic Algorithms

Genetic algorithms (GAs) are search procedures motivated by the principles of genetics and natural selection (Holland, 1973; Goldberg, 1989; Goldberg, 2002). GAs are increasingly being used in a variety of industrial optimization problems. They have also shown promising results in classification and machine learning systems. The steps involved in a simple GA are explained below.

First, we need to decide a method to encode the problem as artificial chromosomes. Various representations such as binary codes, gray codes, permutations, floating-point codes, and program codes have been used in GA literature and they affect the GA performance in different ways. A detailed discussion on representation and its effect on GA performance is

given elsewhere (Rothlauf, 2002).

After encoding a problem, we need a fitness function to measure the quality of a candidate solution to differentiate between good and bad solutions. Unlike traditional optimization techniques which require the fitness function to possess certain mathematical properties, the fitness function for a GA can either be a mathematical equation, model, simulation or be assigned interactively by the user. It can also come out through other individuals existing in a co-evolutionary environment.

Once, we decide on representation and fitness function, we are ready to perform the following key steps of a simple GA:

1. Unlike traditional optimization methods that operate on a single solution at a time, a GA operates on a population of chromosomes (or individuals). Each chromosome (or individual) in the population represents a candidate solution of the search problem. The initial population usually consists of individuals generated randomly. However, prior knowledge and other problem specific information can easily be incorporated while generating the initial population.

2. After the initial population is generated, we have to evaluate the fitness of each individual in the population.

3. Then, the following genetic operators are typically applied to the population:

   (a) **Selection:** Selection enforces the survival of the fittest concept in GAs. Specifically, selection procedures allocate more copies to the better individuals. Many selection procedures exist in GA literature and they can be broadly classified into two types: proportionate and ordinal selection schemes. In proportionate selection, the selection is performed on the basis of the fitness values (Holland, 1975; Baker, 1989). In ordinal selection, the selection is based on fitness rank rather than the fitness value itself (Goldberg, Korb, & Deb, 1989; Bäck, Fogel,

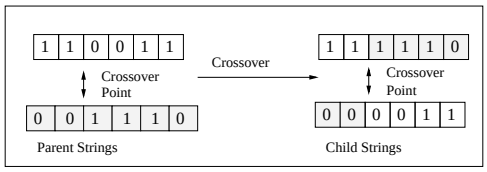


Figure 2.1: Figure showing single point crossover operator in which a random crossover point is selected and alleles on the right side of the crossover point are swapped in the parent strings to create offsprings.

& Michalewicz, 1997). Unlike proportionate selection schemes, which are prone to scaling and stalling problems, ordinal selection schemes do not depend on the scaling of the fitness function (Goldberg & Deb, 1991; Goldberg & Sastry, 2001).

This thesis uses a binary tournament selection which is an ordinal selection scheme. In binary tournament selection, two individuals are randomly chosen from the population and compete against each other. The individual with better fitness wins the tournament and is copied to a mating pool. This procedure is repeated until the mating pool has desired number of individuals. Such a mating pool consists on an average of two copies of the best individual and no copy of the worst individual.

(b) **Recombination:** There are various methods to perform recombination and the type of recombination used significantly affect the performance of a GA (Bäck, Fogel, & Michalewicz, 1997; Thierens & Goldberg, 1993; Sastry & Goldberg, 2002). Recombination is usually performed by pairing parental solutions into groups of two and combining subparts of parents to form new solutions.

This thesis uses single-point crossover operator with simple MOGA. In single-point crossover, the mating pool is partitioned into random pairs of individuals and each pair of parental solutions are recombined with a probability p_c called the crossover probability. A random crossover point is selected and alleles on the right side of the crossover point are swapped in the parent strings to create offsprings. This procedure is shown diagrammatically in figure 2.1.

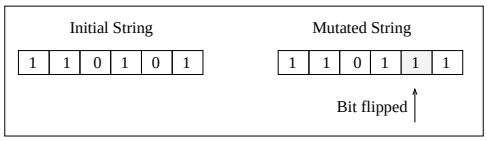


Figure 2.2: Figure showing bit-wise mutation in which every bit of the individual is flipped with a probability p_m called mutation probability.

(c) **Mutation:** While recombination is performed on a pair of individuals, mutation is performed on a single individual by modifying it slightly. There are many ways of performing mutation; however, the key idea is to modify a few (one or two) traits of the individual.

One popular mutation technique is bit-wise mutation in which every bit of the individual is flipped probabilistically with a probability p_m called mutation probability. Usually, the mutation probability is taken to be low. The procedure is shown diagrammatically in figure 2.2.

4. Repeat steps (2) and (3) until one or more termination or convergence criteria are satisfied.

An intuition into the mechanics of GAs has been developed and GAs have been likened to *different facets of human innovation* (Goldberg, 1999; Goldberg, 2002), namely, *continual improvement* and *cross fertilization*. According to this account, selection and mutation, when taken together, are a form of stochastic hill climbing similar to the continual improvement of quality. On the other hand, the combined effect of selection and recombination is more global in nature and has been likened to the process of cross fertilizing or discontinuous innovation.

With a brief understanding of GA mechanics and GA intuition, we now review a GA design theory based on decomposition principle (Goldberg, 1991; Goldberg, Deb, & Clark, 1992; Goldberg, 1993). Using Holland's notion of building blocks (BBs), steps of designing a successful GA were proposed as follows (Goldberg, 2002):

1. Know what GAs process—building blocks (BBs)

2. Know thy BB challengers—BB-wise difficult problems.

3. Ensure an adequate supply of raw BBs.

4. Ensure increased market share for superior BBs.

5. Know BB takeover and convergence times.

6. Make decisions well among competing BBs.

7. Mix BBs well.

Instead of processing the whole solution, a GA decomposes problems into subproblems and processes them independently, either in a serial or parallel manner, and combines subsolutions to form the global optima. Holland (1975) called good subsolutions *building blocks (BBs)*.

Having understood the concept of BBs, the next point stresses the problem difficulty that can be tackled by GAs. We acknowledge that there are impossibly difficult problems, but we are mainly interested in solving problems that are limited in their level of difficulty. If the order of BB is low, a good GA should be able to solve it efficiently. Hence, we aim to design a GA that will solve problems of bounded difficulty (Goldberg, 2002).

One role of the population is to ensure that there is a sufficient supply of raw BBs. If a raw BB is not present in the initial population, it may be unlikely that it will be created during a GA run and GA may converge to a local optima. Facetwise models for predicting population size to ensure an adequate supply of BB have been developed (Holland, 1975; Goldberg, 1989; Goldberg, Sastry, & Latoza, 2001).

It is not enough only to ensure an adequate supply of raw BBs, we must also ensure growth in the market share of the superior BBs. BB growth has been modeled through the schema theorem (Holland, 1975; Goldberg, 1989) and it has been shown that the schema

theorem can be easily satisfied, thereby ensuring the growth of BB market share (Goldberg & Sastry, 2001).

While guaranteeing the growth of superior BBs, we should also ensure that their growth rate is neither too fast nor too slow. In case of too fast growth rate, the raw BBs may not get enough chance to combine to form the global optimum and this may result in premature convergence. Also, in the case of deceptive problems, high growth rate may lead to convergence to the local optima. In case of too slow growth rate, the convergence will be delayed and this may result in a high computational cost. Takeover time and convergence time models for various selection schemes with and without noise have been developed (Goldberg & Deb, 1991; Mühlenbein & Schlierkamp-Voosen, 1993; Thierens & Goldberg, 1993; Bäck, 1994; Miller & Goldberg, 1995).

One of the key points behind the successful design of a GA is that the decision making between competing BBs is stochastic in nature. Decisions are made between individuals and not between competing BBs. Another role of the population is to ensure that decision between competing BBs is made in a statistically correct manner. Increasing the population size reduces the noise in decision making, and population size estimates for ensuring correct decision making have been developed (Holland, 1973; Goldberg, Deb, & Clark, 1992; Harik, Cantú-Paz, Goldberg, & Miller, 1997).

Above steps ensure sustained growth of individual BBs and for a GA success we need to combine subsolutions to form the global solution. Hence, we need to ensure a good mixing of BBs from different partitions to create superior solutions. Facetwise models of BB mixing have shown that fixed recombination operators are exponentially inefficient in mixing BBs and are the primary cause for GA failure (Goldberg, Deb, & Thierens, 1993; Thierens & Goldberg, 1993).

Though most of the above conditions are easy to satisfy and they have been successfully modeled through facetwise models; BB identification and mixing are hard to satisfy. However, after the advent of the fast messy GA (Goldberg, Deb, Kargupta, & Harik, 1993;

Kargupta, 1995), several other GAs have been proposed that can identify and exchange BBs. These advanced GAs that adapt linkage are called competent GAs. Based on the methodology used by these competent GAs to identify BB, they can be broadly classified into 3 categories (Goldberg, 2002):

1. **Perturbation Techniques:** Fast messy GA (fmGA) (Goldberg, Deb, Kargupta, & Harik, 1993), gene expression messy GA (gemGA) (Kargupta, 1996), and linkage identification by non-linearity check/linkage identification by non-monotonicity detection GA (LINC/LIMD GA) (Munetomo & Goldberg, 1999) belong to this class of algorithms.

2. **Linkage Adaptation Techniques:** Linkage learning GA (LLGA) (Harik & Goldberg, 1996) belongs to this class.

3. **Probabilistic Model-Based Techniques:** Population based incremental learning (PBIL) (Baluja, 1994), compact GA (cGA) (Harik, Lobo, & Goldberg, 1998), Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 1999), and hierarchical Bayesian optimization algorithm (hBOA) (Pelikan & Goldberg, 2000) belong to this class of algorithms.

For a more complete discussion of competent GAs and their design, the reader is referred elsewhere (Goldberg, 2002).

2.2 Multiobjective Optimization and Multiobjective Genetic Algorithms (MOGAs)

The previous section briefly reviewed the mechanics, intuition, and design methodology of GAs, and those interested in more background on design and implementation should consult standard references (Goldberg, 1989; Goldberg, 2002). This section briefly introduces

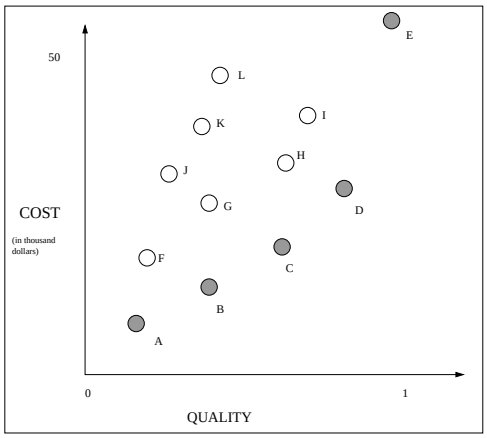


Figure 2.3: Figure showing various levels of quality and cost of the product of a manufacturing company. The dark dots show the points forming the Pareto optimal front. The solution A, B, C, D, and E form a nondominated set of the solution space.

the concepts of multiobjective optimization and GAs used for optimizing multiobjective problems. An algorithmic description of NSGA-II is also presented.

2.2.1 Multiobjective Optimization

Many real-world problems are multiobjective in nature, that is, they have two or more conflicting objectives that have to be optimized. Consider an example of a manufacturing company that wants to produce high quality goods at a low cost. Here, the company has two objectives. The first objective is to produce goods of better quality and the second objective is to produce goods at lower cost. Figure 2.3 shows all possible levels of the cost and the quality that can be achieved by the manufacturing company. Circles from A-L correspond to each level. In figure 2.3, solution D is clearly better than the solution H because it is of higher quality and lower cost. Hence, solution D dominates solution H. Whereas solutions A and D are noninferior with respect to each other. Solution A is of lower cost but the quality of solution D is better than the quality of A. Hence, we cannot say which of the solutions, A and D, is better. The solution to a multiobjective problem consists of a set of individuals that are noninferior among themselves.

A *nondominated set* is defined as the set of solutions which are nondominated among

themselves and any solution outside this set is dominated by at least one solution belonging to this set; and the *globally Pareto optimal set* is defined as the nondominated set of the entire feasible search space (Deb, 2001). The goal of multiobjective optimization is to find the true Pareto optimal set of the solution space. In figure 2.3 the solutions A, B, C, D, and E form a Pareto optimal set over the whole solution space.

The above concepts can be easily generalized to higher number of objective functions. The solution space shown in the previous example (see figure 2.3) is very sparse; however, there can be infinite number of points in the Pareto optimal set. The main challenge there is to find representative solutions of the true Pareto optimal front and to maintain a good spread of solutions over the Pareto optimal front (Goldberg, 1989).

Different approaches commonly employed to tackle multiobjective problems can be broadly classified into two groups (Deb, 2001):

1. **Preference-based Multiobjective Optimization Procedure:** The algorithms belonging to this class employ user's preferences to combine all the objective functions into one single objective function. The combined single objective function is optimized using a single objective optimization technique. However, it is usually difficult to quantify user's preferences; hence, algorithms belonging to this class cannot be easily applied to all problems.

2. **Ideal Multiobjective Optimization Procedure:** Unlike preference-based algorithms, this class of algorithms tries to find all the solutions belonging to the Pareto optimal set. Hence, a user does not need to quantify his or her preferences among the different objective functions. After the representative solutions belonging to the Pareto optimal set are found, user can pick one of those solutions that best suits his or her requirements.

With this basic understanding of multiobjective optimization, we will now briefly discuss multiobjective genetic and evolutionary algorithms that are used as multiobjective solvers.

2.2.2 Multiobjective Genetic Algorithms (MOGAs)

GAs operate on a population of solutions and can process a number of solutions in parallel; therefore, they are particularly suited for multiobjective problems. This is unlike classical approaches that utilize sequential processing and generally need to be applied many times with different weights to discover all the solutions belonging to the Pareto optimal front (Deb, 2001; Coello Coello, Van Veldhuizen, & Lamont, 2002). Similar to their single objective counterparts, MOGAs are robust solvers and are relatively unaffected by the nature, such as shape and continuity, of the Pareto optimal front.

Over the last two decades there has been a growing interest in using GAs to solve multiobjective optimization problems and many algorithms have been proposed. The vector evaluated GA (VEGA) was one of the early implementations of a MOGA (Schaffer, 1985). VEGA was simple and easy to implement, but it was not able to generate concave portions of the Pareto optimal front. Goldberg (1989) suggested the use of selection based on nondominated sorting to move the population towards the Pareto optimal front. He also suggested the use of a niching operator to maintain a good spread of solutions on the front. Many researchers implemented Goldberg's idea to form MOGAs. Here, we briefly describe some of these algorithms and details are given elsewhere (Deb, 2001; Coello Coello, Van Veldhuizen, & Lamont, 2002).

Fonseca and Fleming (1993) proposed a multiobjective genetic algorithm in which the rank of an individual is governed by the number of solutions in the current population by which it is dominated. Their algorithm was efficient and easy to implement. It uses sharing in the fitness function space and its performance is highly dependent on the value of sharing parameter. However, Fonseca and Fleming (1993) suggested a good method for computing it.

Srinivas and Deb (1994) proposed the nondominated sorting genetic algorithm (NSGA). In NSGA, all nondominated individuals are assigned a dummy fitness value and sharing

is performed in the variable space. It was also sensitive to the sharing parameter value. Deb, Agrawal, Pratap, and Meyarivan (2000) proposed the nondominated sorting genetic algorithm-II (NSGA-II) that is computationally more efficient than its predecessor. NSGA-II uses the crowding distance comparison as the niching operator; hence, does not require specification of any sharing parameter. NSGA-II uses elitism, whereas NSGA is a non-elitist scheme.

Horn, Nafpliotis, and Goldberg (1994) proposed the niched Pareto genetic algorithm (NPGA) which used tournament selection based on Pareto dominance. Besides sharing factor, NPGA requires specification of an additional parameter t_{dom}, size of the comparison set, for its good performance. Erickson, Mayer, and Horn (2001) proposed the niched Pareto genetic algorithm 2 (NPGA 2) that uses Pareto ranking along with tournament selection.

Knowles and Corne (2000a) proposed the Pareto archived evolutionary strategy (PAES), in which a solution is mutated to produce a child and a comparison is made between the parent and the child and the better of the two is kept in the population. In PAES, elitism sometimes creates large selection pressure; hence, care needs to be taken while using it. Corne, Knowles, and Oates (2000) proposed the Pareto envelope-based selection algorithm (PESA) which tries to maintain diversity in the fitness function space. A variant of PESA, called the Pareto enveloped-based selection algorithm-II (PESA-II) was also introduced (Corne, Jerram, Knowles, & Oates, 2001).

Zitzler and Thiele (1999) proposed the strength Pareto evolutionary algorithm (SPEA) in which an external nondominated set is maintained and nondominated individuals from each generation are copied to it. Zitzler, Laumanns, and Thiele (2001) introduced a variant of the above scheme called the strength Pareto evolutionary algorithm 2 (SPEA 2).

Van Veldhuizen and Lamont (2000a) proposed a multiobjective messy genetic algorithm (MOMGA) which is based on the messy genetic algorithm of Goldberg, Korb, and Deb (1989). Zydallis, Van Veldhuizen, and Lamont (2001) proposed a multiobjective fast messy genetic algorithm (MOMGA-II) which is based on the fast messy GA of Goldberg, Deb, Kar-

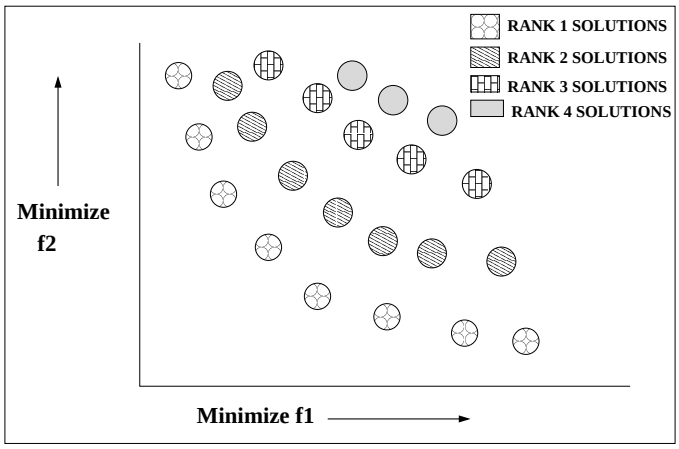


Figure 2.4: Figure showing ranks of different solutions in a population of solutions for a 2-dimensional multiobjective optimization problem. Both the functions are to be minimized. The circles filled with the same pattern represent solutions having the same rank.

gupta, and Harik (1993). Rowe, Vinsen, and Marvin (1996) proposed a Pareto deme-based selection scheme in which Pareto ranking is applied over subparts of the whole population. This approach proved to be more efficient than the other sequential approaches. Coello and Pulido proposed a micro-GA (Coello Coello & Toscano Pulido, 2001a; Coello Coello & Toscano Pulido, 2001b) which uses the Pareto elitist-based selection.

This thesis uses the selection procedure of NSGA-II to develop competent MOGAs. Therefore, a brief outline of NSGA-II is presented next.

2.2.3 Nondominated Sorting Genetic Algorithm-II (NSGA-II)

NSGA-II proposed by Deb, Agrawal, Pratap, and Meyarivan (2000) can be algorithmically explained as follows:

1. Generate an initial random population of size n.

2. Perform nondominated sorting on the population. Ranks are assigned to each member of the population. All the members belonging to the best nondominated set of the members of the population are assigned a rank equal to 1. Every individual of the population, not having rank 1, will be dominated by at least one individual with rank

```
01  for r = 1 to R do the following
02      I is the set of solutions with rank = r
03      c = |I|
04      Initialize the crowding distance of points in I to 0.
05      for each objective b
06        I = sort(I,b)
07        I[1]_distance = I[c]_distance = ∞
08        for i = 2 to (c − 1)
09            I[i]_distance = I[i]_distance + (I[i + 1].b − I[i − 1].b)
```

Figure 2.5: Pseudocode for calculating crowding distance of each member of the population. This is reproduced from Deb, Agrawal, Pratap, and Meyarivan (2000). For calculating crowding distance of solutions having rank equal to r, a set, I containing all the solutions having rank r is formed. c is the cardinality of set I. All the solutions are initially assigned a crowding distance value of 0. $sort(I, b)$ is a function that sorts all the solutions in set I with their b^{th} objective function value. $I[i].b$ refers to the b^{th} objective function value of the i^{th} individual. $I[i]_{distance}$ is the crowding distance of the i^{th} solution.

```
01  if R1 < R2  then Solution1 is the winner.
02  if R1 > R2  then Solution2 is the winner.
03  if R1 = R2  then
04      if CR1 > CR2  then Solution1 is the winner.
05      if CR1 < CR2  then Solution2 is the winner.
06      if CR1 = CR2
07          then randomly choose one of the solutions
as the winner.
```

Figure 2.6: Pseudocode for the multiobjective selection scheme of NSGA-II. The task is to select one of the two solutions, *Solution1* and *Solution2*. *Solution1* has a rank *R1* and a crowding distance *CR1*. *Solution2* has a rank *R2* and crowding distance *CR2*.

1. Next, all the members having rank 1 are temporarily ignored and the members belonging to the next best nondominated set of members are assigned rank equal to 2. This process is continued until all the members in the population are assigned some rank. Any solution other than those with rank 1 will be dominated by at least one solution with a better (lower) rank. Figure 2.4 shows rank assignment of various members of the population.

3. Calculate the crowding distance of each member of the population. The crowding distance of a member is only influenced by the members belonging to the same rank as that of the member. It is a measure of how crowded the neighborhood of the member is. Let us assume that the highest rank assigned in the current population is R then the crowding distance is calculated using the algorithm shown in figure 2.5. The end points of the Pareto optimal front are assigned a very high crowding distance value to ensure that the end points are not lost during the run.

4. Use selection operator to choose promising solutions from the population. Since a multiobjective optimization problem has more than one objective function, NSGA-II uses its special selection operator to compare two solutions and decide which one is better, and this selection operator is algorithmically described in figure 2.6.

5. Perform crossover and mutation operator on the population to generate offspring population.

6. Combine the parent and child populations. Elitism proved to be very useful in the case of multiobjective optimization problems and this step incorporates elitism in NSGA-II.

7. Perform nondominated sorting on the combined population of size *2n*.

8. Calculate crowding distance for each member of the combined population.

9. Keep adding *best* fronts of the combined population to the the new population until

new population is of size n. In case it is not possible to add the whole front to the new population (as population size should not exceed n), then the members of the front are added based on the crowding distance value (members having higher crowding distances are preferred over the members having smaller crowding distance value).

10. Repeat steps (5) to (9) until some convergence criterion are satisfied.

NSGA-II has been tried on various multiobjective problems and it has been successful in consistently maintaining a good spread of solutions on the Pareto optimal front.

2.3 Summary

In this chapter, the mechanics, intuition, and design theory of GAs were explained. Brief introduction to multiobjective optimization in general and multiobjective genetic algorithms in particular was given. Also, the workings of NSGA-II were described in the later part of the chapter.

Chapter 3

Probabilistic Model-building Genetic Algorithms

Recent studies on simple and competent GAs have demonstrated that BB identification and BB exchange are critical for solution success (Goldberg, Deb, & Thierens, 1993; Goldberg, 2002). Recently, probabilistic model building GAs (PMBGAs) have emerged as a leading class of competent GAs, which not only handle problems of bounded difficulty with single level variable interactions, but also solve boundedly difficult problems with hierarchical variable interactions. Next, a brief introduction to PMBGAs is given, followed by a description of BOA and hBOA. In short, PMBGAs built probabilistic model of promising solutions and create new offsprings based on the model. A typical PMBGA consists of the following steps:

1. Randomly generate an initial population.
2. Select good solutions out of the current population. Any selection procedure that can be used in a simple GA can also be used here.
3. Build a probabilistic model of the selected individuals.
4. Create offsprings using the model built in the previous step.
5. Repeat steps (2) to (4) until some termination criterion has been met.

Two of the above steps need further explanation. One is model building (step 3) and the other is creation of new solutions (step 4) from the model.

Building a model involves 3 key issues:

1. **Representation:** The first thing we need before building a model is a way to represent it. Various representation such as marginal product model, Bayesian networks, and decision graphs can be used.

2. **Metric:** Once we decide on a representation of the model, we need a measure to distinguish between good and bad models. Different metrics such as Bayesian metrics (Cooper & Herskovits, 1992; Heckerman, Geiger, & Chickering, 1994) and minimum description length (MDL) metrics (Rissanen, 1978; Rissanen, 1989; Rissanen, 1996) have been used in literature.

3. **Search:** With representation and a metric in hand, we have to finally decide on a search method to obtain the best among different possible models of the selected population. Usually, local search methods such as greedy search algorithm have been used in the previous studies.

After the model is built, we have to estimate the distribution of the selected set of individuals from the current population. New solutions are sampled using this estimated probability distribution.

Existing PMBGAs can be broadly classified into two categories, namely, discrete domain PMBGAs and continuous domain PMBGAs; they can be sub-classified into three classes in each category based on the types of the model and representation used. The following is a concise survey of PMBGAs and interested reader should refer elsewhere (Pelikan, Goldberg, & Lobo, 1999):

1. **Univariate:** The algorithms belonging to this class assume that there is no interaction between variables. These algorithms are good for solving linear problems but they fail in case of problems with variable interactions. In this case, there is no search for good models; therefore, there is no need for any metric. PBIL (Baluja, 1994), UMDA

(Mühlenbein & Paaß, 1996), and cGA (Harik, Lobo, & Goldberg, 1998) are discrete domain univariate PMBGAs. $PBIL_C$ (Sebag & Ducoulombier, 1998) and $UMDA_C$ (Larrañaga, Etxeberria, Lozano, & Peña, 2000) are continuous domain univariate PMBGAs.

2. **Bivariate:** The algorithms belonging to this class model binary interactions among variables. In this class of algorithms, there is a structure associated with the model such as chain or dependency trees. MIMIC (De Bonet, Isbell, & Viola, 1997), COMIT (Baluja & Davies, 1997), and BMDA (Pelikan & Mühlenbein, 1999) are discrete domain bivariate PMBGAs. $MIMIC_C$ (Larrañaga, Etxeberria, Lozano, & Peña, 2000) is a continuous domain bivariate PMBGA.

3. **Multivariate:** The algorithms belonging to this class model multivariate interactions among variables. Compared to bivariate PMBGAs, they have more complex structure associated with them such as clusters, Bayesian networks, or decision graphs. ECGA (Harik, 1999), FDA (Mühlenbein, Mahnig, & Rodriguez, 1999), PADA (Soto, Ochoa, Acid, & de Campos, 1999), BOA (Pelikan, Goldberg, & Cantú-Paz, 1999), hBOA (Pelikan & Goldberg, 2000), and EBNA (Etxeberria & Larrañaga, 1999) are discrete domain multivariate PMBGAs. IDEA (Bosman & Thierens, 2000) and extension of BOA to continuous domains by Pelikan, Goldberg, and Tsutsui (2001) are continuous domain multivariate PMBGAs.

This thesis is based on BOA and hBOA. Hence, BOA and hBOA are described next.

3.1 Bayesian Optimization Algorithm (BOA)

The Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 1999; Pelikan, 2002) is a PMBGA that uses Bayesian networks to build model and sample new solutions from the selected members of the parent population. It is able to encode multivariate

interactions among variables.

The algorithm can be described by following steps:

1. Generate an initial random population.

2. Select good solutions from the current population.

3. Construct a Bayesian network using the chosen metric.

4. Generate a set of new strings using the constructed Bayesian network.

5. Repeat steps (2) to (4) until some termination criterion is met.

Next, the key issues of a PMBGA, namely, representation, metric, model building and model sampling are explained specifically for BOA.

3.1.1 Representation

A Bayesian network is a directed acyclic graph with nodes representing variables and edges representing dependencies between them. The edges are directed and have a numeric values associated with them that correspond to the conditional probabilities of the variables corresponding to the edges. The conditional probability of a child node (a variable which is dependent on other variables) conditioned with the parent node (a variable on which the child variable depends) is encoded in the Bayesian network. Bayesian networks are composed of two components, namely, *structure* and *parameters* (Howard & Matheson, 1981; Pearl, 1988). Structure corresponds to the nodes and edges present in the network, while parameters correspond to tables or other data structures containing the probabilities.

3.1.2 Metric

This thesis uses the K2 metric, a special kind of Bayesian Dirichlet metric, to evaluate the quality of networks. The following description of K2 metric is based on Pelikan (2002).

Bayesian metric uses Bayes rule and computes the marginal likelihood of a Bayesian network structure with respect to the given data set. The data set here is the existing population and fitness values of each member in the population. The quality of a network is given by its marginal likelihood with respect to the given data set. The Bayes rule for the network can be written as,

$$p(B|D) = \frac{p(B)}{p(D)} \int_\theta p(\theta|B)p(D|B,\theta)d\theta,$$

here, B is the evaluated Bayesian network structure; D is the data set, and each value of θ represents one possible mode of assigning conditional probabilities in the network B, $p(B)$ is the prior probability of the network structure B, $p(\theta|B)$ is the prior probability of parameters θ given B, and $p(D|B,\theta)$ is the probability of D given B and θ. If it is assumed that the conditional probabilities follow Dirichlet distribution, after some approximation it can be written,

$$BD(B) = p(B) \prod_{i=1}^{n} \prod_{\pi_i} \frac{\Gamma(m'(\pi_i))}{\Gamma(m'(\pi_i)+m(\pi_i))} \prod_{x_i} \frac{\Gamma(m'(x_i,\pi_i)+m(x_i,\pi_i))}{\Gamma(m'(x_i,\pi_i))},$$

here, $X = (X_0, X_1, ..., X_{n-1})$ is a vector of all variables in the problem, Π_i is the set of parents of X_i in the network (the set of nodes from which there exists an edge to X_i), $p(B)$ is the prior probability of B; $m(\pi_i)$ is the number of instances with parents Π_i equal to π_i, the product over x_i runs over all instances of X_i (in the case of binary representation these are 0 and 1), the product over π_i runs over all instances of parents Π_i of X_i (all possible combinations of values of Π_i), $m(x_i,\pi_i)$ is the number of instances with $X = x_i$ and $\Pi_i = \pi_i$. $m'(\pi_i)$ and $m'(x_i,\pi_i)$ are prior information about the statistics $m(\pi_i)$ and $m(x_i,\pi_i)$, respectively. In the case of K2 metric, no prior information is used; hence, $m'(x_i,\pi_i) = 1$ and $m'(\pi_i) = \sum_{x_i} m'(x_i,\pi_i)$. For further details, refer elsewhere (Cooper & Herskovits, 1992; Heckerman, Geiger, & Chickering, 1994; Pelikan, 2002).

3.1.3 Model Building

In BOA, a greedy algorithm is used to search for a good network. Learning of the Bayesian network takes place in two phases:

1. **Learning the Structure:** Greedy algorithm (Heckerman, Geiger, & Chickering, 1994) starts with an empty network and performs the following graph operations until it ceases to observe any improvement in the network measured by the chosen metric:

   (a) **Edge Addition:** An edge can be added to the network. The resulting network will take into account an additional dependency between the variables.

   (b) **Edge Removal:** An edge may be removed from the network. The resulting network will ignore one of the dependencies between the variables.

   (c) **Edge Reversal:** This is equivalent to removing an edge and then adding an edge.

2. **Learning the Probabilities:** The selected pool of good solutions is used as data set to calculate the conditional probability of the variables conditioned on their parent variables.

The model is constructed by first learning the structure and then learning the probabilities.

3.1.4 Model Sampling

Once the Bayesian network has been found, it is used to generate offspring population. It must be ensured that the parent variables on which a child variable depends must be generated before the child variable. Sampling of new solutions is done in the following two steps:

1. **Creation of Ancestral Ordering of the Variables:** A sequence of variables is found so that parent variables are listed before their children. This can be done by performing the following steps (Pelikan, 2002):

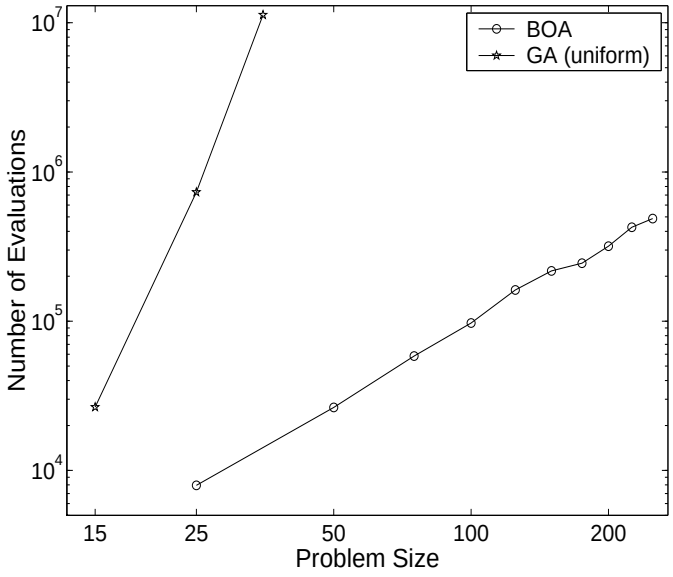


Figure 3.1: Figure comparing the performance of BOA and that of simple GA on trap-5 function. The figure is reproduced from Pelikan (2002) and is published here with prior permission from the author.

(a) Mark all the variables as unprocessed.

(b) Empty the list of ordered variables.

(c) Add any variable with marked parents at the end of the ordering and mark the variable.

(d) If any unmarked variables remain, go to (c).

2. **Generating New Solutions Using the Ancestral Ordering:** After determining the ancestral ordering, new solutions are generated. For each individual, variables are generated according to the ancestral ordering.

The Bayesian optimization algorithm has been shown to perform well on the deceptive functions. BOA was compared with a simple GA on a onemax problem and a composed trap function of order 5 (Pelikan, 2002). The simple GA performed better than BOA on a onemax problem, but both the algorithms required near-linear number of evaluations. Figure 3.1 compares the performance of simple GA and that of BOA on trap-5 function. The simple GA was unable to identify the correct BB linkage; hence, it needed exponentially large

population size. Whereas, BOA still needed only subquadratic number of fitness function evaluation.

3.2 Hierarchical Bayesian Optimization Algorithm (hBOA)

Hierarchical composition is commonly found in numerous complex systems and processes employed in commerce, science, and technology. A hierarchical system consists of a big system composed of many smaller subsystems and each of the subsystems also has a hierarchical structure (Simon, 1969). They can be modeled as a hierarchical problem that is not decomposable into subproblems of bounded order and in which variables interact over multiple levels. Hierarchical decomposition has been found to be very useful in reducing the complexity of such problems. Pelikan and Goldberg (2000) suggested that to solve a hierarchical problem, an algorithm must contain linkage learning, niching, and chunking capabilities. BOA, described in the previous section, does not have these capabilities; hence, cannot solve hierarchical problems. Pelikan and Goldberg (2000) modified BOA by incorporating the necessary tools in it for handling hierarchical problems. While search method and linkage learning procedure are similar to BOA, three issues in hBOA need further explanation:

1. **Representation:** In case of hierarchical functions, lower level solutions are combined to form solution at the higher level. Hence, the inherent structure behind the optimizer should have a hierarchical nature that will allow a group of variables to be treated as a single variable and care must be taken to do this compactly to make the algorithm efficient. Decision graphs are used in hBOA to ensure parameter compression, more complex models, and better learning (Chickering, Heckerman, & Meek, 1997; Friedman & Goldszmidt, 1999; Pelikan, 2002). The decision graphs can be constructed using the following two operations:

(a) **Merge:** Two leaves can be merged to one.

(b) **Split:** A leaf can be split into two children.

The Bayesian network with decision graphs can be constructed by the following greedy algorithm (Pelikan, 2002):

(a) Initialize a decision graph DG for each node X to a graph containing only a single leaf.

(b) Initialize the network B as an empty network.

(c) Choose the best split or merge. It must be ensured that it does not result in a cycle in B.

(d) Stop when the score does not improve by the best operator.

(e) Perform the chosen operation.

(f) If the operator was a split, add the corresponding edge into B.

(g) Go to step (c).

2. **Metric:** Metric used in BOA can also be used in hBOA with some modifications to ensure that good quality models with less complexity are chosen over the others. Bayesian metrics are very sensitive to the noise in the data; therefore, many times lead to overly complex models. Using prior information to bias the metric to favor simpler models, the above problem can be solved. Simpler networks are preferred over complex networks by decreasing the prior probability of a network exponentially with the description length of the set of parameters they require (Friedman & Goldszmidt, 1999). In this thesis, K2 metric with penalty for complex models is used.

3. **Niching:** Since, it is very difficult to know at the early stage which of the partial solutions are leading to the correct optima, it is important to maintain diverse partial solutions in the population. Restricted tournament replacement (RTR) (Harik, 1995)

is used as a diversity preserving mechanism in hBOA. In RTR, for each individual in child population window size number of parents are randomly selected. The child is compared with the closest parent and the winner is put into the new population. It is empirically found that the window size equal to problem size is best suited (Pelikan & Goldberg, 2001).

After understanding the key differences in BOA and hBOA, hBOA is algorithmically explained in the following steps:

1. Generate an initial random population.
2. Select good solutions from the current population.
3. Construct a network by constructing decision graphs encoding conditional probabilities.
4. Generate a set of new strings using the network with decision graphs and call this population O(t).
5. Create a new population using RTR to incorporate each member of O(t) into the parent population.
6. Repeat steps (2) to (5) until some termination criterion is met.

hBOA is not only able to solve boundedly difficult problems with variable interactions at a single level, but is also able to solve problems with variable interactions at multiple levels in subquadratic time.

3.3 Summary

This chapter introduced PMBGAs and discussed two powerful PMBGAs, BOA and hBOA. BOA is able to solve problems for which BB identification and linkage learning are critical, while it is not able to solve hierarchically difficult problems. On the other hand, hBOA is able to solve both classes of problems.

Chapter 4

Test Functions for Testing mBOA and mhBOA

This chapter designs multiobjective test functions in a principled manner to test mBOA and mhBOA. The problem difficulty of these test functions can be easily controlled by the researchers. For a GA to solve these problems efficiently, it should be able to identify BBs and mix them well. These functions are boundedly difficult and an algorithm that is able to solve them without any problem specific information is also expected to be able to solve a problem of equal or lower level of difficulty.

Before developing the test functions, we present a brief overview of existing multiobjective test functions. Later, the subfunctions that form the building blocks for the test functions are described followed by a brief note on the linkage patterns in the test functions. Lastly, the test functions are described in the increasing order of problem difficulty.

4.1 Survey of Multiobjective Test Functions

Coello Coello, Van Veldhuizen, and Lamont (2002) have classified existing multiobjective test functions into six categories:

- **Extension of Single Objective Functions:** Multiobjective test functions belonging to this category are extensions of single objective test functions (Kursawe, 1990; Quagliarella & Vicini, 1997; Deb, 1999a).

- **Unconstrained Multiobjective Functions:** Unconstrained multiobjective problems do not have any constraints imposed on the objective functions (Fonseca & Fleming, 1995; Schaffer, 1984; Van Veldhuizen & Lamont, 2000b; Zitzler, Deb, & Thiele, 2000).

- **Constrained Multiobjective Functions:** Constrained multiobjective optimization problem have constraints imposed on the objective functions (Binh & Korn, 1997; Deb, Pratap, & Meyarivan, 2001).

- **Multiobjective Function Generators:** Deb proposed a methodology to construct multiobjective optimization problems having the desired characteristics (Deb, 1999a; Deb, 1999b; Deb, 1999c).

- **Combinatorial Multiobjective Test Functions:** Combinatorial multiobjective problems have discrete solution space (Horn & Nafpliotis, 1993; Deb, 1999c).

- **Real World Multiobjective Test Functions:** The test functions belonging to this class are used in solving real-world problems (Obayashi, Nakahashi, Oyama, & Yoshino, 1998; Knowles & Corne, 2000b).

The test functions used in this thesis are unconstrained, combinatorial, and extensions of single objective test functions. They are adversarially designed and belong to a class of additively decomposable, boundedly difficult problems.

4.2 Building Blocks for Test Functions

An adversarial approach to test function design is sometimes employed in principled algorithm design practice (Goldberg, 1987; Goldberg, 2002) and three key problem difficulties have been identified, namely, intra-BB, inter-BB and extra-BB problem difficulties. If an algorithm is able to solve boundedly difficult problems without any problem specific information then it is expected to be able to solve other problems of equal or lower level of

difficulty. Since, deception is considered to be the hardest of the three problem difficulties, this thesis designs test function that are boundedly deceptive, and to solve these functions it is extremely important for a GA to identify and mix BBs well.

This section explains the subfunctions that are used as building blocks to design test functions. In this thesis, we encode individuals as binary strings. All the subfunctions described below are functions of unitation of the string, u, that can be calculated as follows,

$$u(x) = \Sigma_{i=0}^{\ell-1} x_i,$$

where, x is the binary string, x_i represents the bit at the i^{th} position and ℓ is problem size.

4.2.1 Minimal Deceptive Function

The function is defined for a 2-bit string as follows,

$$g_{MDP}(x) = \begin{cases} 1.0 & \text{if u} = 2 \\ 0.9 & \text{if u} = 0 \\ 0.0 & \text{if u} = 1. \end{cases}$$

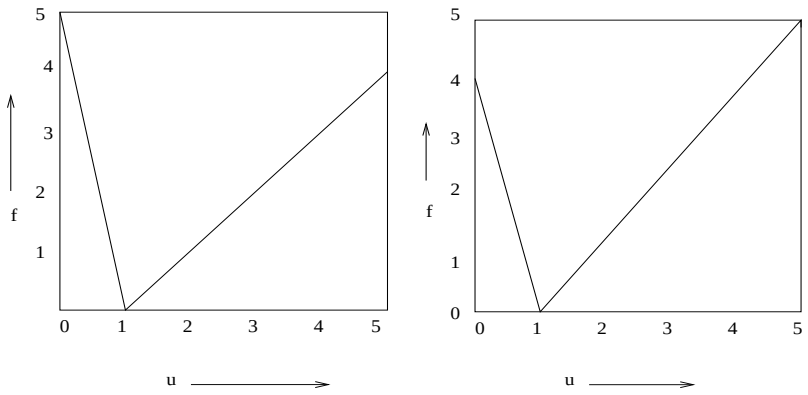
The function involves bivariate interaction between the variables. The complement of the above function can be created as follows:

$$g'_{MDP}(x) = g_{MDP}(\bar{x}),$$

where, $\bar{x}$ is the complement of the binary string x.

4.2.2 k-Bit Trap Function

The $k-$bit trap is defined for a $k-$bit binary string as follows (Deb & Goldberg, 1993; Deb, Horn, & Goldberg, 1993),



(a) Trap-5 function with global optima at the string with all bits equal to 0.

(b) Trap-5 function with global optima at the string with all bits equal to 1.

Figure 4.1: Figure describing trap-5 functions for a 5-bit string. The x-axis represents unitation function of the string, u. The y-axis represents the fitness function value of the string.

$$g_{trap-k}(x) = \begin{cases} \frac{a}{z}(z-u) & \text{if } u \leq z \\ \frac{b}{\ell-z}(u-z) & \text{otherwise.} \end{cases}$$

The function is deceptive and involves multivariate interaction and has a global optima at all 1s ($b > a$). The bitwise complement of the above function has global maxima at all 0s. 5-bit trap functions are shown in figure 4.1.

4.3 Linkage Patterns

In the functions of the previous section, alleles belonging to a building block can be placed anywhere in the string. If the alleles, belonging to the same building block, are placed at the adjacent positions in the string then the function is said to be *tightly linked*, and in this case, the probability of a simple crossover operator disrupting a BB is very low. If the alleles belonging to a building block are placed far away from one another then the linkage pattern is called *loose linkage*, and in this case the probability of a simple crossover operator disrupting

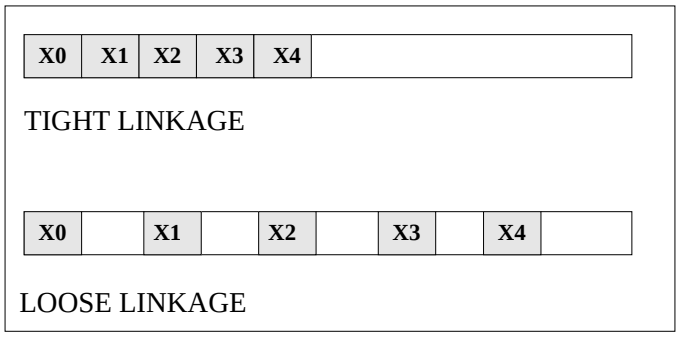


Figure 4.2: Figure illustrating tight and loose linkage. X0, X1, X2, X3, and X4 are alleles of the same building block. In tight linkage, alleles of the same building blocks are placed adjacent to each other. In case of loose linkage, alleles of the same building blocks are placed faraway from each other.

a BB is very high (see figure 4.2). Hence, the problem with tight linkage is relatively easy for a GA with a simple crossover operator, whereas, the problem with loose linkage is hard for them.

4.4 Test Functions

Having explained the building blocks for creating test functions and the concept of loose linkage, we now list the test functions in increasing order of problem difficulty.

4.4.1 Easy Functions

The functions belonging to this category do not have any variable interaction. Hence, they can be easily solved by a simple GA.

Onemax (T1)

This is the number of bits in the string which are equal to one. It has global optima at the string with all 1s.

$$F_{T1}(x) = \Sigma_{i=0}^{\ell-1} x_i.$$

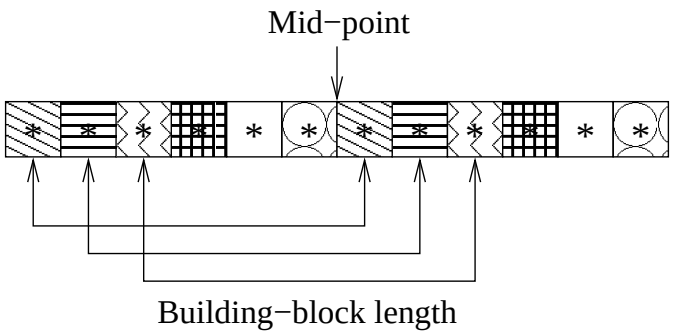


Figure 4.3: Figure illustrating the linkage in the interleaved MDP (T3 and T4). Rectangular blocks represent a bit position in the binary string. Bits, represented by rectangular blocks filled with the same pattern, are linked with each other.

Zeromax (T2)

This is the complement of onemax and has the global optima at the string with all 0s.

$$F_{T2}(x) = \Sigma_{i=0}^{\ell-1}(1 - x_i).$$

4.4.2 Hard Functions with Loose Linkage

The functions belonging to this class involve variable interaction (bivariate or multivariate) at a single level. They are loosely linked and parameterized on the number of building blocks, m. They are difficult in three aspects, namely, deception (Deb & Goldberg, 1993), loose linkage, and multimodality (Deb, Horn, & Goldberg, 1993; Goldberg, Deb, & Horn, 1992).

Interleaved Minimal Deceptive Function (T3 & T4)

The function can be mathematically formulated as,

$$F_{T3} = \Sigma_{i=0}^{m-1} g_{MDP}(u_i),$$

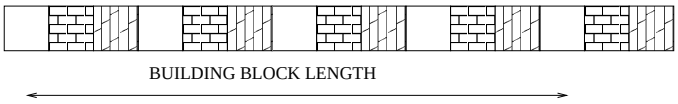


Figure 4.4: Figure describing the coupling of the bits in the multiple interleaved trap-5 function (T5 and T6). Bits, represented by rectangular blocks filled with the same pattern, are linked with each other.

where,

$$u_i = x_i + x_{\frac{\ell}{2}+i},$$

The loose linkage is incorporated by dividing the string into two halves and one bit from each half is coupled with one corresponding bit from the other half. The zeroth bit is coupled with $(\frac{\ell}{2})^{th}$ bit. The first bit is coupled with $(\frac{\ell}{2}+1)^{th}$ bit, and so on (see figure 4.3). T4 is the complement of T3.

Multiple Interleaved 5-bit Trap Function (T5 & T6)

The function can be mathematically formulated as,

$$F_{T5} = \Sigma_{i=0}^{m-1} g_{5-trap}(u_i),$$

where,

$$u_i = \Sigma_{j=0}^{4} x_{i+\frac{\ell}{5}j},$$

g_{5-trap} is the $k-$bit trap function, described earlier, with $a = 4$, $b = 5$, $z = 1$ and $k = 5$. The loose linkage is incorporated by coupling 0^{th} bit, $(\frac{\ell}{5})^{th}$ bit, $2(\frac{\ell}{5})^{th}$ bit, $3(\frac{\ell}{5})^{th}$ bit, and $4(\frac{\ell}{5})^{th}$ bit altogether; and 1^{st} bit, $(\frac{\ell}{5}+1)^{th}$ bit, $(2\frac{\ell}{5}+1)^{th}$ bit, $(3\frac{\ell}{5}+1)^{th}$ bit, and $(4\frac{\ell}{5}+1)^{th}$ bit altogether; and so on (see figure 4.4). T5 has optima at all 0s. T6 is complement of T5 and has optima at all 1s.

4.4.3 Hierarchical Test Functions

This class of test functions have multivariate interactions at multiple levels and are called hierarchically decomposable functions (Goldberg, 1998; Watson, Hornby, & Pollack, 1998). It is observed that the variable interaction at multiple levels increases the problem difficulty considerably. The following explanation of hierarchically decomposable functions (HDFs) is based on Pelikan and Goldberg (2000).

In HDFs, each building block is assigned two distinct properties, namely, *fitness contribution* and *interpretation*. The overall fitness function is defined as the sum of fitness contributions of each building block. Let $X = (X_0, ..., X_{\ell-1})$ be the input vector. The value of variable X_i is denoted by x_i. The subscript $i \in \{1, ..., L-1\}$ denotes the level. Also, at the i^{th} level there are p_i functions that will contribute to the overall fitness. A vector of building block interpretations is passed from the lower level to the higher level. Each level i will have p_i recursively computed interpretations.

The zeroth level has interpretation value equal to the value of the input variables. Hence, $v_{0,k} = x_k$. Hence, $p_0 = \ell - 1$. For $i > 0$, let $z_{i,j}$ be the j^{th} contributory function at the i^{th} level. $z_{i,j}$ is defined on a subset of interpretations from the lower level, with indices from $S_{i,j} \in \{0, .., p_{i-1} - 1\}$. These interpretations are combined by the function $T_{i,j}$ to form a higher level interpretation. The vector of interpretations with indices from $S_{i,j}$ is denoted by $V_{i,j}$. Hence, $V_{i,j} = \{v_{i-1,k} | k \in S_{i,j}\}$. Also,

$$v_{i,j} = \begin{cases} T_{i,j}(V_{i,j}) & \text{if } i > 0 \\ x_j & \text{otherwise,} \end{cases}$$

where, $v_{i,j}$ is the j^{th} interpretation at the i^{th} level.

As an example of HDFs, hierarchical trap functions are explained. In case of the hierarchical trap function, the underlying structure is a balanced q-ary tree ($q \geq 3$) (Pelikan & Goldberg, 2001). The interpretation function is used to interpret blocks of all 0s and all 1s

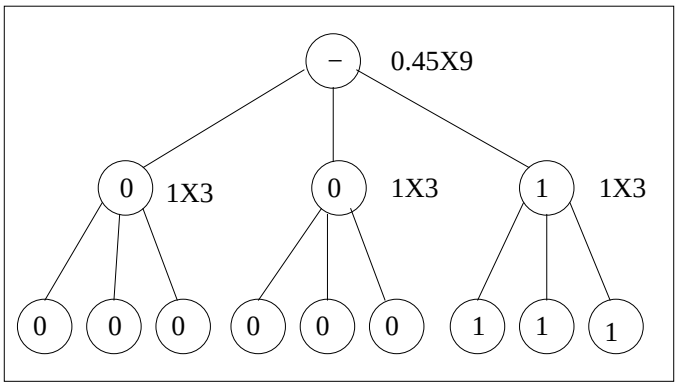


Figure 4.5: An example of the hierarchical trap function for a string X = 000000111. The interpretations are displayed by circles. The contribution of each node is shown on the right side of the circle with interpretation in the node. The fitness of this input string is 13.05. Note that figure is reproduced from Pelikan and Goldberg (2001).

to 0 and 1, respectively. Any other block is interpreted as '-'. The contribution function is a function of unitation. If the input string has '-' then the function returns 0. The function is formulated as follows,

$$z(u) = \begin{cases} z_{high} & \text{if } u = q \\ z_{low} - u\frac{z_{low}}{z_{high}} & \text{otherwise.} \end{cases}$$

Leaves of the tree do not contribute to the fitness function. In this thesis, q is taken as 3; hence, the tree is ternary.

Hierarchical Trap-3 (T7)

For a non-leaf node r except the root node, the contribution function is calculated by taking $z_{high} = z_{low} = 1$ multiplied by $3^{height(r)}$. For the root node, the contribution function is calculated by taking $z_{high} = 1$ and $z_{low} = 0.9$ multiplied by $3^{height(root)}$. The optimum is the string with all 1s. Figure 4.5 shows the calculation of hierarchical trap function.

MOP	Constituting problems	Description
MOP#1	T3 and T4	Multiple interleaved MDP
MOP#2	T5 and T6	Multiple interleaved trap-5
MOP#3	T7 and T2	Hierarchical trap and zeromax
MOP#4	T8 and T2	Hierarchical deceptive trap and zeromax

Table 4.1: Table illustrating multiobjective problems used in the thesis.

Hierarchically Deceptive Trap-3 (T8)

For all non-root nodes r, the contribution function is calculated by taking $z_{high} = 1$ and $z_{low} = 1 + \frac{0.1}{3}$. For the root node, the contribution function is calculated using $z_{high} = 1$ and $z_{low} = 0.9$. Hence, the bias toward a solution with all zeros is increased and this makes this function more difficult to solve. The optimum of this function is the string with all 1s.

4.4.4 Multiobjective Test Problems

Multiobjective problems are created by taking various combinations of single objective functions described in the previous section (T1 to T8), and these multiobjective problems are listed in Table 4.1.

Table 4.1 shows that MOP#1 is a multiobjective problem consisting of two fitness functions, T3 and T4. Similarly, other multiobjective problems are described in the table 4.1. MOP#3 and MOP#4 have only two points in the Pareto optimal front. One point is the optima of the zeromax function and the other point is the optima of the hierarchical function.

4.5 Summary

In this chapter, a brief overview of existing multiobjective test functions was given. Multiobjective test functions were designed that will be used to test the competent MOGAs developed later in this thesis.

Chapter 5

Multiobjective Bayesian Optimization Algorithm

The previous chapter designed multiobjective test functions, and in this chapter, we develop a competent multiobjective GA and test its performance on those multiobjective test functions. The competent multiobjective GA developed in this chapter borrows the BB identification and mixing capabilities of a competent GA and the multiobjective selection process of a MOGA. Specifically, BOA (Pelikan, Goldberg, & Cantú-Paz, 1999) is coupled with the selection technique of NSGA-II (Deb, Agrawal, Pratap, & Meyarivan, 2000) to form a multiobjective competent GA called the multiobjective Bayesian optimization algorithm (mBOA), details of which are given in the next section.

5.1 Multiobjective Bayesian Optimization Algorithm

mBOA is identical to BOA explained in chapter 3 except that we replace the selection procedure by the nondominated sorting and selection mechanism of NSGA-II. Algorithmically, mBOA may be written as follows:

1. Generate an initial random population.

2. Perform selection,

   (a) Compute ranks of solutions in the population by performing the nondominated

sorting. Also, calculate the crowding distance for each solution in the population. Rank and crowding distance calculations are identical to those of NSGA-II as described in chapter 2.

(b) Select good solutions from the current population based on rank and crowding distance as described in chapter 2. Prefer solutions with the better (lower) rank and the higher crowding distance.

3. Construct a network, using the chosen metric, from the selected members of the population in a similar manner as in BOA, described in chapter 3.

4. Generate a set of new strings O(t) according to the joint distribution encoded by the network in a similar manner as in BOA, described in chapter 3.

5. Combine the parent and the child populations to form a combined population.

6. Calculate the rank and the crowding distance of each member of the combined population.

7. Select the best half of the combined population (based on rank and crowding distance) and put it in the new population. Selection of the best half of the combined population is identical to that of NSGA-II described in chapter 2.

8. Repeat steps (3) to (7) until one or more termination criteria are met.

The selection procedure of NSGA-II gives mBOA the capability to handle multiple objectives. The model building and model sampling technique of BOA provide mBOA with the necessary tools for linkage identification and propagation. mBOA is tried on different adversarially designed test functions, which are described in the following section.

MOP	Description	Problem size (ℓ)	# of points with distinct fitness values on the Pareto optimal front
MOP#1	T3 and T4 (MDP)	30	16
		90	46
MOP#2	T5 and T6 (trap-5)	30	7
		60	13
		90	19
MOP#3	T7 and T2 (hierarchical trap and zeromax)	27	2
MOP#4	T8 and T2 (hierarchical deceptive trap and zeromax)	27	2

Table 5.1: Table listing multiobjective problems used to test mBOA.

MOP	Algorithm	Problem size (ℓ)	# of fitness function evaluations	Population-size	# of generations
MOP#1	simple MOGA	30	9000	150	60
		90	54000	300	180
	mBOA	30	9000	300	30
		90	54000	600	90
MOP#2	simple MOGA	30	3000	50	60
		60	24000	200	120
		90	42120 *	234	180
	mBOA	30	3000	300	10
		60	24000	400	60
		90	42000	600	70
MOP#3	simple MOGA	27	2100000	3000	700
	mBOA	27	2100000	3000	700
MOP#4	simple MOGA	27	2100000	3000	700
	mBOA	27	2100000	3000	700

Table 5.2: Table listing experimental setups for simple MOGA and mBOA for different multiobjective test problems. Number of fitness function evaluations are kept same for both algorithms. In case of $*$, the number of fitness function evaluations is slightly higher for simple MOGA to keep the population size an integer.

Point No.	1	2	3	...	m	$m+1$
No. of solutions	$\binom{m}{0}$	$\binom{m}{1}$	$\binom{m}{0}$	...	$\binom{m}{m-1}$	$\binom{m}{m}$

Table 5.3: Table showing number of solutions at each point of the Pareto optimal front for MOP#1 and MOP#2. "Point No." refers to the point on the Pareto optimal front, "No. of solutions" is the number of solutions with the same fitness values corresponding to that particular point, and m is the number of building blocks.

5.2 Default Experimental Setup

To test mBOA, the multiobjective problems designed in chapter 4 are used and these multiobjective problems are listed in table 5.1. As shown in table 5.1, MOP#1 consists of two multiple interleaved minimal deceptive functions (described in chapter 4), T3 and T4, and is tried with two problem sizes, $\ell = 30$ and $\ell = 90$. Similarly, other multiobjective problems are described in table 5.1. The fourth column of table 5.1 shows the number of different points on the Pareto optimal front. There is a subtle difference between the points on the Pareto optimal front and the solutions belonging to the Pareto optimal front. Since, each function of the multiobjective problem is a function of unitation, each point of the Pareto optimal front may have more than one binary string (individual) corresponding to it. Hence, the total number of different solutions that belong to the Pareto optimal front will be more than the value shown in the fourth column of table 5.1. The next paragraph explain this in more detail.

In the case of MOP#1 (T3 and T4) and MOP#2 (T5 and T6), combinations of complementary functions are taken. If the problem is composed of m subfunctions (see chapter 4) of order k (here, $\ell = mk$), then each end point of the Pareto optimal front has $\binom{m}{0}$ distinct solutions (with same fitness values) corresponding to it. Similarly, each point next to end points has $\binom{m}{1}$ distinct solutions (with same fitness values) corresponding to it, and so on. The Pareto optimal front has a total of $(m+1)$ number of points. Let us take an example of MOP#2 that is formed of two trap-5 functions with $\ell = 30$. The number of building blocks m is equal to 6. There are two optimal building blocks, 00000 and 11111, respectively. A

Point No.	No. of solutions	No. of 00000 building blocks	No. of 11111 building blocks
1	$\binom{6}{0}$	6	0
2	$\binom{6}{1}$	5	1
3	$\binom{6}{2}$	4	2
4	$\binom{6}{3}$	3	3
5	$\binom{6}{4}$	2	4
6	$\binom{6}{5}$	1	5
7	$\binom{6}{6}$	0	6

Table 5.4: Table listing the number of Pareto optimal solutions in MOP#2 with $\ell = 30$. "Point No." represents the distinct points on the Pareto optimal front; "No. of solutions" represents the number of Pareto optimal solutions corresponding to the particular point on the Pareto optimal front; "No. of 00000 building blocks" represents the number of 00000 building block present in the solutions; "No. of 11111 building blocks" represents the number of 11111 building block present in the solutions. Note that the point no. 1 and point no. 7 are the end points of the Pareto optimal front.

Pareto optimal solution consists of 6 building blocks where each building block can either be 00000 or 11111. Therefore, the Pareto optimal solutions consists of combinations of these two (00000 and 11111) optimal building blocks. The end points are the points with all 6 building blocks being the same. Hence, one end point corresponds to the string with all 30 bits equal to 0; and the other end point corresponds to the string with all 30 bits equal to 1. The point next to the end points will have 5 out of 6 building blocks same and the sixth building block different. There are $\binom{6}{1}$ solutions that will have 5 out of 6 building blocks as 00000 and the sixth building block as 11111. Also, there are $\binom{6}{1}$ solutions that will have 5 out of 6 building blocks as 11111 and the sixth building block as 00000. The table 5.4 shows the number of solutions corresponding to different points on the Pareto optimal front for the case of MOP#2 with $\ell = 30$.

In the middle of the Pareto optimal front there are more number of distinct solutions with same fitness function values as opposed to the end points where there is only one solution at each point. The exact number of solutions at each point on the Pareto optimal front is shown in table 5.3. Hence, given a finite size population, the probability of having a string belonging to the middle region of the Pareto optimal front is higher than the probability of

having a string belonging to the end. Therefore, in the case of MOP#1 and MOP#2, points near the ends are more difficult to obtain than points near the center of the Pareto optimal front. Also, there are exponentially large number ($\approx 2^m$) of distinct solutions belonging to the Pareto optimal front; hence, we cannot obtain all the solutions belonging to the Pareto optimal front without using exponentially large population size. Both the algorithms, simple MOGA and mBOA, use niching in the fitness function space instead of variable space. So, we aim to obtain representative samples of all distinct function value combinations on the Pareto optimal front. In other words, our objective is to obtain and consistently maintain a good spread of solutions over the true Pareto optimal front.

To compare the performance of mBOA, we develop a simple MOGA by incorporating the multiobjective selection scheme of NSGA-II in a simple single-objective binary GA. One-point crossover with crossover probability, p_c, equal to 0.9 and bit-wise mutation with mutation probability, p_m, equal to 0.01 is used (see chapter 2 for a brief explanation of one-point crossover and bit-wise mutation). Since mBOA and simple MOGA work on different underlying principles, we run each with different population sizes and number of generations while keeping the total number of fitness evaluations same for both the algorithms. The population size and the number of generations for mBOA were set to the value that was suitable for the harder of the two single objective functions that formed the multiobjective problem. Therefore, in case of MOP#2 that consists of two trap functions, we set the population size to the value that will be suitable for a single objective BOA for solving trap-5 function. Simple MOGA was tried with many different combinations of population size and number of generations keeping the number of fitness function evaluations constant and equal to the number of fitness function evaluations for mBOA. The best result obtained out of different experimental settings was compared with the result of mBOA. Both algorithms were run until the allotted number of fitness function evaluations were expended. Table 5.2 shows the experimental settings for simple MOGA and mBOA.

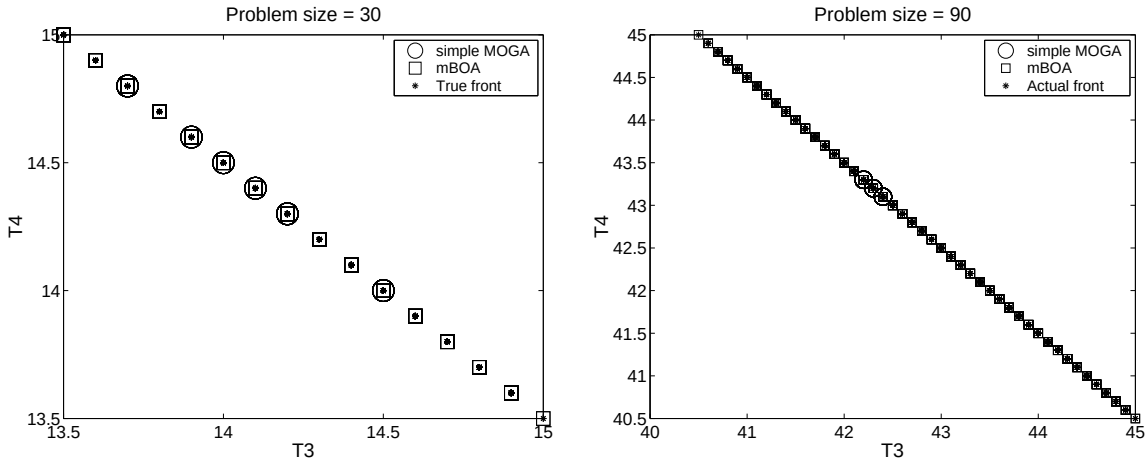


Figure 5.1: Figure comparing the performance of simple MOGA and that of mBOA on MOP#1 for $\ell = 30$ and $\ell = 90$. $*$ shows the true Pareto optimal front. $\square$ shows the nondominated set of the population at the final generation obtained by mBOA. $\bigcirc$ corresponds to the nondominated set of the population at the final generation obtained by simple MOGA. With the increase in the problem size, the performance of simple MOGA decreases, while the performance of mBOA stays the same. Note that simple MOGA was not able to discover the end points for both the cases ($\ell = 30$ and $\ell = 90$).

5.3 Results

This section discusses the performance of mBOA on the multiobjective test problems defined in the previous section. The results are compared with the performance of simple MOGA on the same test functions.

5.3.1 MOP#1 (T3 and T4)

In both cases ($\ell = 30$ and $\ell = 90$), mBOA discovered representative samples at all Pareto optimal function values combinations on the Pareto optimal front including the end points, while the simple MOGA was not able to do so. Figure 5.1 compares the performance of simple MOGA and that of mBOA on MOP#1 with $\ell = 30$ and $\ell = 90$. In both cases ($\ell = 30$ and $\ell = 90$), simple MOGA converged to only few points of the true Pareto optimal front and was unable to discover the end points of the Pareto optimal front. The building blocks of the problem become more loosely linked with the increase in problem size; therefore, the

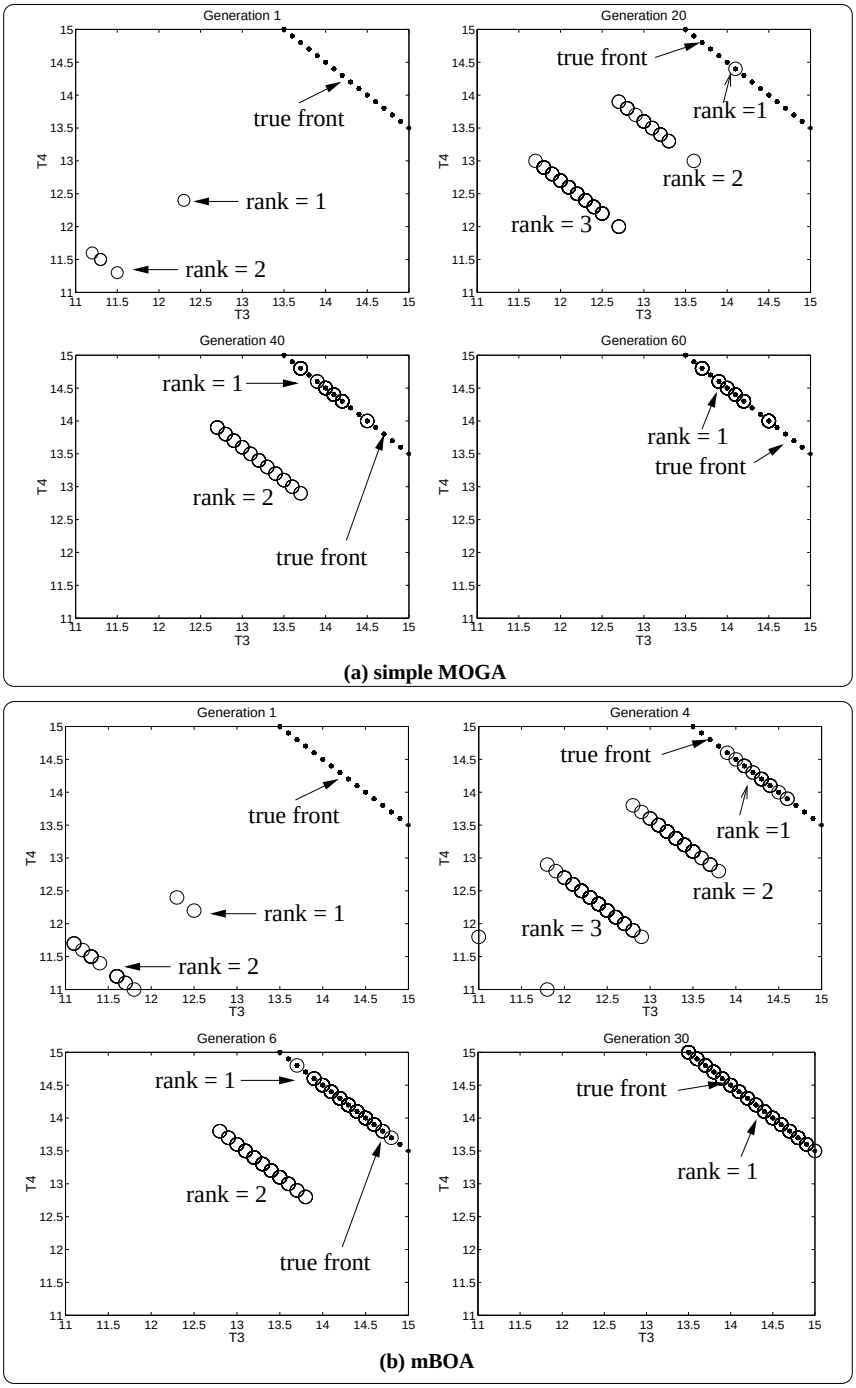


Figure 5.2: The figure shows the evolution of a solution in the simple MOGA and mBOA on MOP#1 with $\ell = 30$. $\bigcirc$ denotes the population at the respective generation and $*$ represents the true Pareto optimal front. Note, not all members of the population are shown due to lack of space.

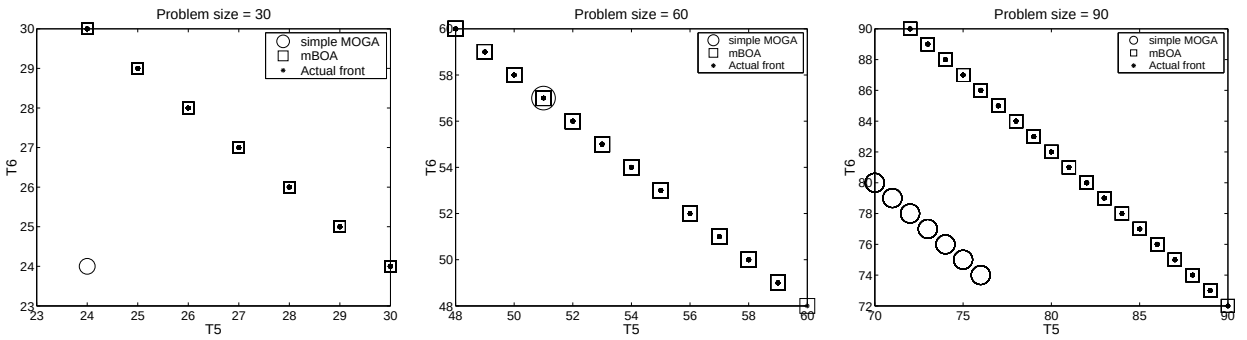


Figure 5.3: Figure comparing the performance of simple MOGA and that of mBOA on MOP#2 (T5 and T6) for $\ell = 30$, $\ell = 60$, and $\ell = 90$. $*$ shows the true Pareto optimal front. $\square$ shows the nondominated set of the population at the final generation obtained by mBOA. $\bigcirc$ corresponds to the nondominated set of the population at the final generation obtained by simple MOGA.

performance of simple MOGA deteriorates considerably with the increase in problem size. On the other hand, the performance of mBOA is not affected by the loosely linked building blocks and mBOA consistently performs well at the increased problem size as shown in figure 5.1. Figure 5.2 shows populations at various generations during the run for simple MOGA and mBOA on MOP#1 with $\ell = 30$. Simple MOGA was unable to discover all the points of the Pareto optimal front, whereas mBOA discovered all the points on the Pareto optimal front including the end points.

5.3.2 MOP#2 (T5 and T6)

As in MOP#1, simple MOGA performed worse than mBOA on MOP#2. For $\ell = 30$ and $\ell = 90$, simple MOGA could not discover the true Pareto optimal front. For $\ell = 60$, it converged to one of the points of the true Pareto optimal front. mBOA, in all cases, converged to the true Pareto optimal front and consistently maintained a good spread over it. Figure 5.3 compares the performance of the two algorithms on MOP#2 with $\ell = 30$, $\ell = 60$, and $\ell = 90$. Figure 5.4 shows the populations at the first, the last, and two intermediate generations for MOP#2 with $\ell = 30$.

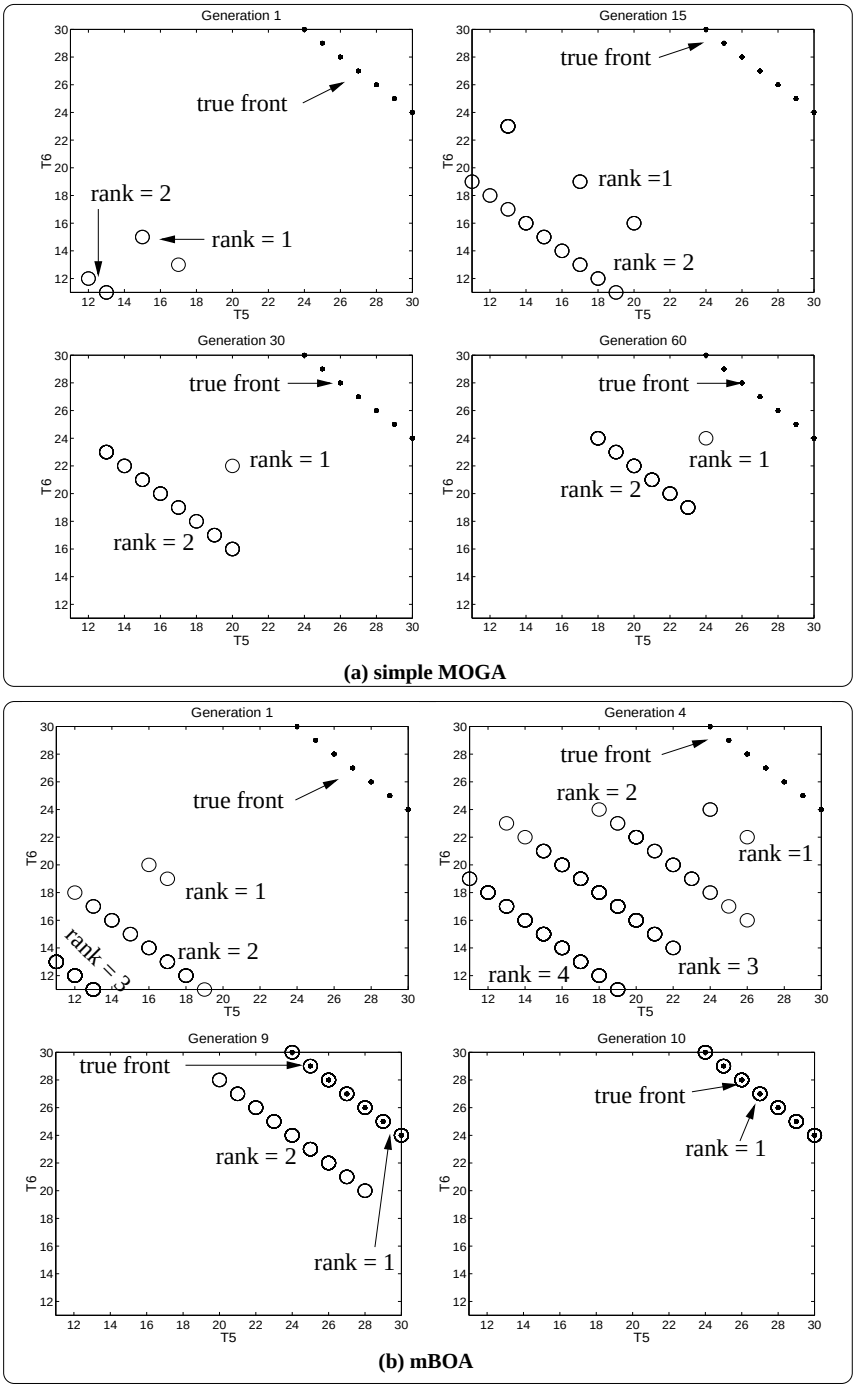


Figure 5.4: The figure shows the evolution of a solution in the simple MOGA and mBOA on MOP#2 with $\ell = 30$. ◯ denotes the population at the respective generation and $*$ represents the true Pareto optimal front. Note, not all members of the population are shown due to lack of space.

Problem	Problem size (ℓ)	No. of points on the Pareto front with distinct fitness values			No. of different solutions on the Pareto front		
		Actual No. of points	simple MOGA	mBOA	Actual No. of strings	simple MOGA	mBOA
MOP#1	30	16	6	16	2^{15}	7	224
	90	46	3	46	2^{45}	5	591
MOP#2	30	7	0	7	2^{6}	0	30
	60	13	1	13	2^{12}	1	102
	90	19	0	19	2^{18}	0	327
MOP#3	27	2	1	1	2	1	1
MOP#4	27	2	1	1	2	1	1

Table 5.5: Table illustrating results of simple MOGA and mBOA on the multiobjective test functions.

5.3.3 MOP#3 (T7 and T2) and MOP#4 (T8 and T2)

Both MOP#3 and MOP#4 consist of combinations of a harder hierarchical and an easier zeromax problems. The local optimum of the harder hierarchical problem is only slightly worse than its global optimum and is the same as the global optimum of the easier zeromax problem. The true Pareto optimal front has two points, corresponding to the global optima of the two functions. As expected, in both the cases (MOP#3 and MOP#4), both the algorithms failed to discover the optima of the harder hierarchical function and converged to the optima of the easier zeromax function.

The results of the experiments are summarized in the table 5.5. In case of MOP#1 and MOP#2, mBOA found all the points on the true Pareto optimal front, whereas simple MOGA often times failed to do so. Also, mBOA maintains a larger number of distinct nondominated Pareto optimal solutions than simple MOGA.

5.4 Summary

The chapter combined a competent GA and a multiobjective GA to design a multiobjective competent GA called mBOA. mBOA was compared with simple MOGA on a series of multiobjective test functions. mBOA outperforms simple MOGA on the deceptive problems with loose linkage. As expected, neither simple MOGA nor mBOA could handle problem with multilevel interactions as they do not have any niching or chunking capabilities. Next chapter addresses the issue of hierarchical problem solving in detail.

Chapter 6

Multiobjective Hierarchical Bayesian Optimization Algorithm

The results of the previous chapter were mixed. On the multiobjective simple deceptive problems, mBOA was able to evolve a good spread of solutions on the true Pareto optimal front, whereas simple MOGA failed to do so. However, in hierarchically difficult problems, both algorithms were misled to local optima. The purpose of this chapter is to develop an algorithm that would not only solve problems that could be solved using mBOA, but also solve hierarchically difficult problems.

The next section designs and implements variants of mBOA incorporating niching alone, and tests them on multiobjective hierarchical problems. Later, a multiobjective hierarchical problem solver called multiobjective hierarchical Bayesian optimization algorithm (mhBOA) is designed, implemented, and tested. The performance of mhBOA on the multiobjective test functions is compared with that of mBOA.

6.1 Variants of mBOA with Niching Only

Pelikan and Goldberg (2000) suggested that to solve a hierarchical problem, an algorithm must contain linkage learning, niching, and chunking capabilities. While, mBOA contains linkage learning capabilities, it does not contain any chunking capabilities. The mBOA does have crowding-based niching incorporated into it. However, the primary purpose of the

MOP	Description	Problem size (ℓ)	# of points on the Pareto optimal front
MOP#3	T7 and T2 (hierarchical trap and zeromax)	27	2
MOP#4	T8 and T2 (hierarchical deceptive trap and zeromax)	27	2

Table 6.1: Table illustrating multiobjective problems used to test variants of mBOA with niching only.

crowding distance comparator used in the selection process of mBOA is to perform niching in the fitness function space, whereas a hierarchical multiobjective problem solver would also need to maintain diversity in the genotypic space. Hence, the crowding distance based niching of mBOA is not sufficient for hierarchical problem solving. While it is easy to see that a multiobjective hierarchical problem solver has to contain a niching operator, it is unclear whether it should also contain chunking capabilities.

Therefore, we incorporate niching into variants of mBOA and test them on multiobjective hierarchical problems shown in table 6.1. Specifically, we incorporate the following nichers:

- Restricted tournament selection (RTS) (Harik, 1995).
- Crowding distance in genotypic space.
- Fitness sharing (Goldberg & Richardson, 1987).

Next, implementations and results of each of the algorithms are discussed. All the algorithms were run for 700 generations with a population size of 3000.

6.1.1 Variant of mBOA + RTS: Algorithm 1

Restricted tournament selection (RTS) (Harik, 1995) was used by Pelikan and Goldberg (2001) as a nicher in hBOA. In RTS, for every offspring, w individuals are picked from the

current population and the offspring competes with the closest individual where closeness is measured by the genotypic or the phenotypic distance. The offspring is copied to the next generation if it is better than its competitor else the parent is copied to the next generation. The parameter w is called window size.

As in the original RTS, we also randomly select w individuals from the parent population for every offspring. Out of these w individuals, the individual closest—in terms of genotypic distance—to the offspring is selected and the offspring competes with it. If the offspring has a higher rank than its competitor, it is copied into the next generation. If both the offspring and its competitor has the same rank then the individual with higher crowding distance is the winner. In case if the parent has a higher rank, the parent is the winner. Two window sizes were tried: window size equal to the population size and window size equal to the problem size.

RTS can be used with a variant of mBOA by performing the following steps:

1. Randomly generate n solutions.
2. Compute the rank and the crowding distance of each member of the population.
3. Perform the selection based on rank and crowding distance.
4. Build a probabilistic model (Bayesian network) of the promising solutions.
5. Sample new solutions using the Bayesian network.
6. Perform a nondominated sorting and compute the rank and the crowding distance considering parent and child populations together.
7. For every offspring randomly pick w parents and compare the offspring with the closest parent using rank and crowding distance and put the winner in the new population.
8. Go to step (3) and repeat the process until some convergence criteria are satisfied.

The algorithm described above failed on both the multiobjective problems, MOP#3 and MOP#4, and converged to the optimum of the zeromax problem in both the cases.

6.1.2 Variant of mBOA + Genotypic Crowding Distance: Algorithm 2

The crowding distance comparator used in the selection scheme of NSGA-II ensures only a good spread of solutions along the Pareto optimal front and it does not ensure genotypic diversity preservation in the population. Hence in this variant of mBOA, crowding distance for an individual was replaced by the sum of genotypic distances of the individual from all other members of the population and is called genotypic crowding distance. This will ensure genotypic diversity preservation in the whole population.

Genotypic crowding distance can be used with a variant of mBOA by performing the following steps:

1. Randomly generate n solutions.
2. Compute rank and genotypic crowding distance of each solution in the population.
3. Perform the selection based on rank and genotypic crowding distance.
4. Build a probabilistic model (Bayesian network) of the promising solutions.
5. Sample new solutions using the Bayesian network.
6. Combine both the parent and the offspring populations to form a combined population.
7. Perform a nondominated sorting and compute the rank of each member of the combined population.
8. Calculate the genotypic crowding distance of each individual in the combined population.

9. Select the n *best* (based on the rank and the genotypic crowding distance) solutions from the combined population.

10. Go to step (4) and repeat the process until some convergence criteria are satisfied.

Similar to algorithm 1, this algorithm also failed on both the multiobjective problems and converged to the optimum of the zeromax problem in both the cases.

6.1.3 Variant of mBOA + Fitness Sharing: Algorithm 3

Fitness sharing (Goldberg & Richardson, 1987) is introduced to incorporate niching in the variant of mBOA. A sharing parameter is specified and niche count—the number of individuals in the population that are within sharing parameter distance from the current individual—of each individual in the population is calculated. The rank of each individual is calculated based on its shared fitness. The crowding distance of a solution is replaced by its niche count multiplied by minus one (higher crowding distance corresponds to lower niche count).

Fitness sharing can be used with a variant of mBOA by performing the following steps:

1. Randomly generate n solutions.

2. Compute the niche count of each individual in the population.

3. Compute the shared fitness of each individual in the population using its niche count.

4. Compute the rank of each individual using its shared fitness value.

5. Perform the selection using rank and niche count (lower niche count is preferred).

6. Build a probabilistic model (Bayesian network) of the promising solutions.

7. Sample new solutions using the Bayesian network.

8. Form a combined population by combining both the parent and the offspring population.

9. Compute the niche count of each individual in the combined population and then calculate its shared fitness.

10. Perform a nondominated sorting and compute ranks using shared fitness values.

11. Select the *n best* (based on the rank and niche count) solutions of the combined population.

12. Go to step (6) and repeat the process until some convergence criteria are satisfied.

The results were qualitatively similar to the other two algorithms and it also failed on both the test problems.

Our pilot study showed that all the three algorithms (Algorithm 1, Algorithm 2, and Algorithm 3) failed to solve multiobjective hierarchical test problems. The failure of these algorithms empirically shows that similar to the single objective hierarchical problem solver, a multiobjective hierarchical problem solver needs both niching and chunking along with linkage learning capabilities. Therefore, in the following section, we propose multiobjective hierarchical Bayesian optimization algorithm (mhBOA) which borrows the linkage learning, niching, and chunking capabilities of hBOA (Pelikan & Goldberg, 2001) and nondominated sorting selection procedure of NSGA-II (Deb, Agrawal, Pratap, & Meyarivan, 2000).

6.2 Multiobjective Hierarchical Bayesian Optimization Algorithm

This section provides an algorithmic description of multiobjective hierarchical Bayesian optimization algorithm (mhBOA):

1. Randomly generate an initial population P(0).

2. Perform selection:

   (a) Compute ranks of solutions in the population by performing the nondominated sorting. Also, calculate the crowding distance for each solution in the population. Rank and crowding distance calculations are identical to those of NSGA-II as described in chapter 2.

   (b) Select good solutions from the current population based on rank and crowding distance as described in chapter 2. Prefer solutions with the better (lower) rank and the higher crowding distance. Call the selected population as P1(t).

3. Build the network by constructing decision graphs encoding conditional probability in a similar manner as in hBOA described in chapter 3.

4. Generate child population C(t) by using the network with decision graphs in a similar manner as in hBOA described in chapter 3.

5. Calculate rank and crowding distance of each member considering C(t) and P(t) together.

6. Create a new population P(t+1) by using RTR to incorporate each member of C(t) into P(t) in a similar manner as in hBOA described in chapter 3. Solutions are compared on the basis of their rank and crowding distance values.

7. Repeat steps (2) to (6) until one or more convergence criteria are satisfied.

Therefore, mhBOA is the extension of single objective hBOA to multiobjective domain and is created by replacing the single objective selection scheme of hBOA by the multiobjective selection procedure of NSGA-II.

MOP	Description	Problem size (ℓ)	# of points on the Pareto optimal front with distinct fitness values
MOP#1	T3 and T4 (MDP)	30	16
MOP#2	T5 and T6 (trap-5)	30	7
		60	13
MOP#3	T7 and T2 (hierarchical trap and zeromax)	27	2
MOP#4	T8 and T2 (hierarchical deceptive trap and zeromax)	27	2

Table 6.2: Table illustrating multiobjective problems used to test mhBOA.

MOP	Algorithm	Problem size (ℓ)	# of fitness function evaluation	Population-size	# of generations
MOP#1	mBOA	30	25000	2500	10
	mhBOA	30	750000	2500	300
MOP#2	mBOA	30	2400	300	8
		60	30000	1500	20
	mhBOA	30	12000	300	40
		60	375000	1500	250
MOP#3	mBOA	27	7200	600	12
	mhBOA	27	15600	600	26
MOP#4	mBOA	27	9000	600	15
	mhBOA	27	15000	600	25

Table 6.3: Table illustrating the experimental setups of mBOA and mhBOA for different multiobjective test problems.

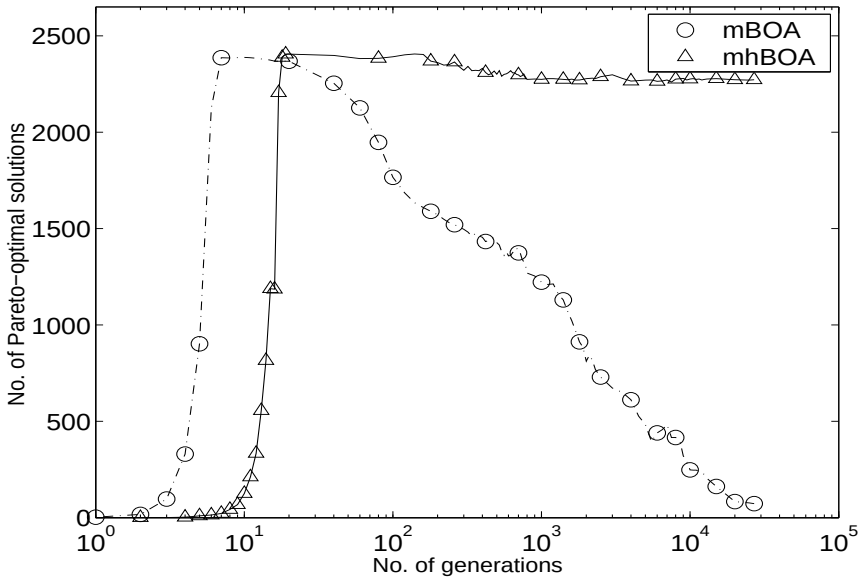


Figure 6.1: Figure comparing number of distinct solutions belonging to the true Pareto optimal front at different generation during runs of mBOA and mhBOA on MOP#1 with $\ell = 30$. mBOA is able to evolve a large number of Pareto optimal solutions faster than mhBOA, but, it is not able to consistently maintain them over the run. Note that the graph is on log scale.

6.3 Experiments and Results

The performance of mhBOA was tested on multiobjective test problems listed in table 6.2. MOP#1 and MOP#2 are multiobjective deceptive problems with loosely linked building blocks, whereas MOP#3 and MOP#4 are multiobjective hierarchical problems. The results of mhBOA are compared with those of mBOA. The population size was set to the same value for both the algorithms. The experimental setting for both the algorithms is shown in the table 6.3. This section discusses the results of mBOA and mhBOA for each of the multiobjective test problem.

6.3.1 MOP#1 (T3 and T4)

Figure 6.1 shows the number of solutions belonging to the Pareto optimal front at various generations in the test runs of mBOA and mhBOA on MOP#1 with $\ell = 30$. Both the algorithms are able to achieve a large number ($\approx$2300) of Pareto optimal solutions. The

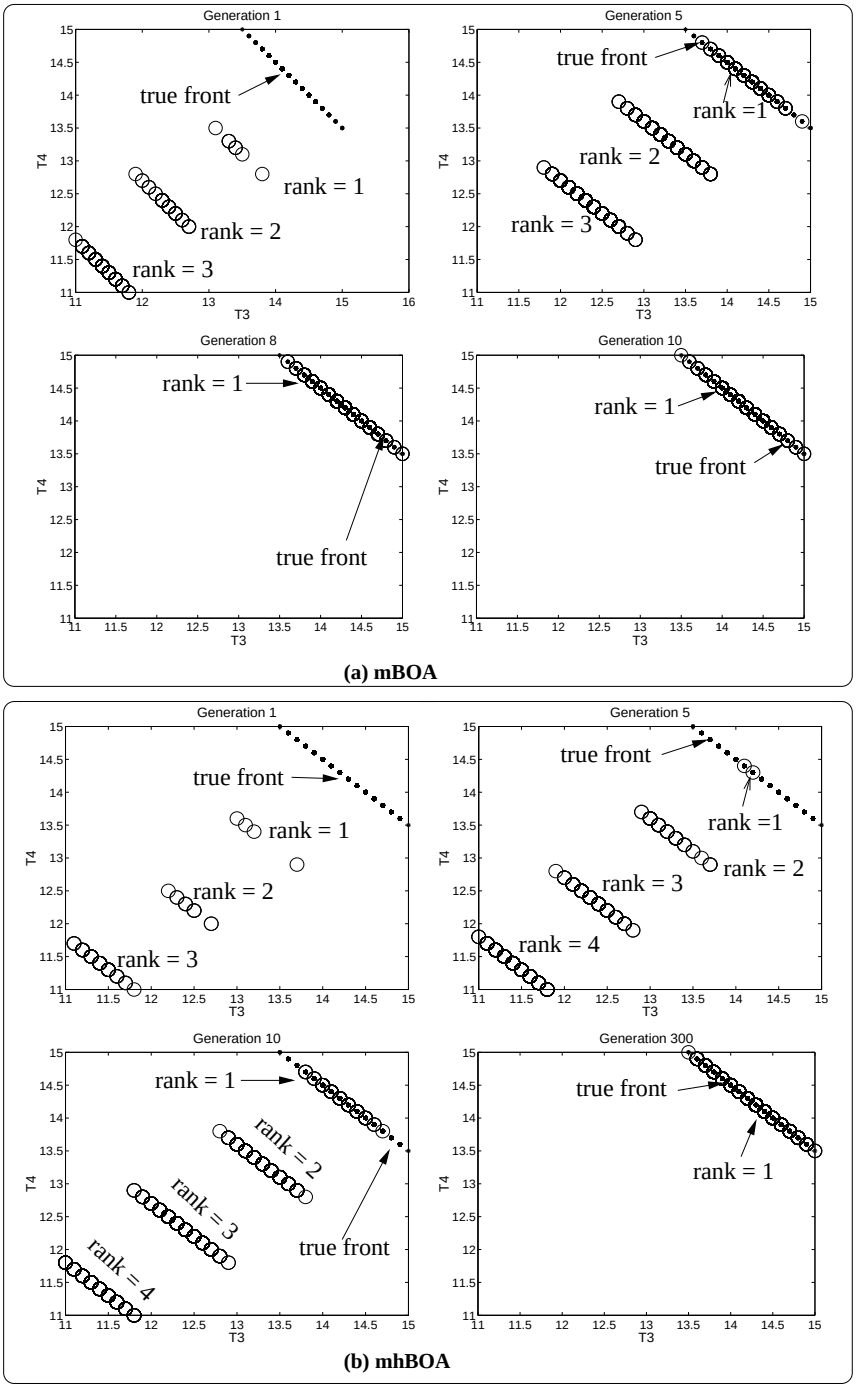


Figure 6.2: The figure shows the evolution of a solution in the mBOA and mhBOA on MOP#1 with $\ell = 30$. $\bigcirc$ denotes the population at the respective generation and $*$ represents the true Pareto optimal front. mBOA is able to converge to the true Pareto optimal front faster than mhBOA. Note, not all members of the population are shown due to lack of space.

problem is a simple deceptive function with loose linkage; hence, there is no need for the more complicated model that is used by mhBOA. Hence, mBOA is able to converge to the true Pareto optimal front faster than mhBOA. However, mBOA is not able to consistently maintain all the diverse solutions over the run. The crowding distance based niching operator of mBOA is sufficient only to maintain diversity in the fitness function space and is unable to maintain diversity in the genotypic space as it does not reward any genotypic diversity. Hence, in the case of mBOA, over the period of time, there is a gradual loss of diversity in the genotypic space, whereas mhBOA is able to consistently maintain all the diverse solutions because of its niching operator that maintains diversity in the genotypic space.

Both the algorithms (mBOA and mhBOA) are able to converge to the true Pareto optimal front. Figure 6.2 shows the population at various generations during the runs of mBOA and mhBOA. mBOA is faster to converge than mhBOA; hence, requires less number of fitness function evaluations to capture the whole front. However, mhBOA is able to find a higher number of distinct solutions belonging to the Pareto optimal front.

6.3.2 MOP#2 (T5 and T6)

In both the cases ($\ell = 30$ and $\ell = 60$), both the algorithms (mBOA and mhBOA) are able to converge to the true Pareto optimal front. Figure 6.3 and figure 6.4 show populations at various generations during the runs of mBOA and mhBOA on MOP#2 with $\ell = 30$ and $\ell = 60$, respectively. mBOA is faster to converge than mhBOA; hence, requires less number of fitness function evaluations to capture the whole front. However, mhBOA is able to find a higher number of distinct solutions belonging to the Pareto optimal front.

6.3.3 MOP#3 (T7 and T2) and MOP#4 (T8 and T2)

In both the cases (MOP#3 and MOP#4), mBOA converged to the global optima of the easier the zeromax function and failed to obtain the other point of the Pareto optimal front,

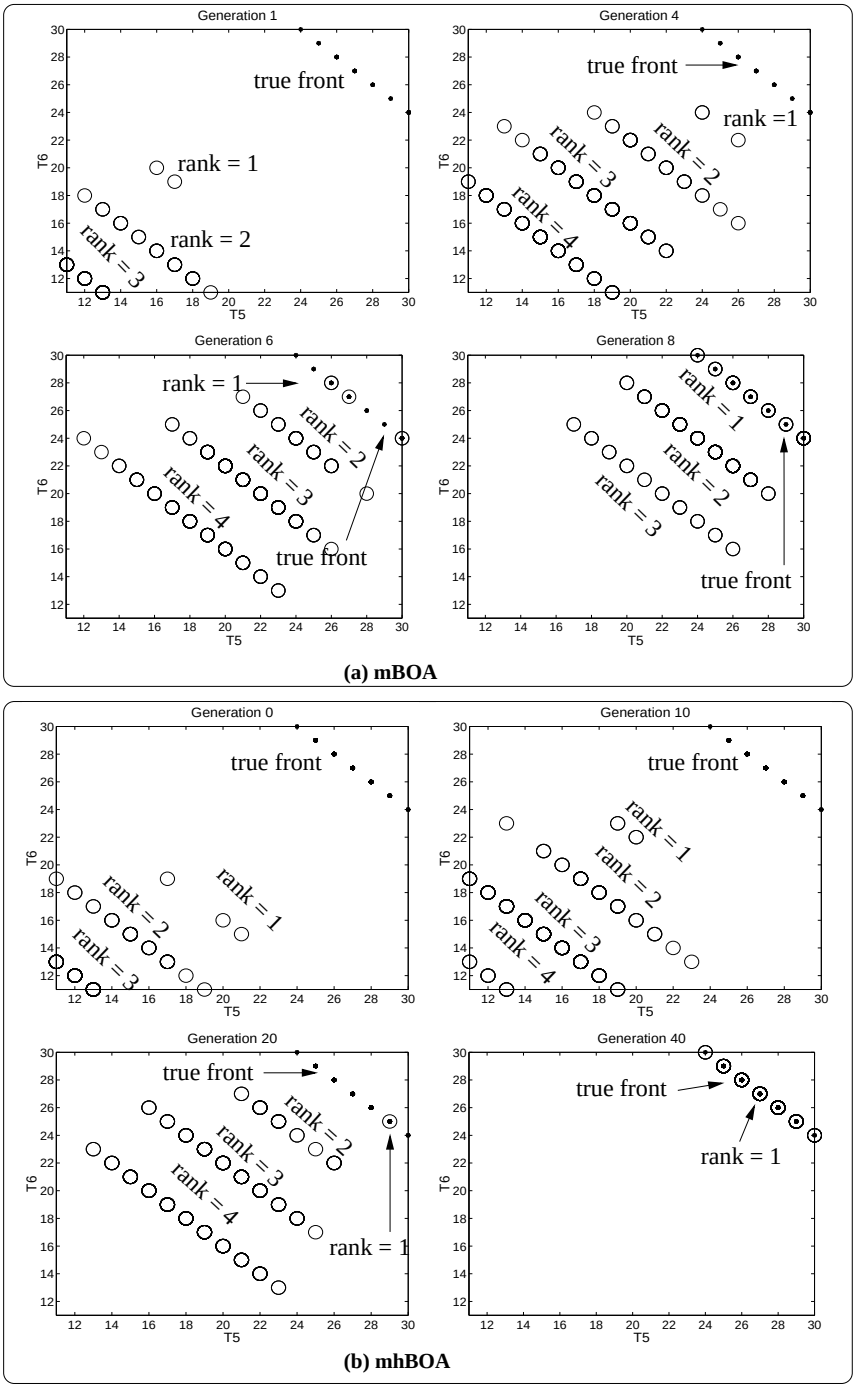


Figure 6.3: The figure shows the evolution of a solution in the mBOA and mhBOA on MOP#2 with $\ell = 30$. $\bigcirc$ denotes the population at the respective generation and $*$ represents the true Pareto optimal front. mBOA is able to converge to the true Pareto optimal front faster than mhBOA. Note, not all members of the population are shown due to lack of space.

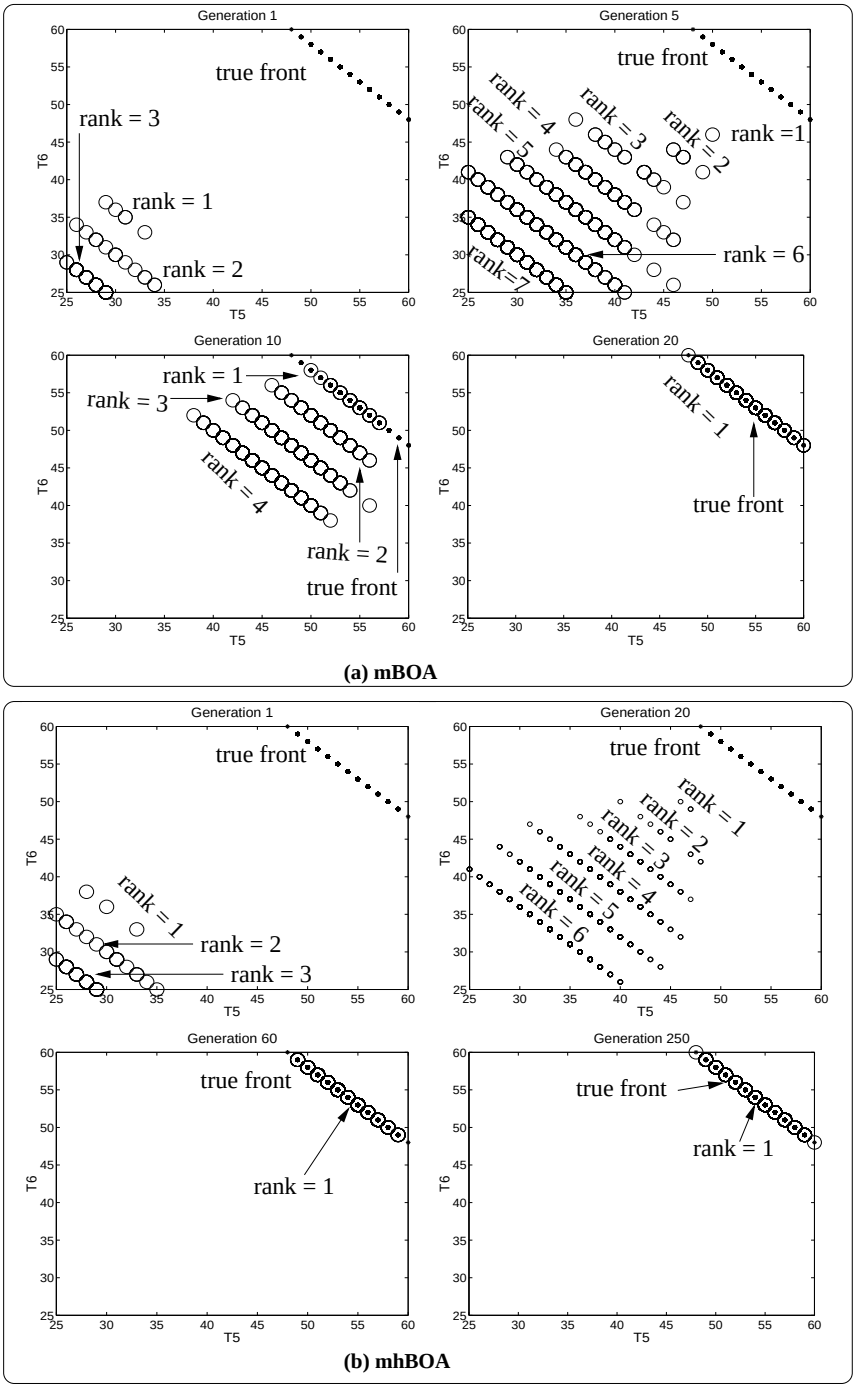


Figure 6.4: The figure shows the evolution of a solution in the mBOA and mhBOA on MOP#2 with $\ell = 60$. $\bigcirc$ denotes the population at the respective generation and $*$ represents the true Pareto optimal front. mBOA is able to converge to the true Pareto optimal front faster than mhBOA. Note, not all members of the population are shown due to lack of space.

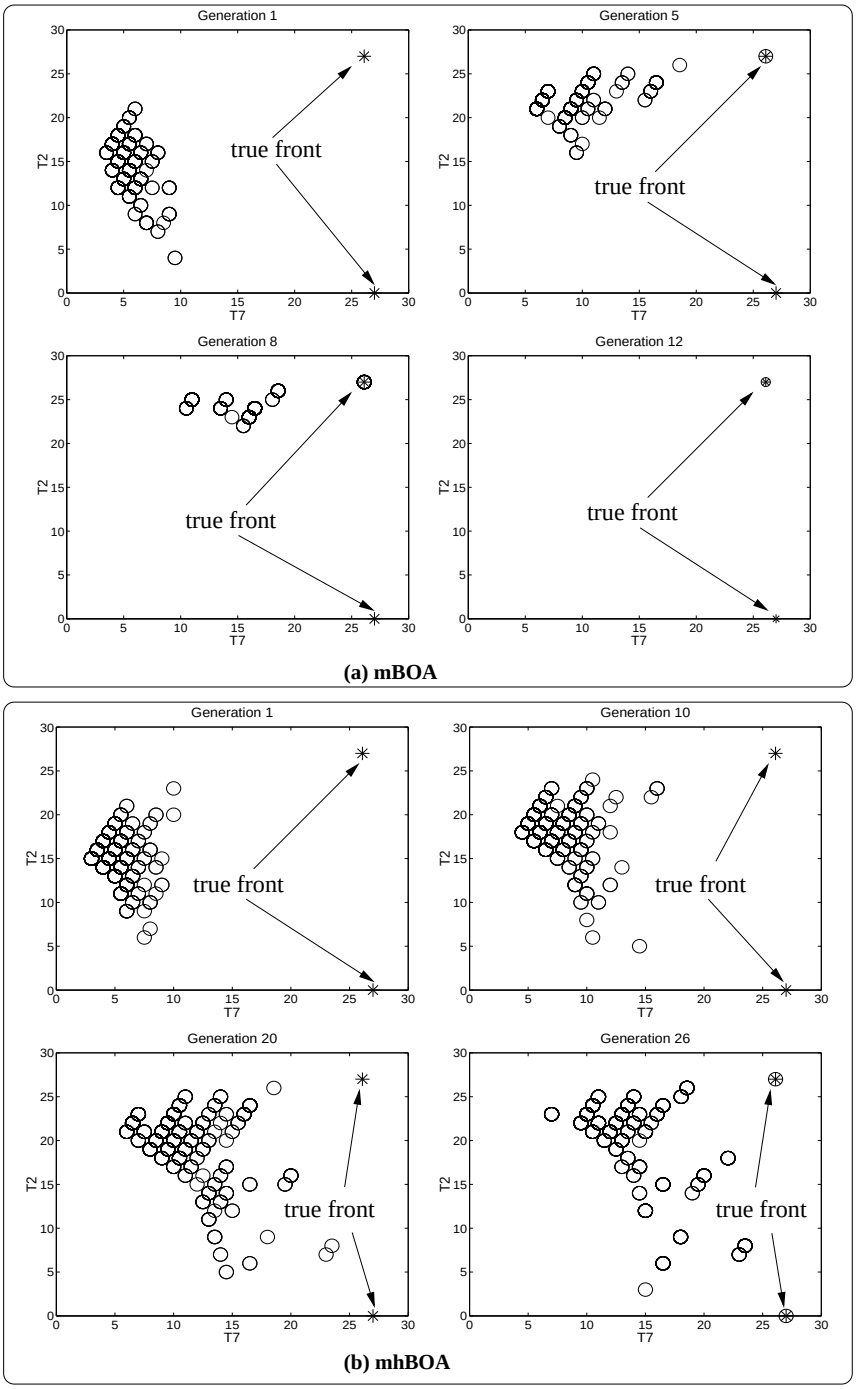


Figure 6.5: The figure shows the evolution of a solution in the mBOA and mhBOA on MOP#3 with $\ell = 27$. $\bigcirc$ denotes the population at the respective generation and $*$ represents the true Pareto optimal front. The true Pareto optimal front consists of only two points. mBOA is not able to find both the points of the true Pareto optimal front and converged to one of them, whereas mhBOA finds both the points of the true Pareto optimal front. Note, not all members of the population are shown due to lack of space.

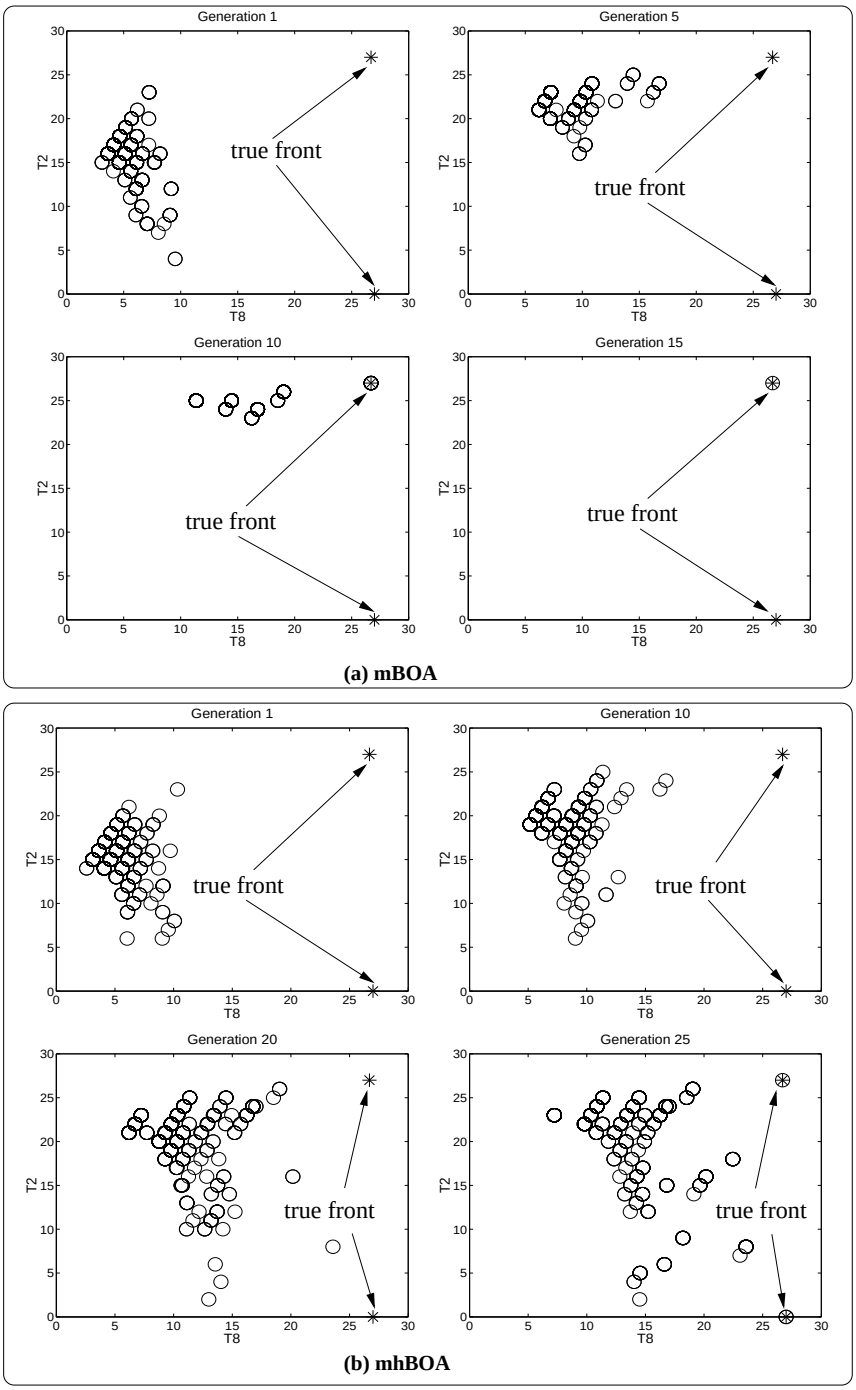


Figure 6.6: The figure shows the evolution of a solution in the mBOA and mhBOA on MOP#4 with $\ell = 27$. $\bigcirc$ denotes the population at the respective generation and $*$ represents the true Pareto optimal front. The true Pareto optimal front consists of only two points. mBOA is not able to find both the points of the true Pareto optimal front and converged to one of them, whereas mhBOA finds both the points of the true Pareto optimal front. Note, not all members of the population are shown due to lack of space.

MOP	(ℓ)	No. of points on the Pareto optimal front with distinct fitness values			No. of different strings on the Pareto front			No. of fitness function evaluations	
		Actual No. of Points	mBOA	mhBOA	Actual No. of strings	mBOA*	mhBOA	mBOA	mhBOA
#1	30	16	16	16	2^{15}	2387	2400	25000	750000
#2	30	7	7	7	2^{6}	19	68	2400	12000
	60	13	13	13	2^{12}	1077	1216	30000	375000
#3	27	2	1	2	2	1	2	-	15600
#4	27	2	1	2	2	1	2	-	15000

Table 6.4: Table illustrating results of mBOA and mhBOA on the multiobjective test problems. The '-' implies that the algorithm is not able to find all the points on the actual Pareto optimal front. In case of $*$, these are peak values of distinct Pareto optimal solutions which disappear if mBOA is run longer as shown in figure 6.1.

whereas mhBOA located both the points of the Pareto optimal front. Figure 6.5 and figure 6.6 show the populations at various generations during the runs of mBOA and mhBOA for MOP#3 and MOP#4, respectively.

The results of mBOA and mhBOA on the test functions are summarized in the table 6.4.

6.4 Summary

In this chapter, a multiobjective hierarchical problem solver was designed, implemented, and tested. The multiobjective hierarchical problem solver is called multiobjective hierarchical Bayesian optimization algorithm (mhBOA) and is constructed by incorporating the multiobjective selection scheme of NSGA-II into the single objective hBOA. mhBOA is able to solve multiobjective hierarchical problems as well as all other problems that could be solved using mBOA.

Chapter 7

Future Work

This chapter discusses a number of future research efforts that will help improve the capabilities of mBOA and mhBOA. First, we discuss work that will verify and improve the performance of mBOA and mhBOA. Then, we discuss theoretical advances to analyze the behavior of the algorithms. Last, we discuss ways to extend the applicability of the algorithms.

Verifying Performance: This thesis tests the performance of mBOA and mhBOA on the multiobjective test function suite designed in chapter 4. However, more rigorous testing of the algorithms on a broader test suite, consisting of a variety of test functions such as constrained multiobjective problems and continuous domain multiobjective problems, is needed to verify the superior performance of mBOA and mhBOA over a simple MOGA.

Moreover, the test functions used in the thesis contain exponentially large number of solutions on the Pareto optimal front. Hence, to capture all the points belonging to the Pareto optimal front, an exponentially large population size is required. Therefore, new test functions, having a tractable number of solutions on the Pareto optimal front, should be designed and the performance of mBOA and mhBOA should be tested on them.

Improving Performance: Attempts to improve the performance of mBOA and mhBOA

by using parallelization, hybridization, and other efficiency-enhancement techniques should be made.

A multiobjective hierarchical problem solver needs two kinds of niching: niching to maintain diverse partial solutions and niching to maintain a good spread of solutions on the Pareto optimal front. This thesis tried to satisfy both kinds of niching needs by using RTR in combination with the crowding distance based selection scheme. However, other ways of achieving the same target should be employed and tested to improve the performance of mhBOA.

Developing Models: The work should be furthered by developing population-sizing models for mBOA and mhBOA to provide guidance in setting the population sizes for the algorithms. A scalability analysis of mBOA and mhBOA should be performed to verify the applicability of the algorithms on larger and more difficult problems. Also, convergence-time models needs to be developed for mBOA and mhBOA.

Increasing Applicability: This thesis applied mBOA and mhBOA to problems where individuals in the GA population were represented by binary strings of fixed length. mBOA and mhBOA can also be used for problems outside the domain of fixed length binary string representation such as variable-length strings or program codes. These procedure should be developed and tested. In addition, mBOA and mhBOA should be tried on real-world problems and their performance should be compared to other available techniques to further confirm its applicability and usefulness in solving real-world problems.

Chapter 8

Summary and Conclusions

This chapter summarizes the thesis and draws conclusions from the work performed.

8.1 Summary

Multiobjective genetic algorithms are widely used for solving real-world problems. Competent GAs are able to solve hard problems quickly, reliably, and accurately. However, not much effort has been put into developing competent MOGAs. This thesis developed two competent MOGAs, the multiobjective Bayesian optimization algorithm (mBOA) and the multiobjective hierarchical Bayesian optimization algorithm (mhBOA), by incorporating the multiobjective selection technique of nondominated sorting genetic algorithm (NSGA-II) (Deb, Agrawal, Pratap, & Meyarivan, 2000) into two powerful competent GAs, the Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 1999) and the hierarchical Bayesian optimization algorithm (hBOA) (Pelikan & Goldberg, 2000), respectively.

The primary contribution of the work can be summarized as follows:

Test Functions: In chapter 4, an adversarial approach was taken to design multiobjective functions for testing the performance of mBOA and mhBOA, developed later in the thesis. Some of these multiobjective test functions are deceptive with loosely linked building blocks; BB identification and proper mixing are critical for solving them. The rest of the multiobjective test problems are hierarchical functions; linkage learning,

chunking, and niching capabilities are required to solve them.

mBOA: In chapter 5, the thesis combined the multiobjective selection scheme of NSGA-II (Deb, Agrawal, Pratap, & Meyarivan, 2000) with the BB identification and mixing capabilities of BOA (Pelikan, Goldberg, & Cantú-Paz, 1999) to form a multiobjective competent GA called mBOA. mBOA was tried on test functions and results showed that it outperformed a simple MOGA on the multiobjective deceptive functions with loosely linked building blocks, but it could not solve multiobjective hierarchical problems.

mhBOA: In chapter 6, the thesis developed a multiobjective hierarchical problem solver by incorporating the selection scheme of NSGA-II (Deb, Agrawal, Pratap, & Meyarivan, 2000) into hBOA (Pelikan & Goldberg, 2000). The resulting algorithm was called mhBOA and proved to be useful in solving both multiobjective hierarchical problems and multiobjective deceptive problems with loosely linked building blocks. The study showed that mhBOA is able to consistently maintain diverse Pareto optimal solutions over the run, whereas mBOA showed gradual loss of diversity in the Pareto optimal solutions.

While this thesis presented a proof-of-principle study of competent MOGAs, there are many avenues of future research and some of them have been described in chapter 7.

8.2 Conclusions

The primary conclusion from this thesis is that GAs can be designed and used to obtain efficient solutions to hard multiobjective problems.

This thesis tested the developed algorithms on a limited number of test cases, but the adversarial design of the test functions implies that the result is of more general applicability. The test functions challenged the working assumptions of GAs; and competent

MOGAs developed in this thesis overcame the challenge. Therefore, we expect the developed algorithms to work in real-world problems without any problems specific modifications. Specifically, mBOA and mhBOA should be tried on real-world multiobjective problems and compared to simple GA derivatives.

This thesis suggests that mBOA and mhBOA will be less sensitive to function type, problem coding, dimensionality, and problem complexity of the function. These tests remain to be done, but the results of this thesis are promising in this regard.

For researchers, the thesis recommends to carry binary-coded MOGAs to real-coded, permutations, trees, networks, and program codes using BOA and hBOA as models. Theoretical studies of population sizing, run duration, and algorithm settings of competent MOGAs like mBOA and mhBOA need to be done to match the level of analogous models in single objective studies. Also, other styles of competent GAs should be mapped to multiobjective GAs. Research efforts have been made in developing multiobjective version of competent GAs and should be continued for other single objective competent GAs (Van Veldhuizen & Lamont, 2000a; Zydallis, Van Veldhuizen, & Lamont, 2001).

Transition from first generation GAs to competent GAs for multiobjective problems is exciting because it promises a day where GAs can be used without constant manipulations of coding, operators, and algorithm parameters. It is hoped that this thesis has played a role in realizing this important dream.

Appendix A

Population sizing for NSGA-II

In GA theory and practice, it is well understood that an adequate population size is essential for obtaining quality solutions. However, too large a population size delays GA convergence and leads to a higher computational time requirement (Goldberg, Deb, & Clark, 1992; Harik, Cantú-Paz, Goldberg, & Miller, 1997; Goldberg, Sastry, & Latoza, 2001). Hence, it is important for a practitioner to know the value of population size required for specified quality of solution. Therefore, a number of research efforts have been devoted to study population size requirement of a GA. However, considerably less effort has been made to study the population sizing for multiobjective GAs even though multiobjective GAs are becoming widely used for solving real-world problems. This chapter takes a preliminary step in analyzing the population size requirement of NSGA-II (Deb, Agrawal, Pratap, & Meyarivan, 2000), a multiobjective problem solver.

The chapter first discusses the two most commonly used population-sizing models: population sizing based on the initial supply of building blocks (BBs) and population sizing based on good decision making between competing BBs. Next, population-sizing model for multimodal GAs using fitness sharing is described briefly. Lastly, population sizing for NSGA-II is discussed.

A.1 Theory of Supply and Theory of Decision Making

An adequate initial supply of good BBs in the initial population is important to enhance the chances of getting to the global optimum. Holland (1975) used Poisson distribution to estimate the number of BBs that receive at least a specified number of trials in a randomly generated population, and Goldberg (1989) used binomial distribution to get a more accurate estimate of the same quantity. Recently, Goldberg, Sastry, and Latoza (2001) estimated the population size that ensures the presence of all building blocks of a partition as

$$n = \chi^k(k \log \chi - \log \alpha),$$

where, α is the supply error, k is the order of BB, and χ is the alphabet cardinality. Let m be the number of BBs, then α can be approximated as $\frac{1}{m}$. If the problem is large but easy ($m \gg \chi^k$), $n \approx O(\chi^k \log m)$. If the problem is complex but short ($m \ll \chi^k$), $n \approx O(k\chi^k \log \chi)$. This model predicts the population size required for adequate supply with reasonable accuracy.

Besides ensuring enough initial supply of BBs, it is also important to ensure that good statistical decisions are made between competing BBs in the presence of noise from the fitness contributions of the remaining BBs (Holland, 1973; De Jong, 1975). The statistical model of Goldberg, Deb, and Clark (1992) provides a conservative estimate of population size required to make correct decision between BBs as

$$n = 2c(\alpha)\beta^2(1 + \rho_T^2)m'\chi^k,$$

where, $c(\alpha)$ is the square of the ordinate of a unit-normal distribution where probability equals α, α is the probability of failure, ρ_T^2 is the total additional relative noise coefficient, $m' = m - 1$, and β^2 is the squared root-mean-squared (RMS) subfunction inverse signal-to-noise ratio. This model assumes that the correct decision between competing BBs must be

made in the first generation; hence, it proved to be very conservative. Harik, Cantú-Paz, Goldberg, and Miller (1997) accounted for the cumulative effects using the gambler's ruin model, and calculated population size as follows:

$$n = -2^{(k-1)} \ln(\alpha) \frac{\sigma_{BB}\sqrt{\pi m'}}{d}.$$

The above model incorporates the initial supply model by assuming that the gambler's initial asset is equal to the number of BBs present in a randomly generated population. The decision-making model is incorporated by taking into account the probability of making the right decision. This is less conservative than the prior model as can be seen by comparing the two population-sizing equations and has been shown to match the experimental results more closely in simple GAs.

A.2 Population sizing for Fitness Sharing

Mahfoud (1995) estimated the population size needed for a single objective GA using fitness sharing to solve multimodal functions. He identified equivalence classes in the GA population as the global and local optima of the multimodal function. Assuming the presence of all equivalence classes in the initial generation, he modeled the minimum population size required to maintain all the classes over the run as

$$n \leq \frac{2 \ln \frac{1-\gamma^{\frac{1}{g}}}{c}}{\ln[(1-p_c)(1-\frac{1}{c})^2 + p_c(1-\frac{1}{c^2}) - 2p_c(1-p_d)\frac{1}{c}(1-\frac{1}{c})]} \leq n',$$

where, g is the number of generations, p_c is the probability of crossover, p_d is the probability of disruption, and c is the number of classes. Also, n and n' are bounds on the population size.

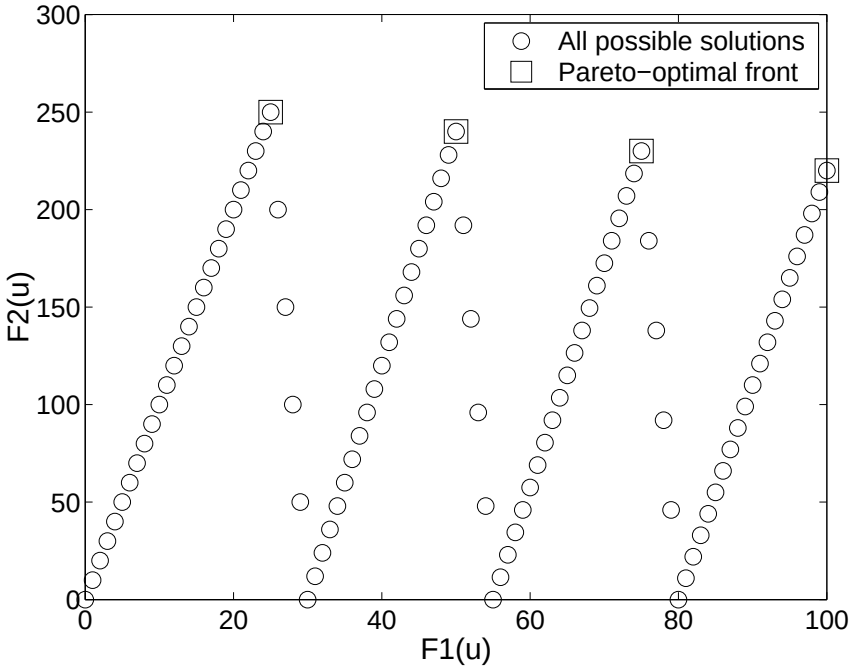


Figure A.1: Figure showing the multiobjective test function with the number of classes = 4 for $\ell = 100$.

A.3 Population sizing for NSGA-II

Reed (2002) proposed the use of population-sizing model given by Mahfoud (1995) for fitness sharing to multiobjective GAs by treating each point of the Pareto optimal front as one of the optima of the multimodal function. This section tests Reed's proposal, and applies initial supply and decision making population-sizing models to study the population size requirement of NSGA-II.

A.3.1 Experimental Design

Multiobjective test functions that resemble multimodal functions are designed to test Reed's proposal. The test functions are listed in table A.1 and table A.2.

Figure A.1 shows the multiobjective test function with the number of classes equal to 4. Both the functions (F1 and F2) of the multiobjective problem are functions of unitation. In case of multimodal functions, a class is defined by the local attractor to which a solution

Problem 1	**No. of classes = 2**		
	$F1(u) =$	u	
	$F2(u) =$	$10u$	$0 \leq u \leq 50$
		$(60-u)50$	$50 \leq u \leq 60$
		$(u-60)(490/40)$	$60 \leq u \leq 100$
Problem 2	**No. of classes = 3**		
	$F1(u) =$	u	
	$F2(u) =$	$10u$	$0 \leq u \leq 33$
		$(40-u)(330/7)$	$33 \leq u \leq 40$
		$(u-40)(320/26)$	$40 \leq u \leq 66$
		$(73-u)(320/7)$	$66 \leq u \leq 73$
		$(u-73)(310/27)$	$73 \leq u \leq 100$
Problem 3	**No. of classes = 4**		
	$F1(u)=$	u	
	$F2(u) =$	$10u$	$0 \leq u \leq 25$
		$(30-u)(250/5)$	$25 \leq u \leq 30$
		$(u-30)(240/20)$	$30 \leq u \leq 50$
		$(55-u)(240/5)$	$50 \leq u \leq 55$
		$(u-55)(230/20)$	$55 \leq u \leq 75$
		$(80-u)(230/5)$	$75 \leq u \leq 80$
		$(u-80)(220/20)$	$80 \leq u \leq 100$
Problem 4	**No. of classes = 5**		
	$F1(u)=$	u	
	$F2(u) =$	$10u$	$0 \leq u \leq 20$
		$(25-u)(200/5)$	$20 \leq u \leq 25$
		$(u-25)(190/15)$	$25 \leq u \leq 40$
		$(45-u)(190/5)$	$40 \leq u \leq 45$
		$(u-45)(180/15)$	$45 \leq u \leq 60$
		$(65-u)(180/5)$	$60 \leq u \leq 65$
		$(u-65)(170/15)$	$65 \leq u \leq 80$
		$(85-u)(170/5)$	$80 \leq u \leq 85$
		$(u-85)(160/15)$	$85 \leq u \leq 100$

Table A.1: Test functions used to analyze population size requirement of NSGA-II. Here, $\ell = 100$.

Problem 5	**No. of classes = 6**		
	$F1(u)=$	u	
	$F2(u) =$	$10u$	$0 \leq u \leq 16$
		$(20-u)(160/4)$	$16 \leq u \leq 20$
		$(u-20)(150/12)$	$20 \leq u \leq 32$
		$(36-u)(150/4)$	$32 \leq u \leq 36$
		$(u-36)(140/12)$	$36 \leq u \leq 48$
		$(52-u)(140/4)$	$48 \leq u \leq 52$
		$(u-52)(130/14)$	$52 \leq u \leq 66$
		$(70-u)(130/4)$	$66 \leq u \leq 70$
		$(u-70)(120/12)$	$70 \leq u \leq 82$
		$(86-u)(120/4)$	$82 \leq u \leq 86$
		$(u-86)(110/14)$	$86 \leq u \leq 100$
Problem 6	**No. of classes = 7**		
	$F1(u)=$	u	
	$F2(u) =$	$10u$	$0 \leq u \leq 15$
		$(20-u)(150/5)$	$15 \leq u \leq 20$
		$(u-20)(140/10)$	$20 \leq u \leq 30$
		$(35-u)(140/5)$	$30 \leq u \leq 35$
		$(u-35)(130/9)$	$35 \leq u \leq 44$
		$(49-u)(130/5)$	$44 \leq u \leq 49$
		$(u-49)(120/9)$	$49 \leq u \leq 58$
		$(63-u)(120/5)$	$58 \leq u \leq 63$
		$(u-63)(110/10)$	$63 \leq u \leq 73$
		$(78-u)(110/5)$	$73 \leq u \leq 78$
		$(u-78)(100/10)$	$78 \leq u \leq 88$
		$(93-u)(100/5)$	$88 \leq u \leq 93$
		$(u-93)(90/7)$	$93 \leq u \leq 100$

Table A.2: Test functions used to analyze population size requirement of NSGA-II. Here, $\ell = 100$.

will converge under a hill climber. Each point on the Pareto optimal front is treated like one of the optima of the multimodal function; hence, in the case of multiobjective functions, we define a class by points on the Pareto optimal front. In the case of the function shown in figure A.1, there are 4 classes and any point in the solution space will belong to exactly one of these classes. For this case (figure A.1), each solution belongs to one of the 4 classes according to the following rule,

$$Solution \in \begin{cases} class1 & 0 \leq u < 25 \\ class2 & 25 \leq u < 50 \\ class3 & 50 \leq u < 75 \\ class4 & 75 \leq u < 100 \end{cases}$$

The NSGA-II was run 100 times on each test function for different population sizes with a crossover probability of 0.9 (single point crossover, see chapter 2 for a brief explanation of single point crossover) and mutation probability of 0.01 (bit-wise mutation, see chapter 2 for a brief explanation of bit-wise mutation). The probability of success—the probability of maintaining all the classes until the last generation—was calculated for each population size as

$$\gamma = \frac{NumOfSuccess}{TotalRuns},$$

where, γ is the probability of success, $NumOfSuccess$ is the number of runs in which all the classes were maintained until the last generation, and $TotalRuns$ is the total number of runs performed (in this case, it is 100).

The population-sizing model proposed by Mahfoud (1995) can be applied to the test functions for the two extreme cases: with no disruption ($p_d = 0$) and with full disruption ($p_d = 1$), and this will give two bounds on the population size. The lower ($p_d = 0$) and upper ($p_d = 1$) bounds of population size obtained from the fitness sharing model are plotted on the same graph with the empirically obtained population sizes for different probabilities of

success. All classes are treated as equally desirable to simplify the equation. However, the selection mechanism of NSGA-II favors the end points (the end points are assigned a higher crowding distance value than other members of the same rank, see chapter 2); hence, classes are in fact not equal (the end points belong to the better fit classes).

The multiobjective test problems have onemax as one of the two objective functions. Goldberg, Deb, and Clark (1992) estimate the population size ensuring good decision making between competing BBs for onemax function as, $n = c(\ell - 1)$. Harik, Cantú-Paz, Goldberg, and Miller (1997) estimate the population size for the onemax function as, $n = -\ln(\alpha)\sqrt{0.25\pi(\ell - 1)}$. Also, Goldberg, Sastry, and Latoza (2001) give population size requirement for an adequate initial supply of BBs for the onemax function as $n = 2(\log 2 - log(\alpha))$. Empirically obtained population sizes for different probabilities of success are plotted on the same graph in figure A.2 with all the above population-sizing models.

A.3.2 Results

The figure A.2 shows that for higher probability of success the two bounds of population-sizing model of Mahfoud (1995) ($p_d = 0$ and $p_d = 1$) bound the empirically obtained population sizes. The actual p_d lies between 0 and 1; hence, estimating true p_d value can provide us with an estimate of population size requirement of NSGA-II for high success rate. Results of figure A.2 indicate that lower bound of Mahfoud (1995)'s population-sizing model ($p_d = 0$) is followed by empirical data more closely than the higher bound ($p_d = 1$); hence, the actual p_d should not be very high. At low probability of success, the population-sizing requirement is unclear and more experimentation needs to be done to analyze the population size requirement of NSGA-II at low success rate. Figure A.3 shows the increase in the population size with the increase in number of classes for the same problem size (ℓ= 100).

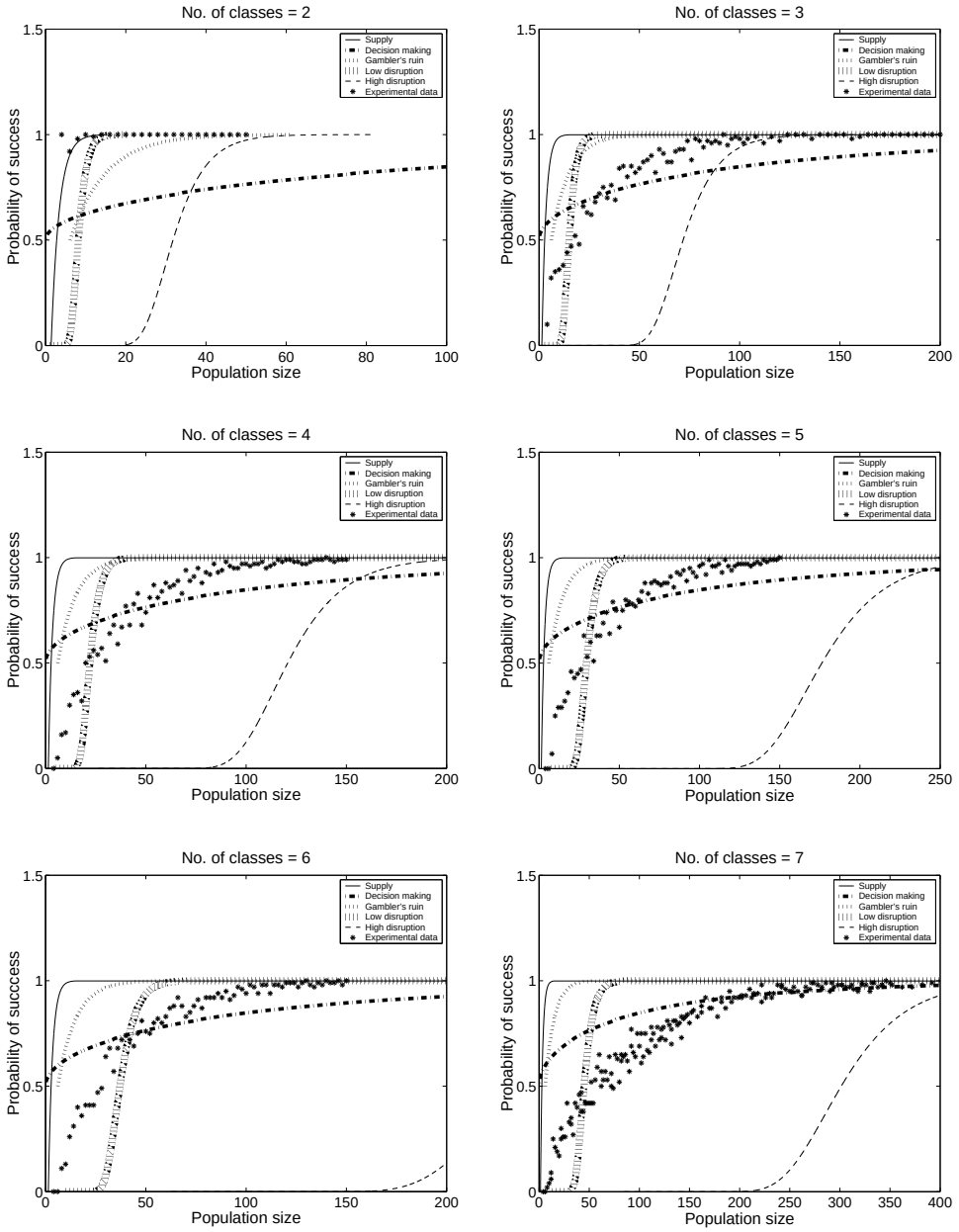


Figure A.2: Figure illustrating the empirically obtained probability of success for different population sizes along with the population-sizing model for fitness sharing, supply model, decision making and gambler's ruin model. 'Supply' represents the Goldberg, Sastry, and Latoza (2001) population-sizing model for initial supply, 'Decision making' represents the Goldberg, Deb, and Clark (1992) population-sizing model for good decision making, 'Gambler's ruin' represents Harik, Cantú-Paz, Goldberg, and Miller (1997) population-sizing model, 'Low Disruption' represents Mahfoud (1995) population-sizing model with $p_d = 0$, 'High disruption' represents Mahfoud (1995) model with $p_d = 1$ and 'Experimental data' represents the values obtained experimentally. The data used to plot the graph is from the test functions shown in table A.1 and table A.2 for $\ell = 100$, $p_c = 0.9$, $p_m = 0.01$, $g = 100$, and $TotalRuns = 100$.

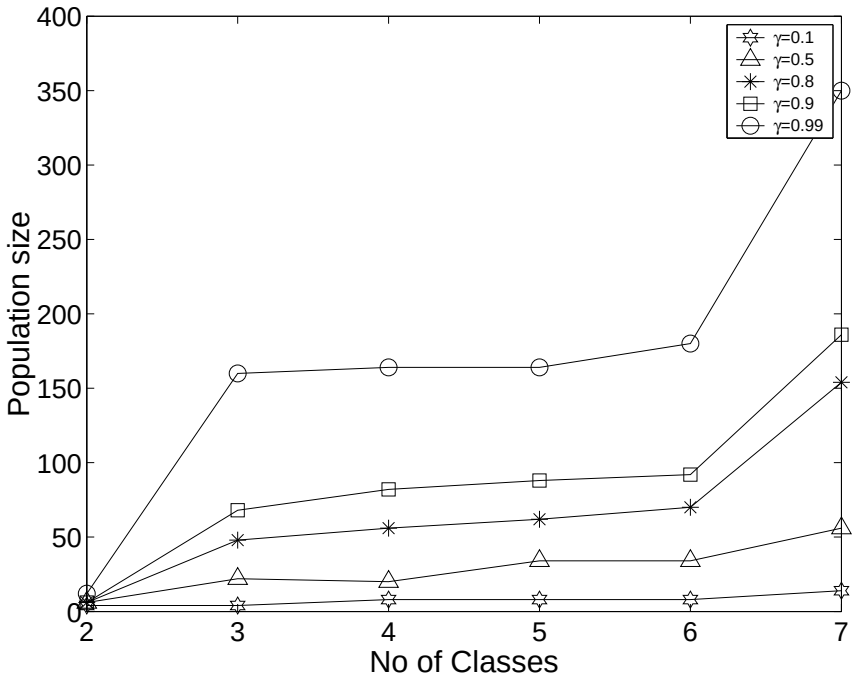


Figure A.3: Figure illustrating the empirically obtained population size for different number of classes to achieve various probabilities of success. γ represents the probability of maintaining all the classes until the last generation. The data used to plot the graph is from the test functions shown in table A.1 and table A.2 for $\ell = 100$, $p_c = 0.9$, $p_m = 0.01$, $g = 100$, and $TotalRuns = 100$.

A.4 Future Work

The study takes some preliminary steps in building a population-sizing model for NSGA-II. There are many avenues for continued research in studying the population size requirement of NSGA-II.

Firstly, the test functions used in the chapter have exponentially large number of solutions belonging to the Pareto optimal front. Hence, to test the proposal, better test functions, having a tractable number of solutions belonging to the Pareto optimal front, should be designed and tested.

Also, this study used discrete domain multiobjective test functions that have finite number of points on the Pareto optimal front. A continuous domain multiobjective function will have infinitely large number of points on the Pareto optimal front. An attempt to study population size requirement of NSGA-II for continuous domain problems should also be made.

A.5 Summary

This appendix verified Reed (2002) proposal of applying Mahfoud (1995)'s population-sizing model for fitness sharing to NSGA-II. Multiobjective test functions that resembled multimodal functions have been designed to test the proposal. Also, population size requirement of NSGA-II was studied from the perspective of ensuring a sufficient initial supply of BBs and a good decision making between competing BBs.

More research needs to be done to study the population sizing of MOGAs, but this appendix showed that at high probability of success, the population-sizing model for fitness sharing (Mahfoud, 1995) can be used to get lower and upper bounds of population size required for MOGAs. However, for low probability of success, the population-sizing model for fitness sharing is not applicable to MOGAs. By estimating the parameter p_d (the probability of disruption), we can get an estimate of population size required for a MOGA.

Bibliography

Bäck, T. (1994). Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. *Proceedings of the First IEEE Conference on Evolutionary Computation*, 57–62.

Bäck, T., Fogel, D. B., & Michalewicz, Z. (Eds.) (1997). *Handbook of evolutionary computation.* Bristol and New York: Institute of Physics Publishing and Oxford University Press.

Baker, J. E. (1989). *An analysis of the effects of selection in genetic algorithms.* Doctoral dissertation, Vanderbilt University, Nashville, TN.

Baluja, S. (1994). *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning* (Tech. Rep. No. CMU-CS-94-163). Pittsburgh, PA: Carnegie Mellon University.

Baluja, S., & Davies, S. (1997). Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. *Proceedings 14th International Conference on Machine Learning*, 30–38.

Binh, T. T., & Korn, U. (1997). Multiobjective evolution strategy for constrained optimization problems. *Proceeding of the 15th IMACS World Congress on Scientific Computation*, *1*, 357–362.

Bosman, P. A. N., & Thierens, D. (2000). *Mixed IDEAs* (Utrecht University Technical Report UU-CS-2000-45). Utrecht, Netherlands: Utrecht University.

Chickering, D. M., Heckerman, D., & Meek, C. (1997). *A Bayesian approach to learning*

Bayesian networks with local structure (Technical Report MSR-TR-97-07). Redmond, WA: Microsoft Research.

Coello Coello, C. A., & Toscano Pulido, G. (2001a). A micro-genetic algorithm for multiobjective optimization. *First International Conference on Evolutionary Multi-Criterion Optimization*, 126–140.

Coello Coello, C. A., & Toscano Pulido, G. (2001b). Multiobjective optimiztion using a micro-genetic algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference 2001*, 274–282.

Coello Coello, C. A., Van Veldhuizen, D. A., & Lamont, G. B. (2002). *Evolutionary algorithms for solving multi-objective problems*. Kluwer Academic Publishers.

Cooper, G. F., & Herskovits, E. H. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, *9*, 309–347.

Corne, D. W., Jerram, N. R., Knowles, J. D., & Oates, M. J. (2001). PESA-II: Region-based selection in evolutionary multiobjective optimization. *Proceedings of the Genetic and Evolutionary Computation Conference 2001*, 283–290.

Corne, D. W., Knowles, J. D., & Oates, M. J. (2000). The Pareto envelope-based selection algorithm for multiobjective optimization. *Proceedings of the Parallel Problem Solving from Nature VI Conference*, 839–848.

De Bonet, J. S., Isbell, C. L., & Viola, P. (1997). MIMIC: Finding optima by estimating probability densities. *Advances in Neural Information Processing Systems*, *9*, 424–431.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. Doctoral dissertation, University of Michigan, Ann Arbor. (University Microfilms No. 76-9381).

Deb, K. (1999a). Construction of test problems for multi-objective optimization. *Proceedings of the Genetic and Evolutionary Computation Conference 99*, 164–171.

Deb, K. (1999b). Evolutionary algorithms for multi-criterion optimization in engineering design. *Proceedings of Evolutionary Algorithms in Engineering and Computer Science (EUROGEN-99)*, 135–161.

Deb, K. (1999c). Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation*, *7*(3), 205–230.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms.* Chichester, UK: John Wiley and Sons.

Deb, K., Agrawal, S., Pratap, A., & Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *Proceedings of the Parallel Problem Solving from Nature VI Conference*, 849–858.

Deb, K., & Goldberg, D. E. (1993). Analyzing deception in trap functions. *Foundations of Genetic Algorithms*, *2*, 93–108. (Also IlliGAL Report No. 91009).

Deb, K., Horn, J., & Goldberg, D. E. (1993). Multimodal deceptive functions. *Complex Systems*, *7*(2), 131–153. (Also IlliGAL Report No. 92003).

Deb, K., Pratap, A., & Meyarivan, T. (2001). Constrained test problems for multi-objective evolutionary optimization. *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, 284–298.

Erickson, M., Mayer, A., & Horn, J. (2001). The niched Pareto genetic algorithm 2 applied to the design of groundwater remediation systems. *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, 681–695.

Etxeberria, R., & Larrañaga, P. (1999). Global optimization using Bayesian networks. *Second Symposium on Artificial Intelligence (CIMAF-99)*, 332–339.

Fonseca, C. M., & Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion, and generalization. *Proceedings of the Fifth International Conference on Genetic Algorithms*, 416–423.

Fonseca, C. M., & Fleming, P. J. (1995). Multiobjective genetic algorithms made easy: Selection, sharing, and mating restriction. *Proceedings of the First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, 42–52.

Friedman, N., & Goldszmidt, M. (1999). Learning Bayesian networks with local structure. In Jordan, M. I. (Ed.), *Graphical models* (1st ed.). (pp. 421–459). Cambridge, MA: MIT Press.

Goldberg, D. E. (1987). Simple genetic algorithms and the minimal, deceptive problem. In Davis, L. (Ed.), *Genetic Algorithms and Simulated Annealing* (Chapter 6, pp. 74–88).

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley.

Goldberg, D. E. (1991). *Six steps to GA happiness.* Paper presented at Oregon Graduate Institute, Beaverton, OR.

Goldberg, D. E. (1993). The Wright Brothers, genetic algorithms, and the design of complex systems. In Schaffer, J. D. (Ed.), *Proceedings of the Symposium on Neural-Networks: Alliances and Perspectives in SEnri 1993* (pp. 1–7). Osaka, Japan: Senri Internation Information Institute.

Goldberg, D. E. (1998). *Four keys to understanding building-block difficulty.* Project FRACTALES Seminar, INRIA Rocquencourt.

Goldberg, D. E. (1999). The race, the hurdle, and the sweet spot: Lessons from genetic algorithms for the automation of design innovation and creativity. In Bentley, P. J. (Ed.), *Evolutionary Design by Computers* (pp. 105–118). San Francisco, CA: Morgan Kaufmann.

Goldberg, D. E. (2002). *The design of innovation: Lessons from and for competent genetic algorithms.* Kluwer Acadamic Publishers.

Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, *1*, 69–93. (Also TCGA Report 90007).

Goldberg, D. E., Deb, K., & Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, *6*(4), 333–362. (Also IlliGAL Report No. 91010).

Goldberg, D. E., Deb, K., & Horn, J. (1992). Massive multimodality, deception, and genetic algorithms. *Parallel Problem Solving from Nature*, *2*, 37–46. (Also IlliGAL Report No. 92005).

Goldberg, D. E., Deb, K., Kargupta, H., & Harik, G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms*, 56–64. (Also IlliGAL Report No. 93004).

Goldberg, D. E., Deb, K., & Thierens, D. (1993). Toward a better understanding of mixing in genetic algorithms. *Journal of the Society of Instrument and Control Engineers*, *32*(1), 10–16. (Also IlliGAL Report No. 92009).

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, *3*(5), 493–530. (Also TCGA Report No. 89003).

Goldberg, D. E., & Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. *Proceedings of the Second International Conference on Genetic Algorithms*, 41–49.

Goldberg, D. E., & Sastry, K. (2001). A practical schema theorem for genetic algorithm design and tuning. *Proceedings of the Genetic and Evolutionary Computation Conference 2001*, 326–335. (Also IlliGAL Report Number 2001017).

Goldberg, D. E., Sastry, K., & Latoza, T. (2001). On the supply of building blocks.

Proceedings of the Genetic and Evolutionary Computation Conference 2001, 336–342. (Also IlliGAL Report No. 2001015).

Harik, G. (1999). *Linkage learning via probabilistic modeling in the ECGA* (IlliGAL Report No. 99010). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Harik, G. R. (1995). Finding multimodal solutions using restricted tournament selection. *Proceedings of the Sixth International Conference on Genetic Algorithms*, 24–31.

Harik, G. R., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Proceedings of the International Conference on Evolutionary Computation 1997 (ICEC '97)*, 7–12. (Also IlliGAL Report 96004).

Harik, G. R., & Goldberg, D. E. (1996). Learning linkage. *Foundations of Genetic Algorithms 4*, 247–262.

Harik, G. R., Lobo, F. G., & Goldberg, D. E. (1998). The compact genetic algorithm. *Proceedings of the International Conference on Evolutionary Computation 1998 (ICEC '98)*, 523–528.

Heckerman, D., Geiger, D., & Chickering, D. M. (1994). *Learning Bayesian networks: The combination of knowledge and statistical data* (Technical Report MSR-TR-94-09). Redmond, WA: Microsoft Research.

Holland, J. H. (1973). Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, *2*(2), 88–105.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press.

Horn, J., & Nafpliotis, N. (1993). *Multiobjective optimization using the niched Pareto genetic algorithm* (IlliGAL Report No. 93005). Urbana, IL: University of Illinois at

Urbana-Champaign.

Horn, J., Nafpliotis, N., & Goldberg, D. E. (1994). A niched Pareto genetic algorithm for multiobjective optimization. *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, 82–87.

Howard, R. A., & Matheson, J. E. (1981). Influence diagrams. In Howard, R. A., & Matheson, J. E. (Eds.), *Readings on the Principles and Applications of Decision Analysis*, Volume II (pp. 721–762). Menlo Park, CA: Strategic Decisions Group.

Kargupta, H. (1995). *SEARCH, polynomial complexity, and the fast messy genetic algorithm.* Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL. (Also IlliGAL Report Number 95008).

Kargupta, H. (1996). The gene expression messy genetic algorithm. *Proceedings of the International Conference on Evolutionary Computation*, 814–819.

Knowles, J., & Corne, D. (2000a). M-PAES: A Memetic algorithm for multiobjective optimization. *2000 Congress on Evolutionary Computation*, *1*, 325–332.

Knowles, J. D., & Corne, D. W. (2000b). Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation*, *8*(2), 149–172.

Kursawe, F. (1990). A variant of evolution strategies for vector optimization. *Parallel Problem Solving from Nature. 1st Workshop, PPSN I*, 193–197.

Larrañaga, P., Etxeberria, R., Lozano, J. A., & Peña, J. M. (2000). Optimization in continuous domains by learning and simulation of Gaussian networks. *Proceedings of the Genetic and Evolutionary Computation Conference 2000 Workshop Program*, 201–204.

Mahfoud, S. W. (1995). *Niching methods for genetic algorithms.* Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL. (Also IlliGAL Report No. 95001).

Miller, B. L., & Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, *9*(3), 193–212.

Mühlenbein, H., Mahnig, T., & Rodriguez, A. O. (1999). Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, *5*, 215–247.

Mühlenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. *Parallel Problem Solving from Nature - PPSN IV*, 178–187.

Mühlenbein, H., & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. *Evolutionary Computation*, *1*(1), 25–49.

Munetomo, M., & Goldberg, D. E. (1999). Linkage identification by non-monotonicity detection for overlapping functions. *Evolutionary Computation*, *7*(4), 377–398. (Also IlliGAL Report Number 99005).

Obayashi, S., Nakahashi, K., Oyama, A., & Yoshino, N. (1998). Design optimization of supersonic wings using evolutionary algorithms. *Proceedings of the Fourth ECCOMAS Computational Fluid Dynamics Conference*, 575–579.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann.

Pelikan, M. (2002). *Bayesian optimization algorithm: From single level to hierarchy*. Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL.

Pelikan, M., & Goldberg, D. E. (2000). Hierarchical problem solving and the Bayesian optimization algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference 2000*, 267–274. (Also IlliGAL Report No. 2000002).

Pelikan, M., & Goldberg, D. E. (2001). Escaping hierarchical traps with competent genetic algorithms. *Proceedings of the Genetic and Evolutionary Computation Conference*

2001, 511–518. (Also IlliGAL Report No. 2001003).

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference 99*, *I*, 525–532. (Also IlliGAL Report No. 99003).

Pelikan, M., Goldberg, D. E., & Lobo, F. (1999). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, *21*(1), 5–20.

Pelikan, M., Goldberg, D. E., & Tsutsui, S. (2001). *Combining the strengths of the Bayesian optimization algorithm and adaptive evolution strategies* (IlliGAL Report No. 2001023). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., & Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. *Advances in Soft Computing - Engineering Design and Manufacturing*, 521–535.

Quagliarella, D., & Vicini, A. (1997). Coupling genetic algorithms and gradient based optimization techniques. In Quagliarella, D., Périaux, J., Poloni, C., & Winter, G. (Eds.), *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science. Recent Advances and Industrial Applications.* (Chapter 14, pp. 289–310). West Sussex, England: John Wiley and Sons Ltd.

Reed, P. (2002). *Striking the balance: Long-term groundwater monitoring design for multiple conflicting objectives.* Doctoral dissertation, University of Illinois, Urbana-Champaign.

Rissanen, J. J. (1978). Modelling by shortest data description. *Automatica*, *14*, 465–471.

Rissanen, J. J. (1989). *Stochastic complexity in statistical inquiry.* Singapore: World Scientific Publishing Co.

Rissanen, J. J. (1996). Fisher information and stochastic complexity. *IEEE Transactions*

on Information Theory, *42*(1), 40–47.

Rothlauf, F. (2002). *Representations for genetic and evolutionary algorithms.* Physica-Verlag.

Rowe, J., Vinsen, K., & Marvin, N. (1996). Parallel GAs for multiobjective functions. *Proceedings of the Second Nordic Workshop on Genetic Algorithms and their Applications*, 61–70.

Sastry, K., & Goldberg, D. E. (2002). *Analysis of mixing in geneic algorithms: A survey* (IlliGAL Report No. 2002012). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Schaffer, J. D. (1984). *Some experiments in machine learning using vector evaluated genetic algorithms.* Doctoral dissertation, Vanderbilt University, Nashville, Tennessee. (University Microfilms No. 85-22492).

Schaffer, J. D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. *Proceedings of the First International Conference on Genetic Algorithms*, 93–100.

Sebag, M., & Ducoulombier, A. (1998). Extending population-based incremental learning to continuous search spaces. *Parallel Problem Solving from Nature - PPSN V*, 418–427.

Simon, H. A. (1969). *The sciences of the artificial.* Cambridge, MA: The MIT Press.

Soto, M. R., Ochoa, A., Acid, S., & de Campos, L. M. (1999). Introducing the polytree approximation of distribution algorithm. *Proceedings of the Second Symposium on Artificial Intelligence. Adaptive Systems.CIMAF 99*, 360–367.

Srinivas, N., & Deb, K. (1994). Multiobjective function optimization using nondominated sorting genetic algorithms. *Evolutionary Computation*, *2*(3), 221–248.

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms*, 38–45.

Van Veldhuizen, D. A., & Lamont, G. B. (2000a). Multiobjective optimization with messy genetic algorithms. *Proceedings of the 2000 ACM Symposium on Applied Computing*, 470–476.

Van Veldhuizen, D. A., & Lamont, G. B. (2000b). On measuring multiobjective evolutionary algorithm performance. *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, 204–211.

Watson, R. A., Hornby, G. S., & Pollack, J. B. (1998). Modeling building-block interdependency. *Parallel Problem Solving from Nature, PPSN V*, 97–106.

Zitzler, E., Deb, K., & Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, *8*(2), 173–195.

Zitzler, E., Laumanns, M., & Thiele, L. (2001). *SPEA2: Improving the strength Pareto evolutionary algorithm* (Technical Report 103). Zurich, Switzerland: Computer Engineering and Networks Laboratory(TIK), Swiss Federal Institute of Technology(ETH).

Zitzler, E., & Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, *3*(4), 257–271.

Zydallis, J. B., Van Veldhuizen, D. A., & Lamont, G. B. (2001). A statistical comparison of multiobjective evolutionary algorithms including the MOMA-II. *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, 226–240.