SEARCH, Polynomial Complexity, And The Fast Messy Genetic Algorithm

Hillol Kargupta
Department of Computer Science
University of Illinois At Urbana-Champaign

IlliGAL Report No. 95008
October 1995

Illinois Genetic Algorithms Laboratory (IlliGAL)
Department of General Engineering
University of Illinois at Urbana-Champaign
117 Transportation Building
104 South Mathews Avenue
Urbana, IL 61801

SEARCH, POLYNOMIAL COMPLEXITY, AND THE FAST MESSY GENETIC ALGORITHM

BY

HILLOL KARGUPTA

B.Tech., Regional Engineering College Calicut, 1988
M.Tech, Indian Institute of Technology Kanpur, 1991

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1996

Urbana, Illinois

Abstract

Blackbox optimization—optimization in presence of limited knowledge about the objective function—has recently enjoyed a large increase in interest because of the demand from the practitioners. This has triggered a race for new high performance algorithms for solving large, difficult problems. Simulated annealing, genetic algorithms, tabu search are some examples. Unfortunately, each of these algorithms is creating a separate field in itself and their use in practice is often guided by personal discretion rather than scientific reasons. The primary reason behind this confusing situation is the lack of any comprehensive understanding about blackbox search. This dissertation takes a step toward clearing some of the confusion.

The main objectives of this dissertation are:

1. present SEARCH (Search Envisioned As Relation & Class Hierarchizing)—an alternate perspective of blackbox optimization and its quantitative analysis that lays the foundation essential for transcending the limits of random enumerative search;

2. design and testing of the fast messy genetic algorithm.

SEARCH is a general framework for understanding blackbox optimization in terms of relations, classes and ordering. The primary motivation comes from the observation that sampling in blackbox optimization is essentially an inductive process (Michalski, 1983) and in absence of any relation among the members of the search space, induction is no better than enumeration. The foundation of SEARCH is laid on a decomposition of BBO into relation, class, and sample spaces. An ordinal, probabilistic, and approximate framework is developed on this foundation to identify the fundamental principles in blackbox optimization, essential for transcending the limits of random enumerative search. Bounds on success probability and sample complexity are derived. I explicitly consider specific blackbox algorithms like simulated annealing, genetic algorithms and demonstrate that the fundamental computations in all of them can be captured using SEARCH. SEARCH also offers an alternate perspective of natural evolution that establishes the computational role of gene expression (DNA→RNA→Protein) in evolution. This model of evolutionary computation hypothesizes a possible mapping of the decomposition in relation, class, and sample spaces of SEARCH into the transcriptional regulatory mechanisms, proteins, and DNA respectively.

The second part of this dissertation starts by noting the limitations of simple GAs, which fail to properly search for relations and makes decision making very noisy by combining relation, class, and the sample spaces. Messy genetic algorithms (Goldberg, Korb, & Deb, 1989; Deb, 1991) are a rare class of algorithms that emphasize the search for relations. Despite this strength of messy GAs, they lacked complete benefits of implicit parallelism (Holland, 1975). The fast messy GA initiated by Goldberg, Deb, Kargupta, and Harik (1993) introduced some of the benefits of implicit parallelism in messy GA without sacrificing its other strengths very much. This dissertation investigates fast messy GAs and presents test results to demonstrate its performance for order-k delineable problems.

For Mom, dad,

and

my mentor

Acknowledgments

One of my most rewarding experiences of finishing the enduring task of writing the Ph.D. dissertation is to finally get a chance to write this part. I look at the past, the present and take a moment to appreciate all the people who have contributed to my life. I put some of them in black and white here.

At this moment, I just wish that if I could see my mom and tell her that I did what she always wanted me to do. I wish if I could. I am where I am now because of her. I don't know what else to say.

My father is the other person who influenced me so much. He taught me how to be upright, showed me fortitude, and told me not be quitter. My parents sacrificed so much, almost every single pleasure of their life for my education. I have never heard any grievance for that.

Professor David Goldberg is a person from whom I learned so much on so many aspects of life. His contribution to my life helped me finding my own identity and realizing what I want to be. His self-less preaching influenced almost every aspect of my thought. On the research front, almost all of my works originated from his insightful notes. He made my life hard while writing this dissertation for the due reason. I know I will live to appreciate that throughout my future life.

My sister, Barnali, is a wonderful person. She sacrificed her career and stayed at home to take care of our dad, while I lived in a different hemisphere of the world to finish my degree. I could not do my Ph.D. without her.

My wife, Kakali, has been with me since I started my graduate studies in IIT Kanpur. She went through all the up and downs of my graduate life, helped me doing research, stayed up night after night for reading my dissertation, and embraced me when I needed a place to hide.

Illinois Genetic Algorithm Laboratory (IlliGAL) is a great place to work. Kalyanmoy Deb, Jeffrey Horn, Sam Mahfoud, Dirk Thierens, Georges Harik, Brad Miller, Eric Cantupaz—they are all great colleagues. We worked, fought, and argued about almost any thing in the world.

We learned a lot of stuff together. I really loved arguing with Jeff and Dirk. I enjoyed pulling Georges' leg. Jeff, and George went over part of my dissertation and gave me many useful feedback. Eric and Brad are two wonderful persons who have picked up the IlliGAL traits quickly. I thank you all for everything.

I thank Professor Sylvian Ray and Professor Robert Skeel for helping me in my research and agreeing to be in my committee. I enjoyed working with Professor Ray and he is one among the modest person I have ever seen in my life. I am very grateful to Professor Jay Mittenthal from giving me useful insights that helped me developing the biological implications of the SEARCH framework. I also thank Professor David Wilkins for being on my prelim committee and the valuable experience that I gathered while working with him.

Our ever smiling librarians—Kerry Zakrzewski, Anju Jaggi, and Cecilia Chang have always provided an excellent service to my research. They took so much pain in typing and mailing the recommendation letters for my job applications. They photocopied stuff whenever I needed any literature. They have been really wonderful.

Kevin Carmody helped me a lot by providing the much needed computational horsepower for the work of this thesis. I am extremely grateful to him. I thank Ms. Sydney Cromwell and Ms. Sheryl Hembrey for taking care of my funding paper works. This work has been supported by US Army Contract DASG60-90-C-0153, AFOSR Grant F49620-94-1-0103, NCSA supercomputing grant, and the UIUC Department of General Engineering. I also thank the Sigma-Xi chapter of UIUC for awarding me travel grants in 1992 and 1994 that helped my research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Searching for a solution to any problem usually starts with understanding the problem. However, as the degree of complexity of the problem increases, the chances for understanding the problem by pure human intellect also diminish. Many practical optimization problems fall into this category. Lack of enough domain knowledge and nonlinear interaction among the optimization variables of a large problem can easily bewilder man and machine. Optimizing the assembly schedule of an automobile industry and finding an optimal layout for placing the logic circuits in a computer chip are some examples. Blackbox optimization (BBO)—optimization in the presence of little domain knowledge—tries to capture the characteristics of such problems.

Because of its practical importance, BBO has recently drawn increased attention. A large number of algorithms have shown up in the literature, for example simulated annealing (Kirpatrick, Gelatt, & Vecchi, 1983), genetic algorithms (Holland, 1975), and tabu search (Glover, 1989). These algorithms apparently differ from one another on many aspects. Their performance and scope often vary widely from one problem to another. With all these new algorithms, designed based on seemingly different principles, it is quite natural to ask whether any common principle and common ground exist for approaching BBO.

This dissertation makes an attempt to answer this question. It introduces SEARCH (Search Envisioned As Relation, and Class Hierarchizing)—an alternate perspective toward BBO. It starts by noting that blackbox search is an induction problem, i.e., a matter of guessing based on what is known. Induction is no better than enumerative table look-up when no relation exists among the members of the search space (Mitchell, 1980; Watanabe, 1969). SEARCH

realizes this, and it tries to exploit the properties of different relations that can be defined among the members of the search space. It decomposes the search for optimal solution in BBO into three components, searching in

1. relation space
2. class space
3. sample space

This decomposition lays the foundation of the SEARCH framework. A probabilistic and approximate framework is developed on this foundation to give SEARCH all its capabilities. Both qualitative and quantitative analyses of this framework are developed here. By doing this, SEARCH makes an attempt to answer some of the following questions: Can we bound the success probability in BBO? Can we quantify the role of relations in BBO? For algorithms that use a representation,[1] can we quantify how good it is for the given problem? Can we bound the sample complexity in blackbox optimization? Both qualitative and quantitative answers are provided. The relation between SEARCH and the PAC learning framework (Natarajan, 1991; Valiant, 1984) is discussed. Popular blackbox optimization algorithms, such as simulated annealing and genetic algorithms, are also projected in the light of SEARCH. Natural evolution can also be viewed as a typical BBO problem. This makes us wonder whether SEARCH tells us anything new about the computational processes in natural evolution. The answer is yes, it does. An alternate model of evolutionary computation is recently proposed (Kargupta, 1995a) that, unlike most of the existing computational models of evolution, accounts for the role of intra-cellular flow of information—the gene expression.[2]

The SEARCH framework has many implications on the approach toward solving BBO problems. Solving a BBO problem requires an understanding of the three participants of the process:

1. algorithm
2. problem

[1] an auxiliary space defined by transforming the original search space.

[2] DNA→RNA→Protein construction process during the transcription and translation is known as gene expression.

3. user

SEARCH identifies the different components of a BBO algorithm. It also provides us with an understanding of problem difficulty in a quantitative manner. It answers the following questions: How does SEARCH characterize difficult BBO problems? Is there a class of blackbox problem that is solvable in polynomial sample complexity?[3] What is the role of user in blackbox search? Again, both qualitative and quantitative arguments are used to answer these questions.

The second half of this dissertation addresses the issue of designing BBO algorithms. I focus on a particular class of genetic algorithms (GAs), called messy genetic algorithms (mGAs), initiated in earlier studies (Deb, 1991; Goldberg, Korb, & Deb, 1989). Unlike simple GAs (De Jong, 1975; Goldberg, 1989; Holland, 1975), messy GAs emphasize the search for appropriate relations among the members of the search space. Since historically the mGAs came before the development of SEARCH, the mGAs deserve the credit for taking the right step in the right direction. Although the original version of messy GA (Goldberg, Korb, & Deb, 1989) successfully solved bounded difficult problems,[4] it adopted an enumerative initialization of relations. Although the sample complexity was polynomial, it was still very expensive in terms of practical use. The fast messy GA (fmGA) was introduced by (Goldberg, Deb, Kargupta, & Harik, 1993) and this eliminated one bottleneck of mGAs. The fmGA brought some of the benefits of implicit parallelism (Holland, 1975) to messy GAs without sacrificing the search for appropriate relations in the messy GA. Carefully designed experiments are also provided to demonstrate the performance of fmGA.

Chapter 2 introduces the SEARCH framework. After presenting a brief review of BBO, it presents both informal and formal description of SEARCH. Chapter 3 presents the main implications of the previous chapter's results. It develops an approach toward the main participants in a blackbox search—algorithm, problem, and user. Chapter 4 presents the design of the fast messy GA, after briefly reviewing the lessons from simple GA and messy GA. Chapter 5 first presents the rationale behind the design of a test suite using our understanding of problem difficulty and then presents the results of testing fmGA on different classes of bounded difficult problems. Chapter 6 describes the results of applying fmGA to track missile blip targets

[3] number of samples needed to solve a problem grows polynomialy with the size of the problem, accuracy of the solution, and the reliability.

[4] a class of problems that is solvable in polynomial sample complexity in SEARCH

in different radar frames. Finally, Chapter 8 concludes this dissertation and outlines future ramifications.

Chapter 2

SEARCH: An Alternate Perspective of Blackbox Optimization

Blackbox optimization (BBO)—a special kind of optimization in which little knowledge about the problem is assumed—has recently enjoyed a large increase in interest. Many different classes of blackbox search algorithms have been developed. Simulated annealing (Kirpatrick, Gelatt, & Vecchi, 1983) genetic algorithms (Holland, 1975), and tabu search (Glover, 1989) are some examples. Despite the increasing interest for solving blackbox problems, each algorithm is a new research field in itself. Some algorithms are quite general, while some of them are for special purposes. Some emphasize the role of representation and some do not. The nature of the search operators varies widely between algorithms. Some are content with local optimization and some aspire to global optimization. In short, there exist few common principles, little common language, little cross-fertilization, and almost no understanding of the fundamental similarities and differences between BBO algorithms in current development and usage.

With all this activity and confusion it is reasonable to ask whether it is possible to create a common perspective of BBO and discuss common design principles. The answer of this chapter is yes. To support this claim, in this chapter I propose a probabilistic framework called Search Envisioned As Relation, and Class Hierarchizing (SEARCH). SEARCH presents an alternate perspective of probabilistic adaptive sampling search in BBO. It is not intended

to be an exact model of one particular BBO algorithm; rather, its purpose is to capture the essential computational aspects of BBO algorithms and explore their effects on the bounds in efficiency.

The following sections of this chapter introduce the SEARCH framework. Section 2.1 introduces blackbox optimization and briefly reviews some related works. Section 2.2 presents a general informal introduction to SEARCH, and this is followed by a formal development in Section 2.3. Section 2.4 quantifies the success probabilities of SEARCH. Section 2.5 specializes this framework for ordinal class and relation selection processes. I compute bounds on success probability and sample complexity and explore the conditions of polynomial complexity search in blackbox optimization. Section 2.6 discusses the correspondence between the SEARCH framework and PAC-learning (Natarajan, 1991; Valiant, 1984). This is followed by a projection of simulated annealing in the light of SEARCH in Section 2.7. Section 2.8 briefly reviews the main points of an alternate model of evolutionary computation proposed using the SEARCH framework (Kargupta, 1995a). Finally, Section 2.9 summarizes the major points of this chapter.

2.1 Background

Although optimization has been addressed in both theory and practice for several centuries, the methodology for solving optimization problems have often followed a pattern: Given a very specific class of problems with some known properties, design an algorithm for solving this class. Unfortunately, because of the ever-growing list of different optimization problems, the process of designing new problem-specific algorithms is unlikely to terminate. Designing algorithms for solving blackbox problems—optimization problems with little knowledge available about the problem domain—offers an alternate approach. By assuming little knowledge about the problem, algorithms designed using this approach aspire to solve a more general class of problems.

The purpose of this section is to introduce BBO and to review some earlier works. Section 2.1.1 introduces BBO and Section 2.1.2 reviews some existing works on BBO.

2.1.1 Blackbox optimization

Almost every discipline of engineering and science can make use of optimization algorithms. As a result, a large number of optimization algorithms have been developed and applied to different problems. For example, smooth convex functions can be efficiently optimized using gradient search techniques (Papadimitriou & Steiglitz, 1982) The simplex algorithm (Dantzig, 1963) performs well for a large class of linear programming problems. Dynamic programming techniques (Dreyfus & Law, 1977) work well when the optimization problems are stage decomposable. Several analyses have been done for local and global optimization of real functions that are Lipschitz continuous with a known Lipschitz constant (Törn & Žilinskas, 1989; Vavasis, 1991). This approach of operations research is characterized by a pattern: Given a class of problem, find an algorithm to solve it. Unfortunately, this approach of designing algorithms that work only for a specific class of problems does not ever seem to end, as the list of different types of optimization problems continues to grow. Moreover, determining the class of problems in which a real-world optimization problem belongs is often as difficult as finding a reasonable solution for the problem. The ever-increasing computing capability has also fueled the desire for solving large-scale problems with little prior knowledge about the objective functions.

This growing demand for algorithms to solve new classes of difficult optimization problems and the never-ending process of designing algorithms that work for a restricted class of problems suggest the need for an alternate approach. The applicability of the previously mentioned optimization algorithms is very restricted, because these algorithms make assumptions about the properties of the objective functions that are often too restrictive. Therefore, one step toward designing optimization algorithms that work for a large class of problems is to reduce assumptions about the objective function. Since these algorithms make little assumption about the objective function, they should be able to solve problems using as little domain knowledge as possible. These problems would fall into the general class of blackbox optimization (BBO) problems, where little knowledge about the objective function is assumed. In this model of optimization, the objective function is often available as a black box, i.e., for a given x in the feasible domain, it returns the function value $\Phi(x)$. No local or global information about the function is assumed. Let us denote the finite input and the output spaces by $\mathcal{X}$ and $\mathcal{Y}$, respectively. The general blackbox optimization problem can be formally defined as follows.

Given a blackbox that somehow computes $\Phi(x)$ for an input x,

$$\Phi : \mathcal{X} \rightarrow \mathcal{Y} \tag{2.1}$$

The objective of a maximization problem is to find some $x^* \in \mathcal{X}$ such that $\Phi(x^*) \geq \Phi(x)$ for all $x \in \mathcal{X}$. Performance of an optimization algorithm in this model depends on the information collected by sampling different regions of the search space. The following section presents a brief review of some previous studies related to the work presented in this chapter.

2.1.2 Brief review of previous works

By definition, a strict blackbox search algorithm must work without any prior information about the structure of the objective function. Although the field of global optimization has a rich volume of literature, many studies are severely restricted because of their assumptions about the properties of the objective function (Schoen, 1991), and therefore it can be questioned whether they can really be called BBO algorithms. The objective of this section is to present a brief account of some previously developed algorithms that make little use of domain information about the problem. First, I present a classification of BBO algorithms based on whether the algorithm is deterministic or non-deterministic. Next, I concentrate on the non-deterministic or stochastic methods. Finally, I present a brief description of some previous efforts to relate different BBO algorithms with one another and to understand them on common grounds.

Although there may be several ways to classify optimization algorithms from different points of view (Törn & Žilinskas, 1989), one natural candidate is classification based on the deterministic or non-deterministic nature of the search algorithm. Several earlier efforts (Archetti & Schoen, 1984; Dixon & Szegö, 1978; Gomulka, 1978) suggested classification of global optimization algorithms using this approach. BBO algorithms can be similarly classified as

- Deterministic approaches
- Stochastic approaches
  - blind random search methods
  - adaptive sampling search methods

Each of these approaches will be briefly described in the following.

Deterministic enumeration of members of the search space is one method. Unfortunately, for most of the interesting optimization problems, deterministic enumeration becomes practically impossible because of the growth in the search space.

On the other hand, the stochastic algorithms introduce some random elements into the algorithm and try to solve the problem by relaxing the guarantee of the deterministic enumerative search. This relaxed nature of stochastic search algorithms makes them more suitable for practical applications.

Blind random search (Schoen, 1991; Törn & Žilinskas, 1989) is probably the simplest class of algorithms within the family of stochastic BBO algorithms. The Monte Carlo and multistart algorithms are examples of this kind of algorithm. The Monte Carlo algorithm generates random samples from the search space according to a fixed distribution. Multistart methods make use of local search techniques in addition to the Monte Carlo sample generation process. Although algorithms of this class are simple in nature, they are likely to be suitable for the worst case when different regions of the search space cannot be qualified and when evaluating a particular member of the search space does not provide information about another member.

Adaptive sampling search techniques try to exploit the information gathered from samples taken from the search space. They try to qualify different regions of the search space in terms of the fitness values of their members and use that information to decide which region to explore next. Bayesian algorithms, clustering methods, simulated annealing (SA) and genetic algorithms (GAs) are examples of this class of algorithms. This dissertation mainly considers this class of algorithms.

Bayesian algorithms (Betrò, 1983) try to develop a statistical model of the objective function. These algorithms do not explicitly construct a function; instead, they use a random variable to minimize the expected deviation of the estimate from from the real global optimum. The expected value of the random variable is set to the best estimate of the function and the variance of the random variable capture the uncertainty about this estimate. The problem of Bayesian algorithms are that they are often complicated and involve fairly cumbersome computations, such as computing the inverse of the covariance matrix (Törn & Žilinskas, 1989).

Clustering methods (Rinnooy Kan & Timmer, 1984; Törn & Žilinskas, 1989) use a Monte Carlo sample generation technique. Cluster analysis algorithms are used to identify local minima. This is followed by a local search for each local optimum. Clustering methods have been

found useful for many global optimization problems (Hart, 1994; Törn & Žilinskas, 1989). However they are likely to perform poorly when the objective function is multimodal and there are many local optima (Hart, 1994).

Since the early 80s, the simulated annealing (SA) algorithms (Kirpatrick, Gelatt, & Vecchi, 1983) and their variants have been used for solving blackbox problems. The natural motivation behind SA is the statistical behavior of molecules during the crystallization process in annealing. SA considers one sample at a time and this sample represents the state of the algorithm. A neighborhood generator is used to generate new samples. SA makes use of a probabilistic comparison statistic (the Metropolis criterion) for deciding whether the new sample should be accepted as the state of the algorithm. The Metropolis criterion dynamically changes along with a parameter known as temperature. The temperature takes a high value in the beginning and gradually decreases according to a chosen cooling schedule. The acceptance probability is often very high in the beginning, when the temperature is high. The acceptance probability decreases as the temperature reduces. SA has a proof for asymptotic convergence to the optimal solution (Kirpatrick, Gelatt, & Vecchi, 1983) SA has been applied to a wide range of blackbox problems. Many of them reported very promising results. However, in the recent past several negative results have also come out (Dueck & Scheuer, 1988; Ferreira & Žerovnik, 1993).

Genetic algorithms (GAs) (De Jong, 1975; Goldberg, 1989; Holland, 1975), evolutionary programming (Fogel, Owens, & Walsh, 1966), and evolutionary strategies (Rechenberg, 1973) are also getting increasing attentions for dealing with global optimization in blackbox problems. Design of the simple genetic algorithm (GA) is motivated by natural evolution. Unlike the SA, it emphasizes the role of representation and the interaction between the representation and perturbation operators. GAs use the representation to implicitly divide the search space into several non-overlapping classes often called schemata (Holland, 1975). Unlike SAs, GAs work from a population of samples, with each sample often represented as sequences. This population of sequences is used to evaluate different schemata. New samples are generated by crossover and mutation operators. Crossover also implicitly combines the better schemata while generating new samples. GAs have been successfully applied to different classes of problems (Goldberg, 1989). However, the simple GA suffers from several limitations. Although the simple GA realizes the role of representation that induces relations among members of the search space, the simple GA does not really search for appropriate relations. Moreover, the evaluation of

schemata is also very noisy in the simple GA. These issues will be revisited and elaborated in Chapter 4.

With all these different BBO algorithms in our arsenal, it is quite natural to ask whether they can be studied on common grounds using common principles. Several previous efforts have been made to address this question. Although Holland's (1975) framework for adaptive search was primarily motivated by evolutionary computation, the underlying concepts of search based on schema processing and decision making are fundamental issues that are equally relevant in the context of any other adaptive BBO algorithms. In fact, Holland's work (1975) is the root of the current thesis. Davis (1987) made an effort to put literature on SAs and GAs under a common title. Unfortunately, this book did not make any direct effort to link them; rather, it simply discussed them separately. Sirag and Weisser (1987) combined several genetic operators into a unified thermodynamic operator and used it to solve traveling salesperson problems. However, this paper did not study the fundamental similarities and differences between SAs and GAs. Goldberg (1990) addressed this issue. He presented a common ground to understand the effects of the different operators of SAs and GAs. He also proposed the Boltzmann tournament selection operator, which attempts to achieve Boltzmann distribution over the population. Mahfoud and Goldberg (1992) introduced a parallel genetic version of simulated annealing called *parallel recombinative simulated annealing*. This algorithm attempted to harness the strengths of both SAs and GAs. Recently Rudolph (1994) developed a Markov chain formulation of SAs and GAs for analyzing their similarities and differences. Jones and Stuckman (1992) made an interesting effort to relate GAs with Bayesian approaches to global optimization. They noted the similarities and differences between these two approaches and concluded that they share many common grounds. They also developed hybrid algorithms that try to harness the strengths of both approaches. Recently Jones (1995) proposed a framework to study the correspondence between evolutionary algorithms and heuristic state space search of graph theory. In this approach the search domain of the objective function is viewed as a directed, labeled graph. Jones and Forrest (1995) also proposed the fitness-distance-correlation measure for quantifying search difficulty and applied this measure to several classes of objective functions.

Unfortunately, very few of the previous works actually made a quantitative effort to study the computational capabilities and limitations of BBO. Little attention has been paid to the role of relations in BBO, which is essential for transcending the limits of random enumerative search.

We still lack any common framework that describes these different algorithms in terms of the basic concepts of theory of computation. The SEARCH framework, which will be introduced in the remainder of this chapter, attempts to do that. The development of SEARCH will require some familiarity with concepts of set theory and computational complexity. Readers not familiar with this field should read Appendix A or a standard textbook such as Cormen, Leiserson, and Rivest (1990).

2.2 SEARCH: An Informal Picture

SEARCH presents an alternate picture of blackbox optimization in terms of relations and classes that can be constructed among the members of the search space. SEARCH is also a formal framework that helps us quantify different aspects of BBO, such as sample complexity, problem difficulty, and many more. However, the fundamental ideas are quite simple and can be understood without introducing any formal terms. The purpose of this section is to provide such an introduction. Section 2.2.1 presents a brief motivation for this section. Section 2.2.2 considers an example optimization problem and illustrates the underlying concepts in SEARCH.

2.2.1 Motivation

Some existing BBO algorithms try to find the optimal solution by directly searching the original domain of optimization variables. Samples are often used to estimate the best solution of the search space. In these approaches, a BBO algorithm always searches for a better solution compared to the current best solution. It takes one or more samples and then decides how to choose the next sample. Although the task is certainly non-trivial, the approach of finding the best solution by iteratively updating the best estimate has a fundamental problem. Sampling one particular point from the search domain does not necessarily tell us anything about another point. When a BBO algorithm makes a decision to sample another member from the domain, it is performing *induction*—the process of hypothesizing the premise from the consequences (Michalski, 1983). This is because we are first observing the objective function values for the members of the sample set and then trying to determine whether an unknown point should have a higher or lower objective function value. In other words, it is guessing; it is a proven fact that induction is impossible when no relation exists between the members (Mitchell, 1980;

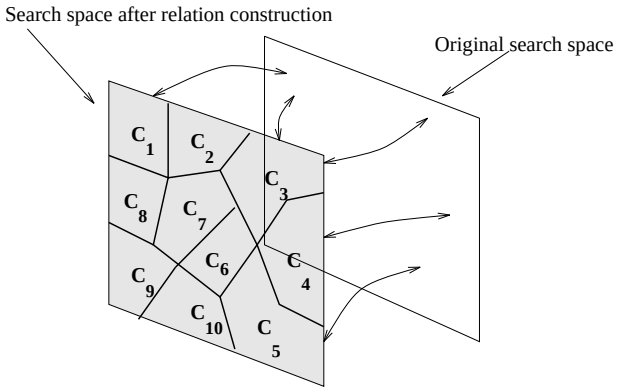


Figure 2.1: Classification of the search space using relations.

Watanabe, 1969). If no prior relation is assumed between them, there is little reason to choose one member over others, and the blackbox search will be no better than the random search unless the algorithm assumes and exploits some relations among the members of the search domain.

If assuming and exploiting relations among the members of a search space is essential, then it will be wise to isolate this possibility, study it, and see how it can be used to the fullest. The SEARCH framework does that. Recall that SEARCH stands for *Search Envisioned As Relation and Class Hierarchizing*. Searching for better relations and better classes are the primary fronts emphasized in SEARCH. Relations classify the search space into different regions. Figure 2.1 presents a schematic diagram of the search spaces divided into a set of non-overlapping classes after a relation is defined. Some relations classify the search space in such a way that it is relatively easier to detect the class containing the optimal solution. SEARCH tries to establish such relations among the members of the search space. Instead of directly searching for the best solution from the beginning, SEARCH tries to find these relations and then use them to locate the classes containing the optimal solution. Although the scope of this framework is quite general, in the following section I consider a simple optimization problem represented using a binary sequence space to explain the main concepts in simple terms.

Table 2.1: A sample set.

x	$\Phi(x)$
1000	1
1001	2
1010	0
1011	1
1110	3

2.2.2 An illustration

Consider an unconstrained optimization problem represented using four bits. Table 2.1 shows a five-member sample set; out task is to determine the optimal solution. All we know is that this sample set and the search domain defined by the binary representation are discrete. With no other information about the objective function available, this task seems to be very difficult. In fact, many established algorithms of operations research such as simplex, ellipsoid, and convex programming, will be clueless when faced with this problem. Unfortunately, we need to deal with it and this is what blackbox optimization is all about.

A more optimistic picture may be developed by carefully considering of the samples and then guessing possible relations among them. We can try to correlate the regularities among the binary representation of the members with the grades in objective function values by making inductive hypotheses. For example, strings 1001 and 1000 differ only at the rightmost bit position, and the objective function value of 1001 is greater than that of 1000 by 1. Similarly, the objective function value of 1011 is greater than that of 1010 by the same amount, 1. These two observations may lead us to hypothesize that having 1 in the rightmost bit position may lead to higher objective function values. Another possible hypothesis may be that a 0 in the second bit position from the left and 1 in the rightmost position may be the features of the good solutions in this representation, since this is a feature of both 1001 and 1011. However, the last member of the sample set 1110 has a much higher objective function value. This may help us in eliminating our second hypothesis that a 0 in the second from left position is associated with better solutions. Among all these hypotheses, one possibility is that three consecutive 1s in the first three bit positions from the left are good features, since 1110 has a higher objective function value; 1 in the rightmost position also results in increasing objective function value. These two

Table 2.2: A function in 4-bit representation.

x	$\Phi(x)$
0000	2.5
0001	3
0010	1
0100	1
1000	1
0011	2
0101	2
1001	2
0110	0
1010	0
1100	0
0111	1
1011	1
1101	1
1110	3
1111	4

hypotheses suggest that the string 1111 is the optimal solution. This turns out to be the correct solution of the problem. Table 2.2 defines the objective function. This function is constructed in such a way that in the chosen representation, it can be decomposed into two subproblems: (1) the first three bit positions from the left together and (2) the rightmost bit position. The search for the optimal solution can therefore be viewed as the search for appropriate relations or regularities among the members of the sample set. This is the fundamental concept in SEARCH.

One way to talk about these regularities is through using a notation of patterns that may be defined using equivalence relations and classes. The four-bit representation used in our example defines $2^4 = 16$ relations corresponding to the different partitions. Every partition defines a unique equivalence relation in this space of binary strings. For example, $\#\#\#f$ (where f stands for a fixed bit that matches for equivalence) is an equivalence relation defined over the last bit position. Relation $\#\#\#f$ divides the search space into two classes, $\#\#\#1$ and $\#\#\#0$.

Now that we have a precise way to talk about the regularities among the members of the sample set, we need to qualify these regularities in terms of the objective function values. In other words, we want to be able to say that a particular class is better than some other class in

Table 2.3: Members of class ###1 and ###0 and the class average statistics.

###1		###0	
x	$\Phi(x)$	x	$\Phi(x)$
1111	4.0	1110	3.0
1101	1.0	1100	0.0
1011	1.0	1010	0.0
0111	1.0	0110	0.0
1001	2.0	1000	1.0
0101	2.0	0100	1.0
0011	2.0	0010	1.0
0001	3.0	0000	2.5
Avg.	2.0	Avg.	1.0625

Table 2.4: Average objective function values of the classes in $fff\#$.

Class	Average
111#	3.5
000#	2.75
100#	1.5
001#	1.5
010#	1.5
110#	0.5
101#	0.5
011#	0.5

a certain sense. Two classes may be compared by first defining a statistic over them. Table 2.3 shows the members of the two classes ###1 and ###0. Note that every member of ###1 has a higher objective function value compared to the corresponding member of ###0. This is just one way to compare the two classes. Comparing the class average objective function value may be a different way to look at them. Table 2.3 also shows the class average statistic for these two classes. Since ###1 contains the optimal solution, the relation $\#\#\#f$ classifies the search space in such a way that correct decision making is possible with any one of the statistics just described. Similarly, the relation $fff\#$ also identifies the class 111# to be the better class using the class average statistic. Table 2.4 shows the class average statistic for all the classes defined by relation $fff\#$ and, as we see, the class 111# is ranked highest. On the other hand, some relations such as $f\#\#\#$ and $\#f\#\#$ do not correctly identify the class

Table 2.5: Class average statistics of the classes defined by relations $f###$ and $\#f##$.

f###		#f##	
Class	Avg.	Class	Avg.
0###	1.5625	#0##	1.5625
1###	1.5	#1##	1.5

containing 1111 as the best class, using the same class average statistic. Table 2.5 shows the class average statistics for classes defined by these two relations, and it shows that the average statistics for 0### and #0## are greater than those of 1### and #1## respectively. For a given statistic for class comparison, some relations may classify the search space in such a way that the class containing the optimal solution is ranked higher than other classes; therefore, correct decision making in choosing the right class is possible. These relations are appropriate for classifying the search space from the optimization perspective. This is an important concept in SEARCH and therefore, we shall give it a name. If a relation ranks the class containing the optimal solution in such a way that correct decision making is possible, then we say that this relation properly *delineates* the search space. If a relation does not do that, it does not properly delineate the search space; One may also say that a relation does not satisfy the delineation requirement.

As I noted earlier, relations do not necessarily have to come from representation. Neighborhood generation heuristics and operators can also be used to define relations. For example, consider the *k-opt* algorithm (Lin & Kernighan, 1973) used for solving the traveling salesperson problem. This algorithm makes use of a *k-change* neighborhood generation heuristic that generates a set of neighbors of a member of the search space and keep the member that is locally optimal with respect to the *k-change* neighbors. In this algorithm the relation among the search space members can be defined by this *k-change* neighborhood generation heuristic. The simulated annealing (SA) is another class of algorithm often used for solving blackbox optimization problems. Again, in SA, the neighborhood generation operator can be viewed as a source of relations. Since relations can be introduced through several means, the SEARCH framework considers a set of relations as an abstract entity, independent of their sources. Among the members of this set of relations, some relations are useful, since they properly delineate the search space and thereby facilitate correct decision making; the rest of the relations do not and

Table 2.6: Class average statistics of the classes defined by relations $###f$ and $#f##$, computed based on the sample set.

###f		##f#	
Class	Avg.	Class	Avg.
###1	1.5	##0#	1.5
###0	1.33	##1#	1.33

Table 2.7: Class average statistics of the classes defined by relations $fff#$, computed based on the sample set.

fff#	
Class	Avg.
111#	3.0
100#	1.5
101#	0.5

therefore, they are not as useful from the perspective of optimization. The smaller the set of relations, the more likely it is that these relations will fail to properly delineate for different kinds of problems. A particular set of relations may be suitable to properly delineate a certain kind of search space, although it may fail to do so for other problems. Therefore, the "richer" the set of relations, the more likely it is that some of them will properly satisfy the delineation requirement. The term "richer" needs some explanation. Simply increasing the size of the set of relations introduces a different problem. For every BBO, an algorithm has to determine which relations satisfy the delineation requirement. Therefore, it is a search problem. The larger the set of relations considered by the algorithm, the more expensive the search for appropriate relations becomes. This is clearly a trade-off and the term "richer" reflects a qualification of the ease to find a relation that properly delineates the search space. One way to quantify this qualitative term is to measure the ratio of the relations that properly delineate the search space and all possible relations under consideration of the algorithm. This ratio will be called the *delineation-ratio* of the problem with respect to the chosen class comparison statistic and the set of relations. SEARCH realizes the importance for searching relations that properly delineate, and it makes an explicit attempt to capture its effect on the overall success of the algorithm.

In a sampling-based algorithm, the classes can only be compared based on a small subset of the set of all members of these classes. Therefore, the detection of better relations and classes has to depend on the quality of sampling; decision error is possible unless the sampling is extensive enough. First of all, SEARCH tries to construct an approximate ranking among the classes defined by a relation. Let us illustrate this using the sample set given in Table 2.1 for our example problem. Table 2.6 shows the class average statistic for the classes defined by relations $###f$ and $##f#$ based on the sample set shown in 2.1. Based on the limited information from the sample set relations $###f$ and $##f#$, we shall rank the classes ###1 and ##0# highest respectively. Since we already know that 1111 is the optimal solution of this problem, clearly relation $##f#$ is providing a ranking that can lead to failure in finding the optimal solution. As we saw earlier, relation $##f#$ is one of those relations that do not satisfy the delineation requirement. However, in absence of any prior information, determining which relation produces a correct ranking is a stochastic decision-making problem. One possible way is to define a measure that can be used to compare different relations, just like the class comparison statistic. For example, one such measure may favor those relations that rank classes in such a way that the top-ranked class is much "better" than the second-best class in the sense of the class comparison statistic; let us illustrate this statistic using our example problem. Consider the relations $##f#$ and $fff#$ and their sample class average statistic shown in Tables 2.6 and 2.7, respectively. Note that table 2.7 shows only three classes since no information about the other classes can be gathered from the given sample set. The difference between the class average statistics of ##1# and ##0# is 0.17. Similarly the difference between the class average statistics of 111# and 100# is 1.5. Since $1.5 > 0.17$ according to this relation comparison statistic, $fff#$ is better than $##f#$. Different measures for comparing relations can be defined. SEARCH recognizes the importance of both correct ranking of classes and correct choice of relations and considers their effect on the overall success probability of an algorithm.

Once the classes are ranked and the corresponding relation is hypothesized to be one that properly delineates the search space with high confidence, then the higher-ranked classes are selected for future consideration and the low-ranked classes are discarded. However, detections of appropriate relations and selection of better classes are not alone sufficient. This information about the classes that are concluded to be promising and those that are discarded from further

consideration needs to be exploited during future evaluation of new relations. Let us illustrate this idea using our example problem. Let us say that we have identified that relations $fff\#$ and $\#\#\#f$ properly delineate the search space with $M_i = 2$ and $M_i = 1$, respectively. The top two ranked classes in $fff\#$ are identified as 111# and 100#. Similarly let the top ranked class in relation $\#\#\#f$ be ###1. Now let us consider the evaluation of the relation $ffff$. This relation divides the space into 16 singleton classes, in which each of the classes is basically a ground member of the search space. However, the class ranking information from relations $fff\#$ and $\#\#\#f$ can be used to conclude that for relation $ffff$ only the following classes need to be considered: $111\# \cap \#\#\#1$ and $100\# \cap \#\#\#1$. In other words, although the relation $ffff$ actually defines 16 singleton classes, the only two classes that need to be considered are 1111 and 1001, and the rest of the classes can be neglected. Note that this pruning of classes is permissible only as long as our hypothesis about the decomposiblity of relation $ffff$ into relations $fff\#$ and $\#\#\#f$ is correct. This process of exploiting the information about the classes defined by some relations, for pruning out some classes from a different relation by computing set intersections, plays an important role in SEARCH. SEARCH gives this process a particular name—*resolution*.

Resolution can be implemented in several ways. For example, in SA, as the algorithm moves from one state to another, some features of the previous state may remain unchanged, and therefore, the new state may be a combination of some old and new features. In SAs the resolution of class features is implicitly distributed over time. In genetic algorithms (GAs), the crossover operator directly combines the features of parent strings to produce offsprings. In a decision tree such as ID3 (Quinlan, 1986), the individual features are sequentially considered, and gradually, a hierarchical decision tree is developed that can be used for classification. The hierarchy of features in a decision tree describes how features should be combined with one another and defines the resolution process in SEARCH. As we see, resolution can be implemented in several ways. However, since the underlying process is very similar to set intersection, quantitative analysis of SEARCH will treat it as an abstract intersection operation. Therefore, in SEARCH, resolution computes set intersection among different classes.

The ultimate objective of a BBO algorithm is to find the optimal solution. BBO algorithms that directly search for optimal solution in the sample by updating their best estimate return the answer found directly from the sample space. However, in SEARCH, it is quite different.

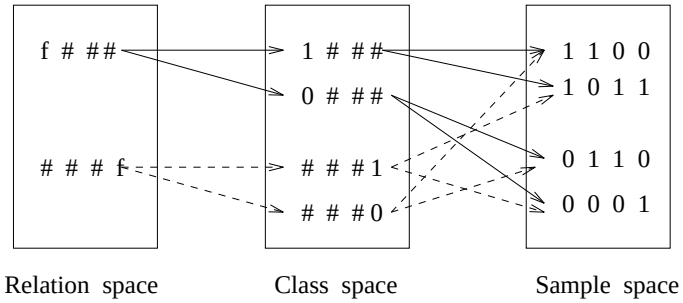


Figure 2.2: Decomposition of blackbox optimization in SEARCH.

SEARCH finds the solution from the class space. When a singleton class is generated that ranks highest, SEARCH returns that class as the optimal solution. Singleton classes can either be produced by resolution when there is some degree of decomposibility or by the relations themselves in the worst possible case.

The above discussion presented an informal account of the different aspects of the SEARCH framework. The remaining part of this section recapitulates the main points.

SEARCH explicitly reduces BBO into searching along three dimensions: (1) relation space, (2) class space, and (3) sample space. Figure 2.2 illustrates this decomposition for a portion of the 4-bit example problem. The relation space contains the relations that are under consideration of an algorithm for classifying the search space. Some of them properly delineate the search space, and some of them do not. Therefore, the algorithm must search for the former kind of relation from the relation space. The class space contains the different set of classes defined by each of these relations. SEARCH tries to construct an ordering among the classes belonging to a particular relation for detecting the class that contains the optimal solution. This search for better classes constitutes another dimension of SEARCH. The sample space contains the sample set that is used to evaluate the classes. Since samples are used to evaluate specific classes, the sample generation process needs to be controlled. Moreover, the same sample set can be used to evaluate the classes belonging to different equivalence relations. This is simply because different relations divide the same search space in different ways. Therefore, the same set of samples just needs to be arranged differently for evaluating different relations. This has a close relation with the so-called *implicit parallelism* in GAs, noted by Holland (1975). This will be elaborated further in later chapters. The resolution computes the intersection classes of

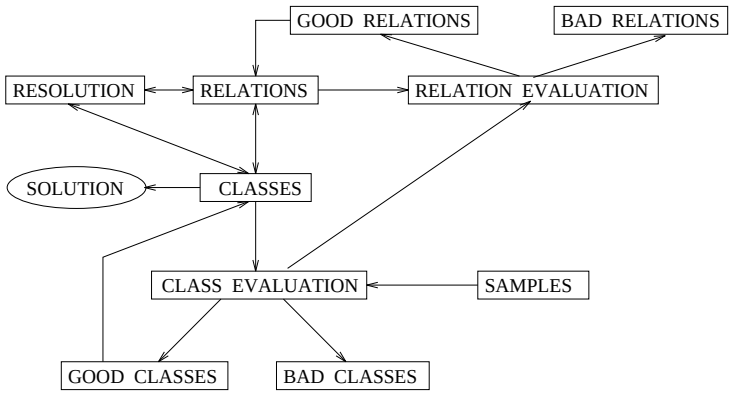


Figure 2.3: Different components of SEARCH framework.

two given classes and thereby exploits any possible decomposibility among the relations. The class space returns the optimal solution, which is a singleton class.

Figure 2.3 shows a process-oriented picture of SEARCH. Relations, classes, and samples are viewed as separate entities. Relations are evaluated based on how they classify the search space. Classes are evaluated using samples taken from the domain of optimization. Resolution eliminates irrelevant classes from the set of all classes defined by a relation. Good relations are preserved and bad relations are discarded. Similarly, in the class space, good classes are selected and bad ones are rejected. The optimal solution is returned from the class space.

This informal presentation has primed the pump with a concrete, if somewhat loose, exposition of the basic ideas in SEARCH. The use of a bit-string example might make it seem as though the framework to be developed is limited to binary representation or some specific algorithm. This is not the case. I will consider different algorithms with different kinds of source of relations in the due course, which will convince the reader about the generality of the underlying concepts. I now turn to a formal development of a general framework—SEARCH.

2.3 SEARCH: The Formal Development

The previous section laid out the SEARCH perspective of BBO as a process of searching for better relations and classes. The presentation was informal and designed mainly for conveying the major concepts. Moreover, the ideas were conveyed mostly using an example problem in sequence representation. However, they can be rigorously cast on a more general ground, which

is the objective of this section. First, Section 2.3.1 presents a formal overview of different aspects of SEARCH. Unlike the previous section, the overview in this section will be given using the formal symbols that will set the stage for the subsequent analysis of SEARCH. Section 2.3.2 presents a detailed description of the different components of SEARCH and formalizes the definitions of different terms introduced earlier. It first addresses the classification, ordering, and selection of classes. Next, it considers the process of selecting better relations. Finally, the resolution process is described.

2.3.1 Overview

Here we overview the major components of SEARCH:

1. classification of the search space using a relation
2. sampling
3. evaluation, ordering, and selection of better classes
4. evaluation, ordering, and selection of better relations
5. resolution

Each component is discussed in more detail in the following paragraphs. To do so requires some notation that we shall use throughout the remainder of the chapter. A relation is denoted by r_i, where i is the index of the set of all relations, Ψ_r, under consideration of the algorithm. Let C_i be the collection of subsets, created by relation r_i. The set of relations S_r actually used by an algorithm to solve the given BBO is a subset of Ψ_r. Denote the members of C_i by $C_{1,i}, C_{2,i} \cdots C_{N_i,i}$, where the cardinality of the class C_i is $\|C_i\| = N_i$. Therefore, C_i is a collection of classes.

Once the relation is used to construct C_i the next step is to evaluate the classes in C_i. To do that we need samples from the domain of optimization. A perturbation operator $\mathcal{P}$ is defined as an operator that generates new samples. This operator can be either a random sample generator or a smarter one that exploits information from the relation, class, and sample memory.

The next step is to construct an ordering among the classes in C_i. To do so, we need a way to compare any pair of classes. A statistic $\mathcal{T}$ can be computed for each of the classes, and they

may be compared on this basis. This statistic will be called a *class comparison statistic*. This class comparison statistic can be used for computing a tentative ranking among the classes in C_i. For certain choices of $\mathcal{T}$, some classes may not be compared with other classes. This means that sometimes a total order may not be constructed. Therefore, in general, a statistic $\mathcal{T}$ can be used to construct a partial order on C_i. Let us denote this partially ordered collection by $C_{i[]}$. Once the ordering is constructed, the next goal is to select some $1 \leq M_i \leq \|C_i\|$ top ranked classes from $C_{i[]}$. M_i represents the total number of top ranked classes that will be selected for future considerations. The exact choice of M_i depends on the decision error probability in choosing an appropriate relation and ordering construction among the classes. For example, if sampling is insufficient, the ordering of classes cannot be relied upon with high confidence, and drastic elimination of classes may not be appropriate. Therefore, a relatively larger value of M_i may be used. These M_i classes constitute the updated version of the class search space.

Next, this ordering among the classes is used to evaluate the relation r_i itself. Different kinds of statistics can be used to compare relations with one another. I denote this relation comparison statistic by $\mathcal{T}_r$ and call it a *relation comparison statistic*. This statistic for relation r_i is now computed. The set of all relations currently under consideration is ordered based on this statistic. Note that, again, this ordering does not have to be a total ordering. The top M_r relations are kept for future consideration and the rest are discarded, in a manner very similar to what we did for the classes.

Not all the classes defined by a relation need to be considered. As more and more relations are evaluated, the information gathered may be used to prune out different classes before evaluating a new relation. Let r_0 be a relation that is logically equivalent to $r_1 \wedge r_2$, where r_1 and r_2 are two different relations; the sign $\wedge$ denotes logical AND operation. If either of r_1 or r_2 was earlier found to properly delineate the search space with certain value of M_i, then the information about the classes that are found to be bad earlier can be used to eliminate some classes in r_0 from further consideration. Blackbox algorithms often implement a resolution-like process to take advantage of any such possible decomposibility. If the chosen relation r_i can be decomposed into a collection of different relations, denoted by $\cup_k r_k$, then resolution can eliminate bad classes using the information collected from possible earlier evaluations of some relations in $\cup_k r_k$.

Repeated iterations of the above steps result in gradual focusing into those regions of the search space which look better using the chosen class and relation comparison statistics. The set of all these relations $r_i, r_{i+1}, \ldots$ used to solve the problem is denoted by S_r. Whether or not the algorithm approaches the globally optimal solution, depends on success in finding proper relations, better classes, and sufficient sampling.

The following section presents a detailed formal description of the different aspects of the SEARCH framework.

2.3.2 SEARCH: The detailed picture

The objective of this section is to present a quantitative picture of SEARCH and formalize the definitions introduced earlier. The definition of a better relation requires defining what we mean by better classes. Therefore, the decision making in the class space is considered first, in Section 2.3.2.1. Section 2.3.2.2 considers the class selection process. This is followed by Section 2.3.2.3 that discusses the relation search. Finally, Section 2.3.2.4 presents the resolution process of SEARCH.

2.3.2.1 Classification and ordering of classes

This section considers the decision-making process among the classes. Classification of the search space requires defining relations. A relation can be defined using different sources, such as operators and representation. In this section I assume no specific source of relations and simply consider Ψ_r, a set of relations, as an abstract entity provided to the search process. However, I continue to give illustrative examples whenever required, using relations defined by sequence representation.

Let C_i be the collection of classes created by some relation r_i. Denote the members of C_i by $C_{1,i}, C_{2,i}, \ldots C_{N_i,i}$, where $\|C_i\| = N_i$. Once a relation r_i is used to define C_i, the collection of classes, each of its members needs to be evaluated first. Since we are interested in the relative "goodness" of the classes with an ultimate goal to pick up some and reject the rest, a statistic that compares any two classes can serve our purpose. If $\mathcal{T}$ is the class comparison statistic used to compare any two subsets $C_{j,i}$ and $C_{k,i}$, then given any two subsets, there must exist an algorithm ? that returns the resulting order among the subsets when compared on the basis of $\mathcal{T}$. It may also be possible that the two classes cannot be compared based on $\mathcal{T}$. The

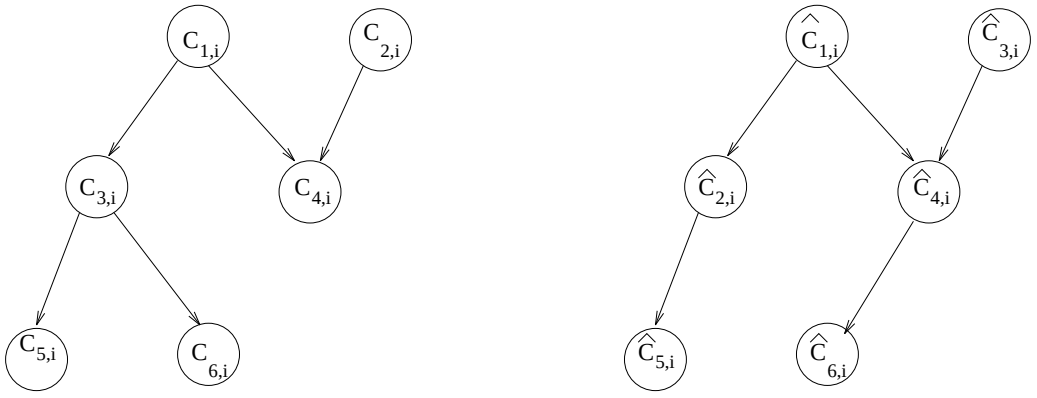


Figure 2.4: Hasse diagram representation of C_i *(left)* and $\hat{C}_i$ *(right)*.

relation constructed among two ordered subsets of C_i is represented by $\leq_{\mathcal{T}}$. In other words, when $C_{j,i}$ and $C_{k,i}$ are compared to each other, then either $C_{j,i} \leq_{\mathcal{T}} C_{k,i}$ or $C_{k,i} \leq_{\mathcal{T}} C_{j,i}$, or they cannot be compared. When C_i is partially ordered on the basis of $\leq_{\mathcal{T}}$, it can be represented by a Hasse diagram. Figure 2.4 (left) illustrates this representation. In a Hasse diagram, the vertices are the members of a poset (partially ordered set); $C_{1,i}$ is drawn above $C_{2,i}$ if and only if $C_{1,i}, C_{2,i} \in C_i$ and $C_{2,i} \leq_{\mathcal{T}} C_{1,i}$. We can say that $C_{1,i}$ covers $C_{2,i}$ if $C_{1,i}, C_{2,i} \in C_i$, $C_{2,i} \leq_{\mathcal{T}} C_{1,i}$, and no element $C_{3,i} \in C_i$ satisfies $C_{2,i} \leq_{\mathcal{T}} C_{3,i} \leq_{\mathcal{T}} C_{1,i}$. The depth of a node, $C_{j,i}$ in a Hasse diagram is the minimum number of links that need to be traversed to reach $C_{j,i}$ from any node at the highest level. Note that this ordering depends on the chosen class comparison statistic.

Let us consider our 4-bit problem again to illustrate these ideas. The partition $ff\#\#$ divides the search space into four classes, 11##, 10##, 01##, and 00##. If we compare these classes on the basis of the mean values of the objective function values of the members calculated in Table 2.8, then they can be ordered as shown in Figure 2.5 (left). Now define a statistic $\mathcal{T}$ such that $C_{2,i} \leq_{\mathcal{T}} C_{1,i}$ if the largest objective function value of $C_{2,i}$ is less than or equal to the minimum objective function value of $C_{1,i}$. The minimum and maximum values of the classes 11##, 10##, 01##, and 00## are shown in Table 2.8. Now note that the partial order constructed using this statistic as shown in Figure 2.5 (right) is different from the partial order shown in Figure 2.5 (left).

In a sampling-based search, the partial-order construction process is based on a finite set of samples taken from each of the subsets, $C_{1,i}, C_{2,i}, \ldots C_{N_i,i}$. Let us denote the approximate

Table 2.8: Different class comparison statistics for the classes in $ff\#\#$.

Class	Average	Min	Max
$11\#\#$	2.0	0	4.0
$10\#\#$	1.0	0	2.0
$01\#\#$	1.0	0	2.0
$00\#\#$	2.125	1.0	3.0

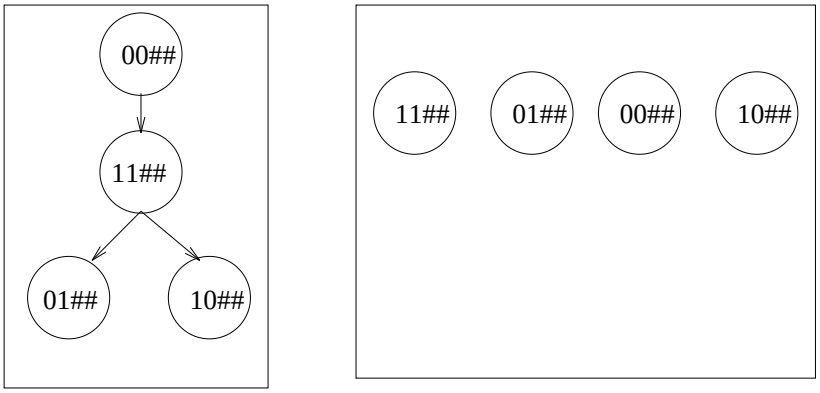


Figure 2.5: Ordering among classes for different class comparison statistics: *(left)* Comparison by average of objective function value. *(right)* $C_{2,i}$ is less than $C_{1,i}$ if the minimum objective function value of $C_{1,i}$ is greater than maximum value of $C_{2,i}$. Table 2.8 presents the corresponding statistic measures of the classes considered here. This figure illustrates that the ordering among the classes can change depending upon the choice of the particular statistic.

descriptions of these classes using the sample sets by $C_{1,i}, C_{2,i}, \ldots C_{N_i,i}$ by $\hat{C}_{1,i}, \hat{C}_{2,i}, \ldots \hat{C}_{N_i,i}$. Let $C_{i[\,]}$ be the ordering of classes from relation i. Denote the class at rank b from the bottom of this ordering by $C_{[b],i}$. This means the top ranked class in this ordering is denoted by $C_{[N_i],i}$. he partial ordering constructed using the sample estimates may be different from the actual ordering. Figure 2.4 (right) shows that the partial ordering constructed from sample estimates may differ from the actual ordering.

2.3.2.2 Selection of better classes

Once the classes are partially ordered based on $\leq_{\mathcal{T}}$, the next immediate objective is to select M_i "top" subsets. Since $C_{i[\,]}$ is a partial order, the notion of "top" needs to be properly defined. This is an implementation-specific issue. One possible way to define this may be

based on the depth of a subset in the Hasse diagram. For the current purpose, I assume that there exists a subroutine TOP$(C_{i[\,]}, M_i)$ which returns the set of "top" M_i subsets from the collection $C_{i[\,]}$. In our 4-bit example problem, with class objective function value average as the comparison statistic (as shown in Figure 2.5 (left)), TOP$(C_{i[\,]}, 2)$ will return the top two classes, $\{00\#\#, 11\#\#\}$. Denote the particular subset that contains x^*—the globally optimal solution—by $C_{*,i}$. If we denote the ordered collection of sample sets $\hat{C}_{1,i}, \hat{C}_{2,i}, \ldots \hat{C}_{N_i,i}$ by $\hat{C}_{i[\,]}$, then we would like $\hat{C}_{*,i}$ to be one among the collection of classes returned by TOP$(\hat{C}_{i[\,]}, M_i)$. Unfortunately, this is very unlikely, unless $C_{*,i}$ itself is not within TOP$(C_{i[\,]}, M_i)$. This sets the stage for introducing the notion of inherently better or worse relations with respect to a given problem, a class comparison statistic, and memory size. This is considered in the following section.

2.3.2.3 Selection of appropriate relations: The delineation property

A relation is not appropriate with respect to the chosen class comparison statistic and the BBO problem if the class containing the optimal solution is not one among some top-ranked classes, ordered based on this statistic. If the class $C_{*,i}$ is not among the top M_i classes, the algorithm is not likely to succeed (neglecting any chance that may rank $\hat{C}_{*,i}$ higher than its actual ranking). Let us quantify this requirement of a relation to be appropriate by a function $DC(r_i, \mathcal{T}, M_i)$. This function returns a one if $C_{*,i} \in$ TOP$(C_{i[\,]}, M_i)$; otherwise, it returns a zero. This will be denoted by $DC()$ in short (DC stands for Delineation Constraint), unless otherwise required.

Definition 1 (Proper delineation) *: For a given BBO problem, a relation r_i, a class comparison statistic $\mathcal{T}$, and a memory size, M_i, if $DC(r_i, \mathcal{T}, M_i) = 1$, we say that r_i properly delineates the search space.*

Let us consider our 4-bit problem again. Consider the relation $f\#\#\#$. As shown in Table 2.5, when the class average is used to compare the classes $1\#\#\#$ and $0\#\#\#$, the class $0\#\#\#$ ranks higher than $1\#\#\#$. When $M_i = 1$, TOP$(C_{i[\,]}, 1)$ does not contain the class $1\#\#\#$, which actually contains the desired solution. Therefore, we say that the relation $f\#\#\#$ does not properly delineate the search space with respect to the class average statistic. On the other hand, relations $\#\#\#f$ and $fff\#$ do satisfy this requirement using this statistic. This *delineation requirement* plays an important role in SEARCH processes. It essentially qualifies or

disqualifies a relation for a particular search problem. If a relation does not properly delineate the search space, there is very little chance that the class with the best solution will be detected. Therefore, for a given class comparison statistic, whether or not a relation is appropriate can be directly quantified based on this characteristic function. However, in reality the algorithm does not know this constraint. The algorithm has to decide whether or not a relation properly delineates the search space from the limited number of samples taken from the search space. Therefore, determining whether or not a relation properly delineates is again essentially a decision-making problem.

Given a finite set of samples from the search space, a class comparison statistic, $\mathcal{T}$, the memory size M_i, and a relation r_i, the goal is to determine whether a relation classifies the search space in such a way that $C_{*,i}$ is in $\text{TOP}(C_{i[]}, M_i)$. Since the problem is now reduced to a decision-making problem instead of the previous binary characteristic function, we can approach it using the same strategy that we took for selecting better classes. In other words, we can start comparing relations, estimate how well a relation would satisfy the delineation requirement compared to another relation, and choose the better relations. This problem is similar to the class selection problem; the only difference is that now we are trying to choose *better relations* instead of better classes. The first question is: How do we compare two relations? While comparing two classes, we needed a class comparison statistic, $\mathcal{T}$. The same thing can be done for relations. Let us denote a *relation comparison statistic* by $\mathcal{T}_r$. This statistic is used to compute an ordering among the relations. Denote this ordering relation by $\leq_{\mathcal{T}_r}$. The ordering among the relations in Ψ_r may not remain the same when relations are compared based on a limited number of samples. In other words, if $r_j \leq_{\mathcal{T}_r} r_i$, then it is not necessarily true that $\hat{r}_j \leq_{\mathcal{T}_r} \hat{r}_i$; I denote a relation r_i when compared based on limited sampling by $\hat{r}_i$. Figure 2.6 shows two orderings of the relation space, one depicting the actual ranking and the other based on sampling information.

I shall assume that there are at least $\|S_r\|$ relations in Ψ_r, needed to solve the problem, that satisfy the delineation constraint. If this is not true, the chosen set of all relations Ψ_r is not appropriate and a new set should be chosen. Before selecting a particular relation, the ordering among the relations is computed and one among the top $\|S_r\|$ relations is chosen. This process of relation selection involves decision making in absence of complete knowledge and

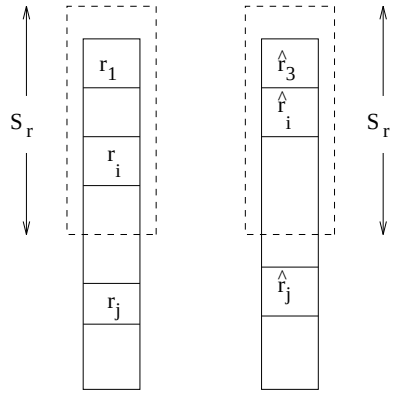


Figure 2.6: Ordering of relations. The left ordering shows the actual ordering and the right one corresponds to the estimated one.

it is therefore susceptible to decision errors. The following subsection describes the resolution process.

2.3.2.4 Resolution of classes

Resolution plays an important role in SEARCH. Resolution takes advantage of possible delineability of relations. Classification of the search space defined by a relation is moderated by the resolution process. If possible, resolution eliminates classes that are not necessary to consider by using the information gathered by previous evaluations of some other relations. Let r_j be a relation that properly delineates the search space with memory size M_j. Let r_i be the relation currently being evaluated, and r_i can be logically expressed as $r_j \wedge r_k$, where r_k is a relation. Resolution of C_i with respect to r_j eliminates those classes of C_i that need not be considered using our knowledge about r_j. This resolved set of classes in C_i can be formally defined as

$$\bigcup_{b=N_j,\ldots N_j-M_j} \bigcup_{a=1,\ldots N_i} C_{a,i} \bigcap C_{[b],j}$$

where the index b varies over the all M_j top ranked classes of relation r_j and index a denotes the different N_i classes in C_i. $C_{[b],j}$ is the rank b member of the ordered collection of classes in C_j and $C_{a,i}$ is the a member of the unordered collection of classes C_i. Let us illustrate the concept using our four bit problem. Consider relations $fff\#$ and $\#\#ff$. Assume that we already evaluated $fff\#$ and that it is believed to delineate the search space properly with

$M_i = 1$. The top ranked class is 111# computed based on the sample set, as shown in Table 2.7. While evaluating relation $\#\#ff$, resolution with respect to $fff\#$ can be used to prune out some of the classes in $\#\#ff$. Relation $\#\#ff$ defines four classes: ##11, ##01, ##10, and ##00. According to the above definition, resolution of $\#\#ff$ with respect to $fff\#$ produces the following classes—##11 and ##10. As we see, resolution eliminates classes ##01 and ##00.

The following sections present an analysis of SEARCH with an objective to quantify the decision success probabilities in class and relation selection processes.

2.4 Decision Making in SEARCH

The previous sections presented SEARCH from both informal and formal points of view. They also posed the class and relation selection processes as decision problems in absence of complete knowledge. In this section I analyze these two sources of decision error and combine them to develop an expression for the overall success probability.

Two kinds of decision errors may make the selection of better classes erroneous:

1. The relation used to define collection C_i is such that for the chosen $\mathcal{T}$, the subset $C_{*,i}$ is not in TOP$(C_{i[\,]}, M_i)$. Therefore, despite how well the sampling is done, the selection process will always miss the subset containing x^*, unless $\hat{C}_{*,i}$ is ranked higher by sampling error. A search algorithm needs to determine whether or not a relation does this from a finite number of samples. Therefore, this could be a source of error. Let us call this error the *relation selection error*.

2. Even when $C_{*,i}$ is in TOP$(C_{i[\,]}, M_i)$, sampling error can produce a different partial order structure for $\hat{C}_{1,i}, \hat{C}_{2,i}, \ldots \hat{C}_{N_i,i}$. As a result $\hat{C}_{*,i}$ may not be in TOP$(\hat{C}_{i[\,]}, M_i)$. The sampling error may result in incorrect ordering of the classes and I call this the *class selection error*.

These two dimensions of decision error in BBO determine the success probability. The following sections analyze the success probabilities associated with each of these dimensions. Finally, they are combined to develop an expression for the overall success probability.

2.4.1 Relation selection success

If an algorithm does not properly delineate the search space, it is not likely to select the class containing the optimal solution. Since, in the absence of knowledge, there is no way to know whether a relation satisfies this requirement or not a priori, this can only be estimated based on the sampling information. Relations are ordered based on the measure $\mathcal{T}_r$, and $\|S_r\|$ top relations are selected. Since these top $\|S_r\|$ relations are just the estimated relations that satisfy the delineation constraint, there is the possibility of decision error. If r_i is actually in the top $\|S_r\|$ relations, then the probability that $\hat{r}_i$ will also be within the top $\|S_r\|$ relations depends on correct decision making in the comparison with at least $\Psi_r - \|S_r\|$ relations. Denote a relation which actually does not satisfy the delineation constraint by r_j. If the minimum probability that $\hat{r}_j \leq_{\mathcal{T}_r} \hat{r}_i$ over all possible relations is denoted by, $Pr(\hat{r}_j \leq_{\mathcal{T}_r} \hat{r}_i)_{min}$, the success probability that $\hat{r}_i$ will be one among the top $\|S_r\|$ relations is

$$Pr(CRS \mid r_i) \geq Pr(\hat{r}_j \leq_{\mathcal{T}_r} \hat{r}_i)_{min}^{\|\Psi_r\|-\|S_r\|}, \tag{2.2}$$

where CRS stands for *correct relation selection*. The following subsection considers the decision making in class selection process.

2.4.2 Class selection success

Let us now consider the class selection problem. The probability that the best solution is in any of the selected subsets will be denoted by $Pr(CCS|r_i)$. CCS stands for *correct class selection* and conditional to r_i, reflects its association with relation r_i. Let $Pr(\hat{C}_{j,i} \leq_{\mathcal{T}} \hat{C}_{*,i})$ denote the success probability given that $C_{j,i} \leq_{\mathcal{T}} C_{*,i}$, and let $Pr(\hat{C}_{j,i} \leq_{\mathcal{T}} \hat{C}_{*,i})_{min}$ be the minimum value of $Pr(\hat{C}_{j,i} \leq_{\mathcal{T}} \hat{C}_{*,i})$ over every $\hat{C}_{j,i}$ which has a depth greater than that of $\hat{C}_{*,i}$ and there is a link connecting it to $\hat{C}_{*,i}$.

Now noting that M_i top classes are selected,

$$Pr(CCS \mid r_i) \geq Pr(\hat{C}_{j,i} \leq_{\mathcal{T}} \hat{C}_{*,i})_{min}^{N_i-M_i}$$

This gives the success probability for a particular relation r_i.

2.4.3 Overall success

The overall success probability for all the considered relations in S_r then becomes

$$Pr(CS \mid \forall r_i \in S_r) = \prod_{\forall r_i \in S_r} Pr(CRS \mid r_i) Pr(CCS \mid r_i). \tag{2.3}$$

This equation captures the general idea that will be used in the following sections. As we see, at the top level, the success of a blackbox search algorithm depends on

1. the success probability in finding relations that properly delineate the search space and
2. the success probability in detecting the class which actually contains the desired solution.

At a lower level, the overall success depends on

1. the class comparison statistic, $\leq_{\mathcal{T}}$;
2. the relation comparison statistic, $\leq_{\mathcal{T}_r}$, which clearly depends on $\leq_{\mathcal{T}}$ and memory size M_i;
3. the way samples are generated by perturbation operator, $\mathcal{P}_i$, which may vary with i. A perturbation operator generates new samples during the search. Note that the samples can either be generated randomly or by using some adaptive strategy;
4. the cardinality of C_i, that is, N_i and the number of top ranked classes selected for future consideration, M_i.

The following sections specialize the observations of this framework to a specific class comparison statistic and representation. First, I consider an ordinal class and relation comparison statistic.

2.5 Ordinal Class and Relation Selection

Constructing a total order and selection of some M_i top subsets from that order have been studied using both parametric and non-parametric approaches (Gibbons, Sobel, & Olkin, 1977). If we are willing to make assumptions about the individual distributions of the members of C_i, nice statistics can be formulated to solve this selection problem. However, in the following

discussion, I adopt a non-parametric, ordinal approach (David, 1981) that allows a distribution-free analysis of the relation and class comparison process. The purpose of this section is to derive bounds on the success probability and sample complexity for a quite general ordinal relation and class comparison statistics.

Section 2.5.1 considers an ordinal class comparison statistic and the SEARCH framework is specialized for this statistic. Section 2.5.2 further specializes SEARCH for an ordinal relation comparison statistic. Section 2.5.3 combines the decision making for both better classes and relations; it also bounds the overall success probability. Finally, Section 2.5.4 derives the overall sample complexity and discusses its properties.

2.5.1 Ordinal class selection

As I argued in the previous section, BBO can be viewed as a combined process of search for better relations and better classes defined by each of these relations. Let us first consider the class comparison process from an ordinal perspective. In order statistics any two classes will be compared based on their α quantile of the cumulative distribution function (cdf). A quantile of order α can be defined as the number Φ_α, such that $F(\Phi_\alpha) = \alpha$, where $F(\Phi)$ is the cdf of Φ. This definition of quantile is not fully satisfactory when the cdf is discrete and the α quantile may not be unique. In such cases, however, we can define it as any convex combination of points in the closure of the set $\{\Phi : F(\Phi) = \alpha\}$. To convey the main idea without unnecessary cluttering of symbols, let us assume that the α quantile is unique. We should note that such quantile-based class comparison will always produce a total order on the collection C_i.

Consider the comparison between two classes $C_{j,i}$ and $C_{k,i}$. Assume that we take n samples from each of these classes. I shall denote n samples from the class $C_{j,i}$ by $\hat{C}_{1,j,i}, \hat{C}_{2,j,i}, \ldots \hat{C}_{n,j,i}$; the corresponding objective function values by $\Phi_{1,j,i}, \Phi_{2,j,i}, \ldots \Phi_{n,j,i}$. These n samples can be totally ordered on the basis of their objective function values as follows:

$$\hat{C}_{[1],j,i} \leq_\Phi \hat{C}_{[2],j,i} \leq_\Phi \cdots \leq_\Phi \hat{C}_{[n],j,i}$$

where, $\hat{C}_{[\omega],j,i} \leq_\Phi \hat{C}_{[\beta],j,i}$ if $\hat{\Phi}_{[\omega],j,i} \leq \hat{\Phi}_{[\beta],j,i}$. $\hat{\Phi}_{[k],j,i}$ denotes the k-th order statistic. The sample estimate of the α quantile for the class j is denoted by $y_{\alpha,j}$. Define an integer $\tau = \alpha(n+1)$; then, $y_{\alpha,j,i} = \hat{\Phi}_{[\tau],j,i}$. If $\alpha(n+1)$ is not an integer, we can set τ equal to the largest integer

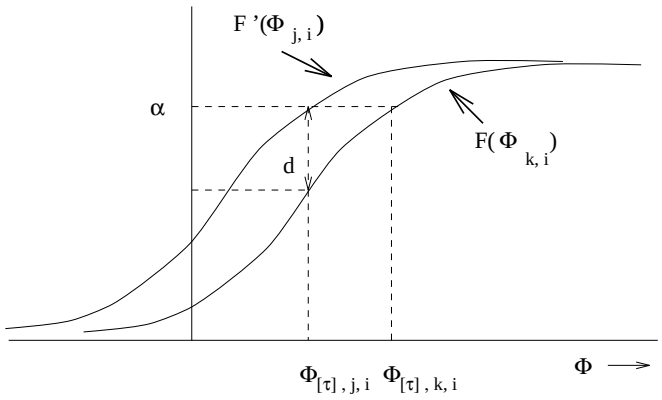


Figure 2.7: Fitness distribution function of two classes $C_{[j],i}$ and $C_{[k],i}$.

contained in $\alpha(n+1)$ and compute $y_{\alpha,j,i}$ as follows:

$$y_{\alpha,j} = [r + 1 - \alpha(n+1)]\hat{\Phi}_{[\tau],j,i} + [\alpha(n+1) - \tau]\hat{\Phi}_{[\tau+1],j,i}$$

This basically interpolates between two adjacent order statistics to approximate the point where the cdf is equal to α. Again, to keep things simpler, I assume that $\alpha(n+1)$ is an integer.

Figure 2.7 shows the cumulative distribution function F' and F of two arbitrary subsets $C_{j,i}$ and $C_{k,i}$, respectively. When these two classes are compared on the basis of the α quantile, then we say $C_{j,i} \leq_\alpha C_{k,i}$, since $\Phi_{[\tau],j,i} \leq \Phi_{[\tau],k,i}$; $\Phi_{[\tau],j,i}$ and $\Phi_{[\tau],k,i}$ are the solutions of $F'(\Phi_{j,i}) = \alpha$ and $F(\Phi_{k,i}) = \alpha$, respectively. Let us define

$$d = F(\Phi_{[\tau],k,i}) - F(\Phi_{[\tau],j,i}).$$

The variable d defines the *zone of indifference*, which is basically the difference in the percentile value of $\Phi_{[\tau],k,i}$ and that of $\Phi_{[\tau],j,i}$ computed from the same cdf F. Figure 2.7 clearly explains this definition.

It can be easily shown that for τ-th order statistics of set $C_{j,i}$ (David, 1981), $\Phi_{[\tau],j,i}$,

$$Pr(\hat{\Phi}_{[\tau],j,i} \leq c') = \sum_{w=\tau}^{n} \binom{n}{w} (F(c'))^w (1 - F(c'))^{n-w}. \tag{2.4}$$

The probability of correct selection among these two classes can be written as

$$\begin{aligned} Pr(\hat{\Phi}_{[\tau],j,i} \leq_\alpha \hat{\Phi}_{[\tau],k,i}) &= \sum_{\forall z} Pr(\hat{\Phi}_{[\tau],j,i} = z) Pr(\hat{\Phi}_{[\tau],k,i} > z) \\ &\geq Pr(\hat{\Phi}_{[\tau],j,i} \leq \Phi_{[\tau],j,i}) Pr(\hat{\Phi}_{[\tau],k,i} > \Phi_{[\tau],j,i}). \end{aligned} \tag{2.5}$$

Now we can write

$$\begin{aligned} Pr(\hat{\Phi}_{[\tau],k,i} > z) &= 1 - Pr(\hat{\Phi}_{[\tau],k,i} \leq z) \\ &= 1 - \sum_{w=\tau}^{n} \binom{n}{w} (F(z))^w (1 - F(z))^{n-w} \\ &\geq 1 - \binom{n}{\tau} F(z)^\tau. \end{aligned} \tag{2.6}$$

Therefore,

$$Pr(\hat{\Phi}_{[\tau],k,i} > \Phi_{[\tau],j,i}) \geq 1 - \binom{n}{\tau} (\alpha - d)^\tau. \tag{2.7}$$

Similarly, we can write that

$$Pr(\hat{\Phi}_{[\tau],j,i} \leq \Phi_{[\tau],j,i}) \geq \binom{n}{\tau} \alpha^\tau. \tag{2.8}$$

Noting that $\binom{n}{\tau} \geq \left(\frac{n}{\tau}\right)^\tau$, and using inequalities 2.5, 2.7, and 2.8, we can write

$$\begin{aligned} Pr(\hat{\Phi}_{[\tau],j,i} \leq_\alpha \hat{\Phi}_{[\tau],k,i}) &\geq \left(\frac{n}{\tau}\right)^\tau \alpha^\tau \left(1 - \binom{n}{\tau} (\alpha - d)^\tau\right) \\ &= \left(\frac{n}{\alpha(n+1)}\right)^\tau \alpha^\tau \left(1 - \binom{n}{\alpha(n+1)} (\alpha - d)^{\alpha(n+1)}\right) \\ &\simeq \left(\frac{n}{\alpha n}\right)^\tau \alpha^\tau \left(1 - \binom{n}{\alpha n} (\alpha - d)^{\alpha n}\right) \\ &= 1 - \binom{n}{\alpha n} (\alpha - d)^{\alpha n}. \end{aligned} \tag{2.9}$$

It can be shown (Cormen, Leiserson, & Rivest, 1990)(pp. 102) that, for $0 \leq \lambda \leq 1$,

$$\binom{n}{\lambda n} \leq 2^{nH(\lambda)}, \tag{2.10}$$

where $H(\lambda)$ is the binary entropy function, $H(\lambda) = -\lambda \log_2 \lambda - (1-\lambda)\log_2(1-\lambda)$. $H(0) = H(1) = 0$ and $H(\lambda)$ takes the maximum value for $\lambda = 0.5$. Using 2.10 and 2.9 we write

$$Pr(\hat{\Phi}_{[\tau],j,i} \leq_{\alpha} \hat{\Phi}_{[\tau],k,i}) \quad \geq \quad 1 - 2^{nH(\alpha)}(\alpha - d)^{\alpha n}. \tag{2.11}$$

If we denote the cdf of the class containing the optimal solution x^*, then define

$$d'' = \min\{F(\Phi_{[\tau],*,i}) - F(\Phi_{[\tau],j,i}) | \forall j\},$$

the probability that the class $\hat{C}_{*,i}$ will be within the top M_i classes is

$$Pr(CCS \mid r_i) \geq [1 - 2^{nH(\alpha)}(\alpha - d'')^{\alpha n}]^{N_i - M_i}. \tag{2.12}$$

Given relation r_i that properly delineates the search space, Equation 2.12 can be used to compute the probability that $C_{*,i}$ will be within the top M_i classes. Before we proceed toward computing the overall correct selection probability, we need to consider the search in the relation space.

2.5.2 Ordinal relation selection

A relation is appropriate if it properly delineates the search space. Determining whether or not a relation satisfies this constraint with absolute certainty is not possible unless we completely enumerate the search space. Therefore, in reality, the characteristic function $DC()$ is replaced by an estimator that measures how likely a relation satisfies delineation constraint. Let us define a measure $\eta : \Psi_r \times 2^C \times 2^{\mathcal{X}} \rightarrow \Re$. 2^C denotes the collection of classes and $2^{\mathcal{X}}$ denotes the sample set. For a given relation r_i, the corresponding set of classes C_i, and a sample set $\mathcal{S}$, this measure $\eta(r_i, C_i, \mathcal{S})$ returns a real value that corresponds to the chances of r_i to satisfy the delineation constraint (i.e. $C_{*,i}$ is a member of $\text{TOP}(C_{i[]}, M_i)$). In short, $\eta(r_i, C_i, \mathcal{S})$ will be written as η_i. This measure will be used to order the equivalence relations $r_i, r_j \in \Psi_r$. Let us again adopt an ordinal approach to compare different relations, just as we did for selection of better classes. For any two relations r_i and r_j, the corresponding η_i and η_j can be treated as random variables. In the class space the random variable was defined to be the objective function value of the samples. Unlike that, here in the relation space the random variable is

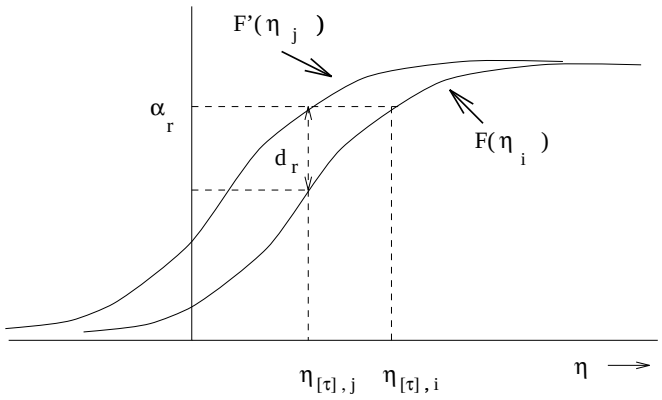


Figure 2.8: Cumulative distribution function of two relations r_i and r_j.

the measure η, which is defined over a collection of classes and a sample set for a given relation relation. Since, for a given r_i, the computation of η_i depends on a tuple from $(2^C \times 2^{\mathcal{X}})$, a collection of n_r such tuples will generate a distribution of different values of η_i. Figure 2.8 shows the cdf of two competing relations r_i and r_j. Let us say that r_i satisfies the delineation constraint and r_j does not.

If we compare these two relations on the basis of some τ_r-th order statistic, the success probability can is computed in exactly the same way that we just did for class comparisons. If α_r be the corresponding percentile,

$$Pr(\hat{r}_{[\tau_r],j} \leq_{\alpha_r} \hat{r}_{[\tau_r],i}) \geq 1 - 2^{n_r H(\alpha_r)}(\alpha_r - d_r^{'})^{\alpha_r n_r} \tag{2.13}$$

where

$$d_r^{'} = \min\{F(\eta_{[\tau_r],j}) - F(\eta_{[\tau_r],i}) | \forall j, \forall i\}$$

where F is the cdf of the relation comparison statistic of relation r_i. $d_r^{'}$ is essentially similar to $d^{''}$, except that this is for relation comparison instead of the previous case of class comparison. In the most general case, a relation needs to be chosen out of the all possible relations in Ψ_r. However, in reality, it may be true that only a subset of Ψ_r is chosen at a time. In the following analyses I consider the general case, in which all relations in Ψ_r are under consideration. Let us assume that among these Ψ_r relations, the set $\Psi_g \subseteq \Psi_r$ contains all the relations that properly

delineate the search space. If relation $r_i \in \Psi_g$, then the probability that r_i will be correctly identified is

$$Pr(CRS \mid r_i \in \Psi_g) \geq [1 - 2^{n_r H(\alpha_r)}(\alpha_r - d_r^{'})^{\alpha_r n_r}]^{\|\Psi_r\| - \|\Psi_g\|}. \tag{2.14}$$

This is the success probability in choosing one good relation. If we need $S_r \subseteq \Psi_r$ relations to solve a problem, we can bound the overall success probability in the relation space as follows:

$$\begin{aligned}
[1 - 2^{n_r H(\alpha_r)}(\alpha_r - d_r^{*})^{\alpha_r n_r}]^{(\|S_r\|(\|\Psi_r\| - \|\Psi_g\|))} &\geq q_r \\
2^{n_r H(\alpha_r)}(\alpha_r - d_r^{*})^{\alpha_r n_r} &\leq 1 - q_r^{1/(\|S_r\|(\|\Psi_r\| - \|\Psi_g\|))} \\
n_r &> \frac{\log(1 - q_r^{1/(\|S_r\|(\|\Psi_r\| - \|\Psi_g\|)})}{H(\alpha_r)\log 2 + \alpha_r \log(\alpha_r - d_r^{*})} \\
n_r &> \frac{\log(1 - q_r^{1/(\|S_r\|(\|\Psi_r\| - \|\Psi_g\|)})}{\alpha_r \log(\alpha_r - d_r^{*})},
\end{aligned} \tag{2.15}$$

where d_r^{*} is a constant such that $d_r^{'} \geq d_r^{*}$ and q_r is the desired bound on the relation selection success probability. Now, noting that $\log(1 - a) \leq -a$ and that both the numerator and denominator of inequality 2.15 are negative numbers, we can write that

$$\begin{aligned}
\alpha_r \log(\alpha_r - d_r^{*}) &= \alpha_r \left(\log \left(1 - \frac{d_r^{*}}{\alpha_r} \right) + \log \alpha_r \right) \\
&\leq \alpha_r \left(-\frac{d_r^{*}}{\alpha_r} + \log \alpha_r \right) \\
&= -d_r^{*} + \alpha_r \log \alpha_r.
\end{aligned} \tag{2.16}$$

For a given class comparison statistic α_r is constant, and therefore, $\alpha_r \log \alpha_r$ is also a constant. Since we are primarily interested in the order of growth of n_r, let us neglect this constant term of Inequality 2.16. Now the Inequality 2.15 can be written as,

$$n_r > \frac{\log(1 - q_r^{1/(\|S_r\|(\|\Psi_r\| - \|\Psi_g\|)})}{-d_r^{*}} \tag{2.17}$$

Inequality 2.17 can be further rearranged. Define *delineation-ratio*,

$$\Omega = \frac{\text{Number of relations in } \Psi_r \text{ that properly delineates}}{\text{Total number of relations in } \Psi_r}$$

$$= \frac{\|\Psi_g\|}{\|\Psi_r\|} \tag{2.18}$$

When this ratio is high, searching for appropriate relations is easier, since most of the members of the relation space are appropriate for properly classifying the search space. Using definition 2.18 and Ineqality 2.17 we can write

$$n_r > \frac{\log(1 - q_r^{1/(\|S_r\| \, \|\Psi_r\|(1-\Omega))})}{-d_r^*}. \tag{2.19}$$

This bounds the overall computational complexity in the relation space. Inequality 2.19 can be further simplified using the approximation $\log(1-x) \approx -x$ for $x << 1$,

$$n_r > \frac{q_r^{1/(\|S_r\| \, \|\Psi_r\|(1-\Omega))}}{d_r^*}. \tag{2.20}$$

This clearly shows that n_r increases as q_r increases and that n_r increases when d_r^* is reduced. Since $q_r \leq 1$, n_r decreases as Ω increases. As the number of relations needed to sove the problem, $\|S_r\|$, increases, n_r also increases. The collection of relations Ψ_r defines the complete search space for relations. The larger the number of relations in Ψ_r, the more computation is required for searching for appropriate relations.

The decision making in the relation and class spaces are combined in the following section.

2.5.3 Overall selection success

Let us now combine the search for better relation and better classes together and compute the overall success probability. Define

$$d^{'} = \min\{F(\Phi_{[\tau],*,i}) - F(\Phi_{[\tau],j,i}) | \forall j, \forall i\}.$$

$d^{'}$ is basically the minimum possible value of d over all classes (index j) which are compared with class containing the optimal solution and all relations (index i) in S_r. Now let us consider the overall class selection success probability given by equation 2.3. Note that the relation $\leq_\alpha$ imposes a total order onto C_i. Define, $N_{\max}$ as the maximum possible value of N_i over all relations in S_r; Let $M_{\min}$ and q_r be the minimum value of memory size M_i and bound on success probability in choosing a relation respectively over all the relations in S_r. In formal

notation,

$$N_{\max} = \max\{N_i | \forall r_i \in S_r\}$$
$$M_{\min} = \min\{M_i | \forall r_i \in S_r\}$$

If d^* is a constant such that $d^{'} \geq d^*$, just like the previously defined d_r^*, then the overall success probability can be bounded as follows:

$$\begin{aligned}
[(1 - 2^{nH(\alpha)}(\alpha - d^*)^{\alpha n})^{(N_{\max} - M_{\min})}]^{\|S_r\|} q_r &\geq q \\
2^{nH(\alpha)}(\alpha - d^*)^{\alpha n} &\leq 1 - \left(\frac{q}{q_r}\right)^{\frac{1}{\|S_r\|(N_{\max} - M_{\min})}} \\
n &> \frac{\log(1 - \left(\frac{q}{q_r}\right)^{\frac{1}{\|S_r\|(N_{\max} - M_{\min})}})}{H(\alpha)\log 2 + \alpha \log(\alpha - d^*)} \\
n &> \frac{\log(1 - \left(\frac{q}{q_r}\right)^{\frac{1}{\|S_r\|(N_{\max} - M_{\min})}})}{\alpha \log(\alpha - d^*)}
\end{aligned} \tag{2.21}$$

The denominator of Inequality 2.21 can be simplified in a manner similar to inequality 2.16. The simplified expression is,

$$n > \frac{\log(1 - \left(\frac{q}{q_r}\right)^{\frac{1}{\|S_r\|(N_{\max} - M_{\min})}})}{-d^*}. \tag{2.22}$$

This inequality bounds the number of samples needed from each class to achieve an overall success probability of q in the combined relation and class spaces; q_r gives the given level of success probability in choosing $\|S_r\|$ relations correctly. The cost of increasing the bound q_r can be realized using inequality 2.19.

2.5.4 Sample complexity

The previous subsection derived closed-form bounds on the overall success probability. This can be directly used to bound the overall **sample complexity**,

$$\text{SC} \leq \frac{N_{\max}\|S_r\| \log(1 - \left(\frac{q}{q_r}\right)^{\frac{1}{\|S_r\|(N_{\max} - M_{\min})}})}{-d^*}. \tag{2.23}$$

This inequality gives the overall sample complexity when the probability to find the globally optimal solution is at least q. This expression can be further simplified using reasonable approximations to clearly explain its physical significance. Since $\left(\frac{q}{q_r}\right)^{\frac{1}{\|S_r\|(N_{\max}-M_{\min})}} \leq 1$ and $\log(1-x) \approx -x$ for $x << 1$, we can approximate inequality 2.23 as follows:

$$\mathrm{SC} \leq \frac{N_{\max}\|S_r\|}{d^*}\left(\frac{q}{q_r}\right)^{\frac{1}{\|S_r\|(N_{\max}-M_{\min})}}. \tag{2.24}$$

Inequality 2.24 presents a clear picture of the contributions of different parameters of the SEARCH framework into the sample complexity. Recall that q is the bound on the overall success probability in the relation and class spaces combined. Clearly, sample complexity SC grows polynomially with q. On the other hand, q_r is the minimum bound in the success probability in choosing all $\|S_r\|$ relations correctly. The cost of demanding higher success probability in the relation space shows up in inequality 2.19. However, as we increase our success probability in the relation space, the overall success probability in the combined relation and class spaces increases. The sample complexity should therefore decrease as success probability in the relation space increases. Inequality 2.24 clearly shows that SC decrease with increase in q_r. Note that the ratio $\left(\frac{q}{q_r}\right)^{\frac{1}{\|S_r\|(N_{\max}-M_{\min})}}$ approaches 1 in the limit as $\|S_r\|(N_{\max}-M_{\min})$ approaches infinity. Therefore, SC grows at most linearly with the maximum index value $N_{\max}$ and the cardinality of the set S_r. Recall that d^* defines the desired region of indifference; in other words, it defines a region in terms of percentile within which any solution will be acceptable. The sample complexity decreases as the d^* increases.

This bound on sample complexity establishes an insight introduced earlier in this chapter. In the beginning of Section 2.2, I argued that BBO can perform no better than random enumeration unless we try to exploit the relations among the members of the search space. Now that we have a closed-form bound on sample complexity, let us investigate the case when no relations are assumed among the members. Saying no relations are assumed essentially means that there exists only one relation in Ψ_r that basically divides the complete search space into a set of singleton classes. For our 4-bit problem representation, this could be the relation $ffff$. This relation divides the search space into 16 singleton classes, which is essentially the complete search space. From the definition of global optima, we know that such a relation always properly delineates the search space. Therefore, $S_r = 1$ and $q_r = 1$. The index of this relation

is same as the caridinality of the search space. So, $N_{\max} = \|\mathcal{X}\|$, where $\|\mathcal{X}\|$ denotes the size of the search space $\mathcal{X}$. Substituting these in Inequality 2.24 we get

$$\text{SC} \leq \frac{\|\mathcal{X}\| q^{\frac{1}{\|\mathcal{X}\| - M_{\min}}}}{d^*}. \tag{2.25}$$

This inequality clearly tells us that the overall sample complexity becomes the size of the search space when we completely neglect all relations that put at least two members together in a class. The only advantage that we get comes from our relaxation in the desired solution quality (d^*) and the overall success probability (q). This confirms that although SEARCH provides one particular perspective of BBO, the importance on relations is fundamental, and it should be emphasized in all possible models of BBO that aspire to guide designing BBO algorithms that perform better than random enumerative search. No BBO algorithm can transcend the limit of random enumerative search without inducing relations among the members.

This sets the background for the coming section. This confirms that induction is an important and essential aspect of BBO. The Probably Approximately Correct (PAC) learning framework (Natarajan, 1991; Valiant, 1984) offers a perspective of induction. It will be interesting to note the correspondences between SEARCH and PAC learning. The following section presents a comparative discussion between them.

2.6 SEARCH and PAC Learning

Inducing relations is an essential aspect of SEARCH. The Probably Approximately Correct (PAC) learning theory (Haussler, 1989; Natarajan, 1991; Valiant, 1984) provides a framework to quantify the computation in inductive learning in a distribution-free, probabilistic, and approximate sense. SEARCH and PAC share some common characteristics, but they also differ in many fundamental aspects. The objective of this section is to point out the main correspondences between these two frameworks.

First, I present a brief review of some of the elementary concepts of PAC framework. Next, I discuss the similarities and the dissimilarities between SEARCH and PAC.

The PAC framework presents a computational theoretic perspective of inductive learning. This framework views inductive learning as a probabilistic process of learning hypothesis which

sometimes may give incorrect results. Although this framework has now led to a separate field with a large volume of literature, no effort will be made to cover all the results. A review of the recent progress in this area can be found elsewhere (Natarajan, 1991). In this section, I shall restrict myself to cover some of the elementary results reported in the PAC literature which will suffice our main purpose—comparing PAC with SEARCH.

Theorem 1 (Blumer, Ehrenfeucht, Haussler, and Warmuth (1987)) *: Let H be a set of hypotheses over a universe U and let S be a set of m training examples drawn independently according to $P(u)$, $\epsilon, \delta > 0$. Then if $\hat{F} \in H$ is consistent with all training examples in S and*

$$m \geq \frac{1}{\epsilon}\left(\log\frac{1}{\delta} + \log\|H\|\right), \tag{2.26}$$

then the probability that $\hat{F}$ has error greater than ϵ is less than δ.

This inequality bounds the sample complexity in PAC. For a given hypothesis space H, acceptable error level ϵ, and failure probability δ, this inequality tells us the minimum number of samples needed to learn a hypothesis with error less than ϵ.

As we noted earlier, the sampling process in blackbox optimization can be viewed as an inductive process. Searching for an appropriate relation can be viewed as an inductive search for a correct hypothesis. However, solving a BBO often requires many such inductive searches, since finding the optimal solution using a single relation is very unlikely for non-trivial problems. Therefore, the PAC framework can be philosophically viewed as a computational process embedded within the SEARCH framework. This argument will be further clear from the following discussion.

When we compare Inequalities 2.26 and 2.23, several observations can be made. First of all, note that both of these frameworks are probabilistic and approximate in nature. However, there are some fundamental differences between how these relaxations are introduced. The δ failure probability of PAC gives the overall bound on the chance to succeed. On the other hand, the success probability in SEARCH in introduced at the level of individual relation and class evaluation processes. Although both q and q_r are defined for bounding the overall success probabilities, the fundamental relaxations originate from the relaxed sampling during the relation evaluation process and the class comparison process.

The ϵ parameter of PAC presents a cardinal relaxation of the solution quality. In other words this relaxation parameter depends on the absolute values of the accuracy of the learned hypothesis. On the other hand, in the SEARCH framework, the relaxation is ordinal in nature, meaning the quality of the solution is determined by its ranking among all the members of the search space.

Both SEARCH and PAC adopt a distribution-free approach for computing the sample complexity. Another interesting similarity between these two can be observed by noting the role of Vapnik-Chervonenkis (VC) dimension in PAC framework. It has been shown elsewhere (Blumer, Haussler, & Warmuth, 1990) that a space of hypotheses H is PAC learnable if and only if it has a finite VC dimension. VC dimension is used as a measure to quantify the learnability of a hypothesis space. The SEARCH framework also has a counterpart of this measure—the delineation constraint. SEARCH requires a set of relations that can be defined over the search space, which must satisfy this constraint for a given class comparison statistic and memory size. If the number of relations satisfying this delineation requirement is too small compared to what is needed to the solve the BBO, success is very unlikely. When representation is used as the major source of relation, this requirement provides one way to quantify what it means to be an appropriate representation.

The following two sections consider simulated annealing—a BBO algorithm and evolution—a natural search process and demonstrate that their underlying computation can be captured using SEARCH.

2.7 SEARCH And Simulated Annealing

Like many other algorithms, simulated annealing (SA) algorithm does not explicitly consider the relations. Therefore, the projection of SA into the SEARCH framework depends on our perspective toward SA as well. Since relations can be defined in many ways, when the relation space is not explicitly specified, identifying it leaves room for speculation. The original version of SA does not emphasize representation. Moreover, the random neighborhood generation operator does not pay enough consideration to the relations and classes defined by the chosen representation.

```
/* Initialization */
T = High_temperature; // Initialize the temperature to a high value
Initialize(x); // Randomly initialize the state
Evaluate(x); // Evaluate the objective function value
{
Repeat
{
    Generate(x'); // Generate new state
    Evaluate(x'); // Evaluate the objective function value
    If ( Metropolis_criterion(x, x') TRUE )
      x = x' // Change state to x'
}
Until (Equilibrium is reached)
  Decrease(T); // Decrease the temperature
}
Until ( T < T_min Or (termination criterion TRUE) )
```

Figure 2.9: A pseudo-code for simulated annealing.

In this section, we therefore choose to view SA as a processor of relations and classes defined by the neighborhood generation operator. The following part of this section briefly discusses different counterparts of SEARCH in the SA.

- **Relation space**: A state x_i and the neighborhood generation operator ($\mathcal{P}$) are the two ingredients of the relations processed by the SA. For a given state x_i, the neighborhood generation operator defines a set of states that can be reached in certain number of steps (s) from x_i. This defines a relation among a certain subset of the search space. Therefore, a relation r_i in SA can be specified by the triple $(x_i, \mathcal{P}, s)$.

- **Class space**: The relation $(x_i, \mathcal{P}, s)$ divides the search space into two classes—(1) the set of states that can be reached from x by applying $\mathcal{P}$ for s number of times and (2) the rest of the search space. This defines the class space for a given relation. Let us denote the first class by $C_{1,i}$ and the second by $C_{2,i}$.

- **Sample space**: The SA processes only one sample at a time. The sample represents the state of the algorithm.

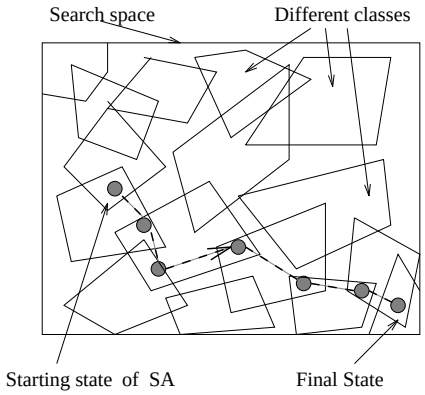


Figure 2.10: The SEARCH perspective of SA.

Searching for the optimal solution in SEARCH also requires different comparison statistics and resolution for combining the features of different classes from different relations. The following discussion points out their counterpart in SA.

- **Relation and class comparison statistics**: Since SA does not explicitly defines the relations and classes, only one statistic, defined by the *Metropolis criterion*, is used for serving both purposes. This comparison statistic varies as the *temperature* changes.

- **Resolution**: Consider the two relations $(x_1, \mathcal{P}, s)$ and $(x_2, \mathcal{P}, s)$, where x_1 and x_2 are two arbitrary states from the search space. Let us denote the set of states that can be reached from x_1 and x_2 by applying $\mathcal{P}$ for s times by $C_{1,1}$ and $C_{1,2}$, respectively. Let x_i be the current state of SA and x_{i+1} be the next state. Now if x_1 and x_2 are such that the $x_i \in C_{1,1}$ and $x_{i+1} \in C_{1,2}$, then the next state, x_{i+1}, is basically a sample from the intersection set of the two classes $C_{1,1}$ and $C_{1,2}$. Generating samples from the intersection set of classes is essentially what resolution does.

The above discussion presents a perspective of SA in the light of SEARCH. Figure 2.10 pictorially depicts this perspective of SA. This figure schematically shows the trajectory of SA within the overlapping classes. As I mentioned earlier, this section presents only one possible way to define classes and relations in SA. Since SA does not explicitly define them, different possibilities may be speculated.

The following section presents the main arguments of an effort to develop an alternate perspective of natural evolution using the SEARCH framework (Kargupta, 1995a).

2.8 SEARCH And Natural Evolution

The SEARCH framework, introduced in this thesis, is developed as a result of an effort to understand the fundamental computation in blackbox optimization. In this section, I take a step in a different direction. I argue that the lessons of SEARCH are also useful in understanding the search in natural evolution. An alternate model of evolutionary computation is proposed by Kargupta (1995a) that establishes the computational role of gene expression in natural evolution. Here, I make an effort to summarize the main arguments of that effort to link SEARCH and natural evolution.

Section 2.8.1 briefly discusses the flow of information in natural evolution. Section 2.8.2 points out the main problem of the existing models of evolutionary computation—lack of emphasis on gene expression. Finally, Section 2.8.3 draws the correspondence between SEARCH and evolution and presents an alternate perspective.

2.8.1 Information flow in evolution

Information flow in evolution is primarily divided into two kinds:

- **extra-cellular flow**: storage, exploration, and transmission of genetic information from generation to generation;
- **intra-cellular flow**: expression of genetic information within the body of an organism.

Each of these will be discussed in the following two paragraphs.

The extra-cellular flow involves replication, mutation, recombination, and transmission of DNA (deoxyribonucleic acid) from parents to offspring. A DNA molecule consists of two long complementary chains held together by base pairs. DNA consists of four kinds of bases joined to a sugar-phosphate backbone. The four bases in DNA are *adenine* (A), *guanine* (G), *thymine* (T) and *cytosine* (C). Chromosomes are made of DNA double helices. For more detailed description, the reader should refer to Alberts, Bray, Lewis, Raff, Roberts, and Watson (1994, Stryer (1988). *Eukaryotes* (most of the developed organisms) have two chromosomes in their cell nucleus, and thus called *diploid* organisms. On the other hand, in *prokaryotes*, such as single-celled bacteria, only one chromosome is present. These are called *haploid* organisms. The DNA sequence is changed by mutation. Crossing over and subsequent recombination result in exchange of base

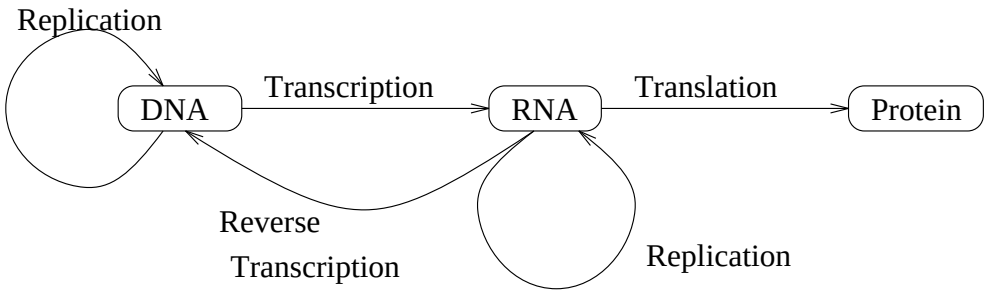


Figure 2.11: Intra-cellular flow of genetic information.

pairs between the parent DNA sequences. These processes result in generation of new DNA sequences. DNA is then transmitted from the parents to the offspring. The DNA is responsible for defining the phenotype of organism and thereby controls the suitability of the organism in the environment. This suitability determines the selective pressure on the organism. Fitter organisms survive, and the rest do not. However, the computation of the phenotype from the DNA—gene expression—is an interesting process in itself. The following paragraph briefly describes the main steps of gene expression, that define the intra-cellular flow of evolutionary information.

Expression of genetic information coded in DNA into the proteins is called the *gene expression*. Expression of genetic information takes place through several complicated steps. However, the major distinct phases are identified as

- transcription: formation of mRNA (ribonucleic acid) from DNA
- translation: formation of protein from mRNA

Figure 2.11 shows the different steps of gene expression. Each of them is briefly described in the following.

Transcription synthesizes messenger RNA (mRNA) from part of the DNA. RNA (ribonucleic acid) consists of four types of bases joined to a ribose-sugar-phosphodiester backbone. The four bases are *adenine* (A), *uracil* (U), *guanine* (G), and *cytosine* (C). Transcription is basically constructing a sequence of bases from another sequence of bases—the DNA. Transcription is initiated by some particular sequences of bases in DNA. They are known as *promoter regions*. For example, in many prokaryotes, the *Pribnow box* sequence TATAAT is a common promoter

region. Transcription continues until it reaches some particular kind of sequences of bases, known as a *terminator region*. RNA polymerase transcribes the portion of DNA between the promoter and terminator regions. Regulatory proteins of a cell can directly control the transcription of DNA sequences. There are two kinds of regulatory proteins:

- gene activator protein, which enhances transcription of a gene, wherever it binds.
- gene repressor protein, which inhibits transcription of a gene.

These proteins usually bind to specific sequences of DNA and determine whether or not the corresponding gene will be transcribed.

Messenger RNA acts as the template for protein synthesis. Proteins are sequence of *amino acids*, joined by peptide bonds. Messenger RNA is transported to the cell cytoplasm for producing protein in the ribosome. There exists a unique set of rules that define the correspondence between nucleotide triplets (known as codons) and the amino acids in proteins. This is known as the *genetic code*. Each codon, comprised of three adjacent nucleotides in a DNA chain, produces a unique amino acid.

Most of the existing models of evolutionary computation do not provide any understanding about the computational role of the intracellular flow of genetic information. The following section points this out.

2.8.2 Problems of the existing views of evolution

Unfortunately, many of the existing computational models of evolution address only the extracellular flow of genetic information. Simple genetic algorithms (De Jong, 1975; Goldberg, 1989; Holland, 1975), evolutionary strategie (Rechenberg, 1973), and evolutionary algorithms (Fogel, Owens, & Walsh, 1966) are some examples. These existing perspectives of evolutionary computation do not assign any computational role to the nonlinear mechanism for transforming the information in DNA into proteins. The same DNA is used for different kinds of proteins in different cells of living beings. The development of different expression control mechanisms and their evolutionary objectives are hardly addressed in these models. They primarily emphasize the extra-cellular flow. The main difference among these models seems to be the emphasis on crossover compared to mutation or vice versa.

Although gene expression is not emphasized very much in most of the popular models of evolutionary computation, several researchers realized its importance. The importance of the computational role of gene expression was first realized by Holland. He described (Holland, 1975) the dominance operator as a possible way to model the effect of gene expression in diploid chromosomes. He also noted the importance of the process of protein synthesis from DNA in the computational model of evolution. Despite the fact that traditionally dominance maps are explained from the Mendelian perspective, Holland made an interesting leap by connecting it to the synthesis of protein by gene signals, which today is universally recognized as gene expression. He realized the relation between the dominance operator with the "operon" model of the functioning of the chromosome (Jacob & Monod, 1961) in evolution and pointed out the possible computational role of gene signaling in evolution (Holland, 1975).

Several other efforts have been made to model some aspects of gene expression. Diploidy and dominance have also been used elsewhere (Bagley, 1967; Brindle, 1981; Hollstien, 1971; Rosenberg, 1967; Smith, 1988). Most of them took their inspiration from the Mendelian view of genetics. The underspecification and overspecification decoding operator of messy GA has been viewed as a mechanism similar to gene signaling in Goldberg, Korb, and Deb (1989). Dasgupta and McGregor (1992) proposed the so-called structured genetic algorithm, which uses a structured hierarchical representation in which genes are collectively switched on and off. This implementation also gathered its primary motivation from gene expression.

However, none of these approaches really addressed the major steps in gene expression primarily identified by the biologists during the last 30 years, and instead quantify their purpose in terms the basic computational principles. Many questions involving the computational benefit from the collective switching of genes or the regulatory mechanisms controlling the transcription of DNA to RNA remain unanswered. In the following section I present a brief account of a recent effort (Kargupta, 1995a) to fill in this lacuna using the lessons from the SEARCH framework. The central argument that evolves out of the following discussion is that the intracellular expression of genetic information plays an important role in evolutionary computation that can be quantified from the SEARCH perspective.

2.8.3 Evolutionary computation: The SEARCH perspective

Despite the fact that nowhere during the development of SEARCH did we consider any idea from biology, it will be interesting to see how it views the evolutionary search in nature. The following part of this section briefly discusses a possible correspondence between SEARCH and natural evolution.

- **Sample space**: DNA constitute the sample space. Crossover and mutation generate new samples of DNA. A population of organisms defines the sample space for the evolutionary search.

- **Class space**: Base sequences of mRNA transcribed in a cell correspond to only a part of the complete DNA. The sequences of amino acids in protein in turn correspond to base sequence in mRNA. The genetic code tells us that there is a unique relationship between the nucleotide triplets of the DNA and the amino acids in the protein. Therefore, if we consider the DNA as a representation defined over the evolutionary search space for life and different forms of life, then the amino acid sequence of a protein corresponds to a class of different DNA; every DNA in this class must have a certain sequence of nucleotides that can be transcribed to that particular sequence of amino acids. Since the genetic code is unique, a particular sequence of amino acids can only be produced by a certain sequence of nucleotides. In other words, the sequence of amino acids in a protein defines an equivalence class over the DNA space.

- **Relation space**: Recall that amino acid sequences in protein are translated from the nucleotide sequences of mRNA. The construction of mRNA is basically controlled by the transcription process. Since an equivalence relation is an entity that defines the equivalence classes, the transcription regulatory mechanism can be viewed as the relation space that defines classes in terms of the nucleotide sequences in mRNA and finally in terms of the amino acid sequences in proteins. Among the different components of this regulatory mechanism, regulatory proteins, promoter and terminator regions play a major role. Regulatory proteins exist as a separate entity from the DNA, but the promoter and terminator regions are defined on the DNA. It appears that there is a distinct relation space comprised of the different regulatory agents, such as activator and inhibitor

Table 2.9: Counterparts of different components of SEARCH in natural evolution.

SEARCH	Natural evolution
Relation space	gene switching mechanism
Class space	amino acid sequence in protein
Sample space	DNA space

proteins. However, it is quite interesting to note that this space also directly makes use of information from the sample space—the DNA. Expression of genetic information in eukaryotic organisms is more interesting than that in prokaryotes. Apart from more sophisticated transcriptional control system, Kargupta (1995a) pointed out that the diploid chromosome can be viewed as a process of relation construction.

These possible relationships between the different spaces of SEARCH and natural evolution are summarized in Table 2.9.

- **Relation and class comparison statistics**: Classes are defined by amino acid sequences in protein. Proteins are directly responsible for almost every functional and organizational behavior of an organism. Proteins are sometimes called the phenotype of an organism because of this reason. The efficacy of a protein is evaluated in terms of the organism's performance, and natural selection assigns a certain selective measure to this process. This can be viewed as a process of evaluating classes. On the relation front, there is already evidence that the settings for the intracellular expression of genetic information evolves during the course of evolution (Alberts, Bray, Lewis, Raff, Roberts, & Watson, 1994). Therefore, there must be some selective pressure toward the appropriate regulatory setting and that can be held responsible for the decision making in the relation space.

- **Perturbation operators**: Crossover, mutation, and gene deletions appear to be the perturbation operators in natural evolution. They change the participating DNA sequence(s) and thereby generate new samples.

- **Resolution**: Crossover and recombination swap different portions of two DNAs. Although the exact mechanism of crossover is quite involved, the main outcome of the pro-

cess is exchange of DNA subsequences between the parent chromosomes. The offspring DNAs are therefore comprised of the features of classes containing the parent DNAs. Therefore, the offspring are samples from the intersection set of the parent classes.

The following section summarizes the major points discussed in this chapter.

2.9 Summary

This chapter started by noting that a BBO algorithm is not likely to perform better than a random search unless some relations are exploited among the members of the search space. This led to the development of the SEARCH framework, which tries to exploit the role of relations in BBO. SEARCH presents an alternate perspective toward blackbox optimization. It views blackbox optimization as a composition of the following different processes:

1. classification of the search space;
2. sampling from the search space;
3. searching for appropriate relations that divide the search space in a suitable way;
4. searching for better classes defined by each of these relations, so that the desired solution is a member of one of these chosen classes;
5. resolution.

Searching along relations and the class spaces has three steps: (1) evaluation, (2) ordering, and (3) selection. Evaluation detects good relations or classes; ordering constructs a ranking based on the evaluation; selection picks up the good ones and discards the bad ones.

A relation is defined to be good if it can classify the search space in such a way that the class containing the globally optimal solution can be detected using the class comparison statistic. This requirement is called proper delineation of the search space. Determining whether a relation satisfies this or not requires decision making in absence of complete knowledge. This is a source of decision error in BBO.

Detecting a good relation in turn requires approximate identification of good and bad classes defined by the relation. This introduces the other dimension of the SEARCH framework,

searching for better classes. Even if the relation properly delineates the search space, decision error can be made during the comparison between two different classes. This is again viewed as a decision problem.

Success in blackbox optimization is tied with the successes along these two different dimensions. After presenting a general description of search from this perspective, I incorporated ordinal class and relation comparison statistics into the framework. A closed form bound in sample complexity is developed for any BBO that can be solved by considering a finite set of relations defined among the members of the search space. The sample complexity grows linearly with the cardinality of the set of relations, the maximum index value of these relations, the desired solution quality (described using order statistics), the desired degree of appropriateness of the relations, and the overall success probability demanded. This expression for sample complexity established my earlier argument that consideration of relations among the members of the search space is unavoidable if an algorithm aspires to perform better than random enumerative search. I specifically showed that the sample complexity approachs the size of the search space when we consider no relations that put at least two ground members into the same class. In this case, the only reduction in sample complexity comes from the relaxation in the desired solution quality and the overall success probability. Therefore, emphasizing the role of relations is fundamental and should be a part of any model of BBO.

This chapter has also noted the correspondences between SEARCH and PAC-learning. In the PAC learning framework, an algorithm seeks an approximately correct hypothesis in a probabilistically correct way. Since hypotheses are nothing but relations, there must be some correspondences between these two frameworks. Although, unlike PAC-learning, the ultimate objective of SEARCH is to find an optimal solution, not the best relation, SEARCH incorporates the search in relation or hypothesis space to surpass the limits of enumerative search. Different components of the bound on sample complexity in SEARCH are compared with their counterparts in PAC. Both PAC and SEARCH are probabilistic and approximate procedures. However, these "probabilistic and approximate" aspects are introduced at two distinctly different levels in SEARCH and PAC. Unlike PAC, SEARCH introduces relaxation at the level of each relation evaluation. I also noted the delineation constraint in SEARCH as a possible counterpart of the VC dimension that characterizes the representation in PAC.

In order to demonstrate the generality of the SEARCH framework, the last part of this chapter considered two case studies. The first one considered the simulated annealing (SA) algorithm in the light of SEARCH. A detailed correspondence between the different aspects of SA with the SEARCH framework is drawn. Next, a brief description of evolutionary computation is presented from the SEARCH perspective. Following Kargupta (1995a), we note that SEARCH offers an alternate model of evolutionary computation that emphasizes the role of intracellular flow of information—the gene expression. Unlike most of the existing models of evolutionary computation, this perspective identifies the DNA→RNA→Protein synthesis as a process of explicit construction of equivalence classes and relations.

SEARCH lays the foundation for the materials in the following chapters of this thesis. In the next chapter, I shall describe the main implications of this framework and how they should be used for developing a comprehensive approach toward solving BBO problems.

Chapter 3

Implications of SEARCH

The SEARCH framework developed in the previous chapter shows an alternate way to view BBO. Different facets of this framework have been addressed from an abstract quantitative perspective. However, those formalisms lead toward meaningful, physical aspects of BBO. In this chapter I present the main physical implications of SEARCH. A framework like SEARCH should contribute to each of the following aspects of problem solving:

1. How do we define an algorithm in this framework? How can we make them efficient?
2. How can we characterize problems that can be solved efficiently and vice versa?
3. What is the user's role in solving a BBO?

The primary objective of this chapter is to answer these questions.

I start the discussion by describing what it means to be a BBO algorithm in SEARCH. This is presented in Section 3.1. This is followed by Sections 3.2 and 3.3, in which two possible ways to make a BBO algorithm efficient are pointed out. Section 3.2 describes the benefits of a bottom-up approach of blackbox search, in which low-order relations are considered first. Section 3.3 discusses implicit parallelism—evaluating relations in parallel at no additional sample complexity. Next, in Section 3.4, I introduce the SEARCH perspective of problem difficulty in BBO. Defining difficult problems sets the background for introducing the class of problems that can be solved in polynomial sample complexity. Section 3.5 presents the class of *order*-k *delineable* problems that can be solved in polynomial sample complexity in SEARCH. Section 3.6 discusses one important role of the user in solving a BBO—defining the source of relations.

Section 3.7 discusses some issues regarding the representation-operator interaction. Finally, Section 3.8 summarizes the main points of this chapter.

3.1 Blackbox Optimization Algorithm in SEARCH

The SEARCH framework decomposed BBO into different components such as the relation space, class space, and the sample space. Searching in these spaces requires some fundamental tools, such as some comparison statistics and a perturbation operator for generating samples. In this section, I list them together and project a complete picture of what it means to be a BBO algorithm in SEARCH.

1. SEARCH views the solution domain through relations, classes, and samples. An algorithm in SEARCH should be provided with a set of relations Ψ_r. Representation in genetic algorithms (GAs), perturbation operators of simulated annealing (SA), and the neighborhood heuristic in k-opt algorithm are some examples of different sources of relations.

2. SEARCH also needs explicit storage for processing relations, classes, and samples. The maximum possible values of M_r and M_i determine the size of the memory for storing relations and classes respectively. In genetic algorithm the population serves as the memory for all three of these spaces. In SA the evaluation of different classes defined using the perturbation operators is distributed over time and only one sample is taken at a time. The state of the SA algorithm serves as the memory for the sample space.

3. Two statistic measures for comparing classes and relations are required. A Selection operator is used for comparing classes in the simple GA. The Simple GA does not really search for better relations. On the other hand, in simulated annealing, the Metropolis criterion is used for comparing two states; this can be viewed as a class comparison statistic.

4. A perturbation operator, $\mathcal{P}$ is required for generating samples. In GA, crossover and mutation generate new samples. SA make use of a neighborhood generator for sample generation.

5. Accepting criterion of success probability, q, and q_r are necessary. Almost every practical application of GA and SA either implicitly or explicitly makes use of an acceptance

criterion for success. Neither simple GA nor SA actually searches for better relations. Neither of them has any explicit criterion like q_r.

6. Required precision in solution quality, d^* is required. Again, in practice, both GA and SA somehow introduce the factor controlling the desired solution quality. Goldberg, Deb, and Clark (1993, Holland (1975) quantified the effect of desired solution quality on the success probability using a parametric approach.

In many algorithms, some of these parameters are not explicitly specified. For example, in random enumerative search, no importance on relations is given. However, even this kind of search can be shown as a special case of SEARCH. As I explained earlier in Chapter 2 during the discussion on sample complexity, the random enumerative search is essentially a search with one and only one relation in consideration that divides the search space into singleton classes. For example, in GA, no explicit specification of class comparison statistic is required. However, this is taken care of by the chosen selection operator. One of the main objectives of this whole effort is to develop a more systematic approach toward the design and use of BBO algorithms by explicitly identifying different components of a BBO algorithm.

SEARCH provides a common ground for developing new blackbox algorithms in the future. Regardless of the motivation and background, any blackbox optimization algorithm should clearly define each of the above listed components. A BBO algorithm should define how it processes relations, classes, and samples. It should state its relation and class comparison statistics. Apart from defining each of them properly, searching in the relation can take advantage of different properties of the relation space. Moreover, the sample generation operator should comply to the decision making of the relation and class spaces.

The following sections consider two possible ways to exploit the structure in relation space.

3.2 Bottom-up Organization of Search

During the development of the SEARCH framework, no prior ordering among the relations in Ψ_r is assumed. We did not care, as long as a relation satisfied the delineation constraint. In this section, I show that imposing a certain kind of ordering among the relations may lead to some computational benefit. First, I define an ordering among the relations based on their *order* and then show that considering relations according to this ordering is advantageous.

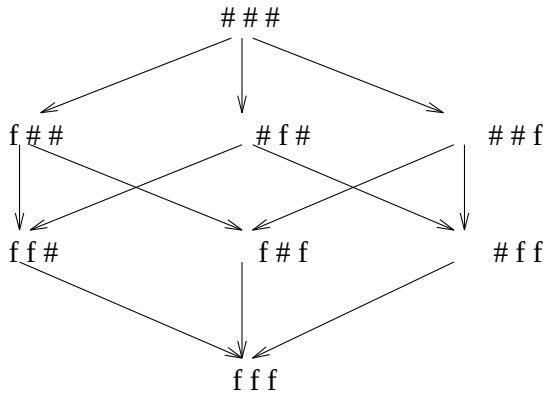


Figure 3.1: Hasse diagram representation of the set of equivalence relations ordered on the basis of a $<_o$.

The order of an equivalence relation $r_i \in \Psi_r$ can be defined as the logarithm of its index with some base α. I shall denote this by $o(r_i)$. The index of r_i is then $N_i = \alpha^{o(r_i)}$. Clearly, the index grows exponentially with $o(r_i)$. The set of all these equivalence relations Ψ_r can be partially ordered on the basis of a $<_o$, defined as follows. If $o(r_i) < o(r_j)$, then we say $r_i <_o r_j$. Figure 3.1 shows a Hasse diagram representation of such an ordering in a $\ell = 3$ representation scheme, which has a depth of 3. Choosing a low-order relation at the initial stage of search when no regions are pruned out from future consideration is computationally advantageous. The following discussion rationalizes this conclusion.

The fundamental idea is quite simple. Comparing classes can be relaxed as we did in the previous chapter; two classes may be ordered with high confidence from a sample set. However, completely discarding a class without taking a single sample can hardly be justified. Note that construction of an ordering among N_i classes requires $\binom{N_i}{2}$ comparisons. Since N_i grows exponentially with the order of the relation, considering a relation of *order* in $O(\ell)$ demands exponential complexity computation. Therefore it is essential that the indices of the relations considered during the initial stages of the search when no regions are discarded are bounded by some constant. The argument can be driven home using an example. Consider an aribitrary problem in 3-bit sequence representation The order one relation $f\#\#$ has an index value of 2. On the other hand, the index of an order two relation $ff\#$, is 2^2. If a search algorithm starts its search with relation $ff\#$ then the number of comparisons needed to construct an ordering is going to be higher.

Therefore, the order of the relation used at the initial stage of the search must be bounded by a constant, k_c; otherwise the complexity of the ordering process will itself be exponential in ℓ. Let us denote the set of relations S_r, when ordered on the basis of the sequence, they are considered by the algorithm by $\gamma(S_r)$; also define $\gamma(\Psi_r)_{<_o}$ to be the set Ψ_r partially ordered based on the order of a relation.

One possible way to keep the complexity under control is to make $\gamma(S_r)$ and $\gamma(\Psi_r)_{<_o}$ order-preserving. In other words, if $r_1, r_2 \in S_r \subseteq \Psi_r$, then if r_2 comes after r_1 in $\gamma(S_r)$, then $r_2 <_o r_1$ in $\gamma(\Psi_r)_{<_o}$. This gradual transition from low order relations to higher-order ones is what I call bottom-up organization of the search. This points out a computational justification of a widely reported, interesting observation about both natural and artificial complex systems. Complex evolving systems are characterized by the gradual growth of patterns from smaller to larger. A small pattern is the one in which the number of features in common to the members of the observed set is less. This bottom-up approach, often observed in nature, seems to have a fundamental computational foundation.

The following section considers another possibility of exploiting the structure of the set of relations when ordered based on the *order* of the relations.

3.3 Implicit Parallelism: Parallel Evaluation of Equivalence Relations

The discussion on bottom-up organization of search led to the solution of considering the relations with low index values first. The structure of Ψ_r, when partially ordered based on order of the relations, can be further exploited to make the relation evaluation process more efficient. In this section I show that when the poset $\gamma(\Psi_r)_{<_o}$ is not linearly ordered, relations can be evaluated in parallel at no additional sample evaluation. This observation appears to be the underlying principle in the so called *implicit parallelism* (Holland, 1975).

The perspective of the blackbox search as an exploration through $\gamma(\Psi_r)_{<_o}$ opens up an interesting possibility. Parallel exploration along different branches in $\gamma(\Psi_r)_{<_o}$ can be done at no additional cost compared to that along a single branch. **Such parallel evaluation is possible as long as $\gamma(\Psi_r)_{<_o}$ is not a totally ordered set**. When $\gamma(\Psi_r)_{<_o}$ is partially

ordered, there will be relations of the same order. Therefore, all these relations of the same order can be evaluated using the same set of samples.

For example, the evaluation of $\#\#\#f, \#\#ff$ can be performed at no additional computational cost in terms of function evaluations when the relations $f\#\#\#, ff\#\#$ are already evaluated. Little attention will make it obvious. Both $f\#\#\#$ and $\#\#\#f$ divide the complete search space into two different ways. Similarly, the relation $ff\#\#$ divides the same search space in a different way than the one by $\#\#ff$ does. Clearly, the same set of samples used to evaluate classes $1\#\#\#$ and $0\#\#\#$ can be used for evaluating classes $\#\#\#1$ and $\#\#\#0$. No additional function evaluation is needed for a constant confidence requirement; the samples are just needed to be differently partitioned. In general, the sample set needed to evaluate a particular relation r_i of order o_i can be used for all other relations of the same order. This computational leverage can make an algorithm very efficient in solving the class of order k bounded, delineable problems. As stated earlier, these problems can be solved by evaluating a subset of all order k relations, whose intersection set is a singleton set. Since the globally optimal solution can be found by simply taking an intersection among the top ranked classes of this subset of all order k relations, the overall computational cost remains polynomial in the problem dimension and the success probabilities.

At this point one must take a moment to put this argument into proper perspective. Our definition of computational cost has been solely focused on the number of function evaluations, i.e., the number of distinct samples taken from the search space. According to this definition, parallel evaluations of several equivalence relations do not incur any additional cost. However, consideration of every different relation required partitioning the same set of samples in a different way, followed by the computation of the class comparison statistic. Although the sample complexity remains the same, the overall time complexity increases.

The previous three sections described different components of BBO algorithm and pointed two possible ways to exploit the structure in the set of relations provided to the algorithm. These discussions naturally lead the reader to wonder about the computational limits of BBO algorithms from the SEARCH perspective. The following section addresses this important aspect of BBO—problem difficulty.

3.4 Problem Difficulty in SEARCH

SEARCH presents an alternate perspective of problem difficulty in BBO. In this section I first identify the main dimensions of problem difficulty in SEARCH and then precisely define a characterization of difficult problems in SEARCH.

The expression for the sample complexity developed in the previous chapter immediately leads to identifying different facets of problem difficulty in SEARCH. As we saw from Inequality 2.23 the sample complexity grows linearly with the size of the set of relations considered to solve the problem, S_r. Often this size depends on the "size" of the problem; the word "size" defines a parameter ℓ that bounds the search domain. In a sequence representation with constant alphabet size, the length of the sequences needed to represent the search space may be an example of such a size parameter. This finally sets the stage for introducing problem difficulty in SEARCH.

Definition 2 (Problem difficulty in SEARCH) *Given an optimization function* $\Phi : X \rightarrow \Re$ *and a set of relations* Ψ_r*, we call a problem difficult for an algorithm if the total number of samples needed to find the globally optimal solution grows exponentially with* ℓ*,* q*,* q_r*,* $1/d^*$*, and* $1/d_r^*$*.*

The size of the problem is represented by ℓ; q denotes the bound in the overall decision success probability in choosing the right classes; $1/d^*$ defines the quality of the desired solution. Both q and $1/d^*$ together can be viewed as representing the overall accuracy and the quality of the solution found; q_r is the bound in success probability in choosing the right relations, and $1/d_r^*$ represents the desired quality of the relations.

The above definition of problem difficulty in SEARCH can be physically interpreted into the following items:

1. growth of the search space along problem dimension
2. inadequate source of relations and decision making in relation space
3. inaccurate decision making in choosing classes
4. quality of the desired solution and relations

Each of these is briefly discussed in the following.

The search space increases as the problem dimension increases. For many interesting classes of problem the growth rate is exponential. This creates difficulty in solving the problem.

Solving a BBO in SEARCH requires a set of relations that properly delineates the search space. If the number of relations in Ψ_r that properly delineate the search space is small, then success probability in solving the BBO is very small. The previous chapter defined a simple but meaningful measure of the overall appropriateness of Ψ_r, the delineation-ratio. This is the ratio of the number of relations in Ψ_r that properly delineates and the total number of relations in Ψ_r. When this ratio is high, searching for appropriate relations is easier, since most of the members of the relation space are appropriate for properly classifying the search space. This point will be illustrated in a later section of this chapter using some example problems.

The overall decision error in choosing the classes that contain the globally optimal solution will be large unless the sampling is sufficiently rigorous. This is a source of decision error and can cause difficulty.

The quality demanded for the solution can also make a problem difficult. If the desired quality is very high, search in BBO may be difficult. Similarly, if the required measure of appropriateness of relations needs to be very high, searching for relations may be difficult.

In a way this basically defines what it means to be efficient search in SEARCH. Therefore, now it is quite natural to ask whether we can identify some classes of BBO problems that can be solved in polynomial sample complexity. The answer is yes. However, the relation between the set of relations S_r and the problem size ℓ cannot be quantified until we specifically consider relations in physical terms. In the following section I consider the set of relations defined by sequence representation and introduce the class of *order-k delineable* problems that can be solved efficiently in SEARCH.

In the following section I introduce the class of *order-k delineable* problems in sequence representation.

3.5 Order-k Delineable Problems in Sequence Representation

The foregoing analysis treated the source of relations in a general way as an abstract entity. However, defining a specific class of problems and studying its sample complexity in SEARCH require relating S_r with the problem dimensions. In this section I consider the set of relations induced by a sequence representation for defining the class of order-k delineable problems that can be solved in polynomial sample complexity.

Although so far we have frequently used sequence representation for illustrating the concepts with examples, in this section I consider specific properties of the set of relations induced by sequence representation. Therefore, it may be appropriate to review the properties of this representation first.

A sequence representation can be defined as

$$I : X \rightarrow \Lambda^{\ell}, \tag{3.1}$$

where Λ is the alphabet set. A particular position in a sequence may sometimes be called a *locus* (*loci* in plural). The particular letter of the alphabet in a locus is called the *value* of the locus. I will also assume a convention of naming in which the leftmost position is called the first locus, increasing toward the right. This sequence representation induces a set of equivalence relations,

$$\Psi_r = \{f, \#\}^{\ell} \tag{3.2}$$

where f indicates values that must match for equivalence and $\#$ is a wild character that matches any value at a locus. The cardinality of the set of all such equivalence relations $\|\Psi_r\| = 2^{\ell}$.

In sequence representation the order of an equivalence relation $r_i \in \Psi_r$ is defined in the same way as we did before; we just choose $\alpha = \Lambda$, as the base of the logarithm. The index of r_i is then, $N_i = \Lambda^{o(r_i)}$.

As we see from Figure 3.1, Ψ_r can be structured as a lattice based on the relation $<_o$. For an ℓ loci representation, the depth of the diagram is always ℓ. The lowest level at order ℓ, the

relation divides the search space into singleton sets. Each of these singleton sets can be reached in at most ℓ steps. However, the problem is that when $o(r_i) = O(\ell)$, $N_i = O(2^\ell)$. Here O is used to denote the order of complexity. Clearly, if we want to solve any arbitrary problem using a ℓ-loci sequence representation, in the worst case, the complexity is exponential.

It will be interesting to know what classes of problems can be solved in polynomial sample complexity. In the following I define a class of order k (a constant) delineable problems which can be solved in sample complexity polynomial in q, q_r, $1/d^*$, $1/d_r^*$, and the problem size ℓ.

Definition 3 (Class of order k delineable problems) *: Let us define a subset of Ψ_r containing every order-*k *relation as follows: $\Psi_{\{o(r)\leq k\}} = \{r_i : o(r_i) \leq k \quad \& \quad r_i \in \Psi_r\}$. For a given class comparison statistic $\mathcal{T}_i$, a problem is order-*k *delineable if there exists a subset $\Psi^{'} \subseteq \Psi_{\{o(r)\leq k\}}$ and at least one member of $\Psi^{'}$ has an order equal to k, such that its every member, r_i satisfies the delineation constraint with memory size M_i and the size of the intersection set,*

$$\mathcal{G} = \bigcup_{a_1,a_2,\cdots a_k} \bigcap C_{[a_1],i} C_{[a_2],i} \cdots C_{[a_k],i},$$

is bounded by a polynomial of ℓ, $\rho(\ell)$. Indices $a_1, a_2, \ldots a_k$ can take any value in between 1 and M_i.

These are the problems that can be solved by considering at most order-k equivalence relations and enumerating the intersection set $\mathcal{G}$. If a problem is order-k delineable in the chosen representation, then once the good classes are detected within each of the relations in $\Psi^{'}$, a simple intersection should give us a set of ground classes of size $\mathcal{G}$. Since by definition the cardinality of this set is bounded by a polynomial in ℓ, this set can be explicitly enumerated to find the best solution among them. Since all the members in $\Psi^{'}$ properly delineate the search space, the best solution of $\mathcal{G}$ is guaranteed to be the optimal solution. For this class of problems, the set of relations needed to solve is $S_r \subseteq \Psi^{'}$. Let us now find the bound on $\|S_r\|$. The best case scenario is when S_r contains ℓ/k non-overlapping relations. Since the problem is order-k delineable, there is at least one relation in S_r that has an order equal to k. In the worst case, there are only one order k relation and $\ell - k$ order one relation in S_r. Therefore, $\ell/k \leq \|S_r\| \leq \ell - k + 1$. The maximum possible index value of any relation in S_r is $\|\Lambda\|^k$. Although S_r may need at most $\ell - k + 1$ number of relations, we do not need separate sample

sets for each of them. The structure of the relation space can be exploited to gain computational benefits. The order-k relation need a sample set for explicit evaluation. The same sample set can be used to evaluate other order one relations. This is basically what we discussed earlier as benefits of implicit parallelism. While solving this class of problem, the set S_r is not explicitly given. The search for S_r from the set $\Psi_{\{o(r)\leq k\}}$ requires evaluation of all relations of order less than k. Again the benefits of implicit parallelism can be exploited for this purpose. Therefore, the k relations along any one particular path from the top node of figure 3.1 down to order-k node need explicit evaluations. Relations along the other paths from the top can be evaluated using the same sample set.

To achieve an overall success probability of q, the computational complexity,

$$SC \leq \|\Lambda\|^k k \frac{\log(1-\left(\frac{q}{q_r}\right)^{\frac{1}{(\ell-k+1)(\|\Lambda\|^k - M_{\min})}})}{-d^*} + \rho(\ell). \tag{3.3}$$

When $q/q_r << 1$, this can be approximated as before,

$$SC \leq \|\Lambda\|^k k \frac{\left(\frac{q}{q_r}\right)^{\frac{1}{(\ell-k+1)(\|\Lambda\|^k - M_{\min})}}}{d^*} + \rho(\ell)$$

This says that for fixed k, the class of order-k delineable problems can be solved with complexity growing polynomially in q, $1/d^*$, and ℓ.

Since by definition $S_r \subseteq \Psi_{\{o(r)\leq k\}}$, the search for appropriate relations is also restricted to $\Psi_{\{o(r)\leq k\}}$. The cardinality of $\Psi_{\{o(r)\leq k\}}$ is

$$\begin{aligned} \|\Psi_{\{o(r)\leq k\}}\| &= \sum_{j=1}^{j=k} \binom{\ell}{j} \\ &< k\binom{\ell}{k} \\ &< k\left(\frac{e\ell}{k}\right)^k \\ &< \frac{(e\ell)^k}{k^{k-1}}. \end{aligned} \tag{3.4}$$

Inequalities 2.19 and 3.4 can be used to find the computational complexity in the relation space for this class of problems,

$$n_r > \frac{\log(1 - q_r^{k^{k-1}/((\ell-k+1)(e\ell)^k(1-\Omega))})}{-d_r^*}. \tag{3.5}$$

This clearly shows that the computational complexity in the relation space is polynomial in q_r, $1/d_r$. We shall also clearly demonstrate that it is polynomial in ℓ too. Note that $\frac{-x}{1-x} \leq \log(1-x)$. Therefore,

$$\log(1 - q_r^{k^{k-1}/((\ell-k+1)(e\ell)^k(1-\Omega))}) \geq \frac{-q_r^{k^{k-1}/((\ell-k+1)(e\ell)^k(1-\Omega))}}{1 - q_r^{k^{k-1}/((\ell-k+1)(e\ell)^k(1-\Omega))}}.$$

For constant k and Ω, this can be written as

$$\log(1 - q_r^{k^{k-1}/((\ell-k+1)(e\ell)^k(1-\Omega))}) \geq \frac{-q_r^{b/\ell^v}}{1 - q_r^{b/\ell^v}}$$

where b and v are appropriate constants. Consider the ratio of this bound of the numerator of inequality 3.5 and an exponential growth function a^ℓ, where a is a constant, and take the limit as $\ell \to \infty$.

$$\lim_{\ell \to \infty} \frac{-q_r^{b/\ell^v}}{a^\ell} \frac{1}{1 - q_r^{b/\ell^v}} \quad = \quad 0$$

This proves that the the expression in Inequality 3.5 is also polynomial in ℓ.

The following section considers the user's role in solving a BBO. I consider two primary roles of the user regarding the source of relations and the interactions among the relations and the search operators.

3.6 Defining The Relation Space

In the SEARCH framework an algorithm searches for appropriate relations. However, this requires first defining the relation space, in other words, the set of relations Ψ_r. Defining this source of relations is one of the primary responsibilities of the user of a blackbox optimization algorithm. Doing so requires some understanding about what it means to be an appropriate

source of relations. The objective of this section is to discuss this, using the concepts developed earlier in this thesis.

In section 3.6.1 I note at least three possible ways to define relations over the search domain. Section 3.6.2 considers the issue regarding proper delineation. This is followed by Section 3.6.3 which discusses the utility of the structure of the set of relations.

3.6.1 Defining relations

As we noted repeatedly in the previous chapter, relations can be defined in many different ways. Some of the possible ways could be

1. representation,
2. perturbation operators, and
3. search heuristics.

I discuss each of them briefly in the following part of this section.

Representation can be directly used to define a similarity measure, and that can be exploited to induce classes over the search domain. I already gave examples of such similarity measures for the sequence representation. Let us consider another example using a different kind of representation—disjunctive normal form (DNF). This kind of representation is sometimes used in machine learning. For the sake of simplicity, consider binary variables $x_1, x_2, \ldots x_n$. A k-DNF representation is defined as follows:

$$(x_1 \wedge x_2 \cdots \wedge x_k) \vee (x_{k+1} \wedge x_{k+2} \cdots \wedge x_{k+k}) \vee \cdots \vee (x_{n-k} \wedge x_{n-k+1} \cdots \wedge x_n)$$

Relations and classes can be defined using this representation in a way exactly similar to what we did in case of sequence representation. Replace some variables with the fixed position corresponding to which equivalence is to be defined and substitute the wild character for rest of the variables. For example, in 3-DNF representation, $(f \wedge f \wedge f) \vee (\# \wedge \# \wedge \#) \vee \cdots \vee (\# \wedge \# \wedge \#)$ defines a relation. The classes within this relation can also be defined in a similar way, described in the context of sequence representation.

Perturbation operators can also be used to define neighborhood and classes. In this approach, a state, x and an operator define a set of states that can be reached from state x by

applying the operator to it. This defines a class of states. Therefore, an operator can be used to divide the search domain into a set of overlapping classes. Many search algorithms use this approach. Graph search algorithms like depth-first and breadth-first search (Rich & Knight, 1991) are some examples that can be viewed from this angle. Simulated annealing can also be viewed from this perspective. Even GAs can also be viewed from this angle (Jones, 1995). Unfortunately, most of these algorithms consider very restricted sets of of fixed operators, in practice it is often like—one algorithm, one operator. This may severaly constrain the relation space of the algorithm.

Search heuristics are another possible way to go. The basic idea is to use some a priori chosen rules for assigning a priority in the order different regions of the search space is explored. The use of a heuristic rule can be illustrated using a simple example. Let $h(x)$ be a heuristic rule that computes the set of states that satisfy the heuristic criterion, given that the current state is x. A heuristic-based algorithm computes $h(x)$ and decides which state to visit next, based on heuristic decision. Clearly, such heuristics divide the search space into a set of classes based on heuristic preferences. This approach is quite popular in the Artificial Intelligence community, where sometimes it is taken to an extreme end. The need for finding appropriate relations is often replaced by assertions of relations using domain knowledge. Such extensive domain knowledge is unlikely to be available in blackbox optimization problems. The field of combinatorial optimization also makes use of heuristics. For example many heuristics have been suggested for solving the traveling salesperson problem; the k-opt algorithm is one such example. A^* (Hart, Nilson, & Raphael, 1968) algorithm is another example from the body of graph search algorithms. Just like the previous approach, heuristic based relations must satisfy the delineation requirement. Therefore, use of a set of heuristics is more appropriate and search for appropriate heuristics is essential.

This section described three possible ways to define relations. The following section considers what makes a set of relations appropriate.

3.6.2 Proper delineation

Proper *delineation* of the search space is the most important requirement of a relation. Regardless of how the relations are defined, they need to satisfy this fundamental requirement. The greater the number of relations in Ψ_r that satisfy this requirement, the better it is. I

Table 3.1: A trap (f_d) and one-max (f_e) function in 3-bit representation.

x	f_d	f_e
111	3	3
110	0	2
101	0	2
011	0	2
100	1	1
001	1	1
010	1	1
000	2	0

have explained these concepts in the previous chapter. In this section, I restrict myself only to illustrate the use of the delineation-ratio for quantifying the appropriatness of a source of relations using a simple example.

Searching for relations that properly delineate the search space may not be that useful, in the case that most of the relations in Ψ_r defined by the chosen representation do not satisfy this requirement. Earlier in this chapter, I defined the delineation-ratio as the ratio of the number of relations in Ψ_r that properly delineates to the total number of relaions. For a given class comparison statistic, memory size, and a BBO, the delineation-ratio reflects how easy the search for better relation will be. Let us illustrate this observation by using the following example. Consider the two functions defined in Table 3.1. Function f_d is a trap function (Ackley, 1987) in a deceptive representation (Goldberg, 1987), and f_e is an one-max function. The one-max function is known to be very easy to solve; on the other hand, the trap function offers a great degree of difficulty to many algorithms beacuse of its isolation of the optimal solution in the hamming space. Let us again consider class average as the comparison statistic. The delineation-ratio of f_e and f_d are 1 and 1/8, respectively. Although this is for a particular class comparison statistic, the qualitative distinction is likely to be invariant for other reasonable statistics.

If the delineation-ratio is very small, then the representation is inappropriate, and we may be better off choosing a new representation. One possible way is to adaptively construct a new representation that redefines Ψ_r.

The inequality bounding the sample complexity in SEARCH clearly tells us that the index of a relation contributes to the sample complexity. The higher the value of the index, the larger is the required number of function evaluations. Therefore, it is computationally advantageous if the relations with low index, or low order, satisfy the delineation requirement.

The following section reminds us that implicit parallelism should be given consideration while choosing the set of relations.

3.6.3 Ordering the set of relations

Earlier in this chapter we noted one interesting possibility—implicit parallelism—that exploits the stucture of Ψ_r when organized based on the *order* (log of the index) of the relations. This opens up the possibility of evaluating relations in parallel at no additional sample evaluation cost. This is another factor that should influence the choice of the set of relations. Computational benefits of implicit parallelism increase as the lattice formed by ordering Ψ_r based on the *order* becomes more bushy. This is a property of the chosen set of relations and it therefore deserves due consideration.

The following section considers the role of operator-representation interaction in SEARCH.

3.7 Search Operators And Representation

Although in some BBO algorithms, search operators may be used to define the set of relations, generating new samples from the search space for evaluating classes is one of their main purposes. Since SEARCH treats relation and sample space as two seperate entities, in this section I shall focus only on the sample generation aspect.

In SEARCH samples are always generated for some particular set of classes. A random perturbation operator does not satisfy this requirement. This is discussed in Section 3.7.1. Once the ordering among the classes is constructed, and once seemingly bad classes are rejected with acceptable level of confidence, there is no reason to generate samples for them again. Therefore, the perturbation operators need to follow the decisions made earlier. This is discussed in Section 3.7.2.

3.7.1 Sample generation for equivalence classes

As I just said, the primary objective of a search operator is to generate new samples. Why does a search algorithm need new samples? The perspective of the search, as an ordering process among equivalence classes, says that we need new samples in order to compute the class comparison statistic. Therefore, an algorithm does not just need *any* sample, that a sample from some particular class. To accomplish this, the perturbation operators should be told about the class for which the sample is needed. Clearly, a random perturbation operator falls short of requirements.

Consider an example. To evaluate the class 11##, only those samples that have two consecutive 1s in first two bit positions are needed. Similarly for evaluating 1##1, we need samples with 1s in the first and last bit positions. Clearly the search operators need to know what class is currently under evaluation. It should accordingly change its region of perturbation. In other words, the search operators need to be adaptive to the direction of the search process. A little more thought should make it clear that what we are really talking about is adaptive operator-representation interaction. Sample generation for specific classes can also be accomplished with a non-adaptive search operator when representation itself is adaptive. A simple example may clear up the rationale behind this argument. Consider a 4-bit binary representation and a perturbation operator which generates new samples by randomly changing the third and fourth bit values. Such an operator could never generate samples from class ##11. Now consider a 4 position representation in which every position has a data structure, which in turn comprises a tuple $(locus, value)$. The data *locus* tells the position where *value* should be placed in a sequence. For example, $[(3,0)(0,1)(2,0)(1,1)]$ can be interpreted to the binary string 1100. Now let us reconsider the same search operator that generates samples by perturbing only the third and fourth positions. In the new representation scheme, the sample generation of class ##11 is quite straightforward, since *locus* is explicitly tagged with a value. The class ##11 can be represented as $[(3,1)(2,1)\#\#]$ or $[(2,1)(3,1)\#\#]$. This simple example illustrates that non-adaptive search operators can also serve the purpose, provided that the representation itself is adaptive. This section noted that the representation-operator interaction needs to be adaptive to ensure proper sample generation. The following section points out that the perturbation

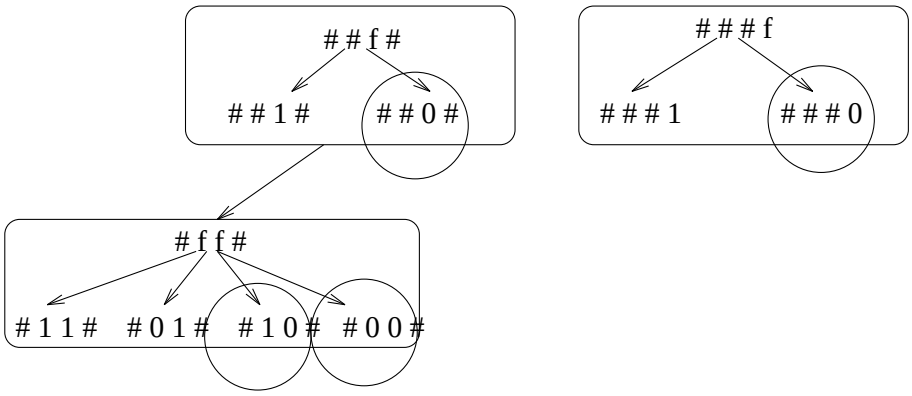


Figure 3.2: Effect of low-order equivalence class pruning on future exploration. The classes marked by circles need not be considered.

operators need to be controlled to ensure that they do not generate samples for the classes that are already discarded.

3.7.2 Controlling the perturbation

Better control of operator perturbation is also needed for another reason. Consider the subgraph shown in Figure 3.2. Let us say that the class ##0# is evaluated to be worse than other order one classes of relation ##*f*# and that it is eliminated from further consideration; also assume that the class ###0 is rejected. This essentially means that all higher-order equivalence classes such as #10#, and #00#, which are subsumed by the class ##0#, can also be pruned out. Moreover, since ###0 is already pruned out, the samples generated for evaluating classes #11# and #01# should be restricted from being a member of ###0. The search operators need to keep track of the equivalence classes that are eliminated and accordingly adapt themselves to generate appropriate samples. Again, a simple randomized perturbation operator does not satisfy this requirement. Radcliffe (1993) discussed this desirable feature of the perturbation operator in details.

The following section summarizes the major points of this chapter.

3.8 Summary

This chapter laid out the basic principles for understanding the three major entities in BBO,

- search algorithm
- problem
- user's role in defining the set of relations

The main points of this chapter are recapitulated in the following.

First we defined what it means to be an algorithm in SEARCH. Different components that are required for solving a BBO problem in SEARCH are listed together and discussed. I hope this will provide a common ground for developing new blackbox optimization algorithms in the future. Before designing any new algorithm, we should ask several questions: how does this algorithm process relations? How does it treat the classes? Does the operator for sample generation follow the decision making in the relation and class spaces? What are the relation and class comparison statistics? How does it take advantage of implicit parallelism?

Next, I discussed two important issues that may help make a BBO algorithm computationally more efficient. It is also noted that the index of the relation considered in the beginning of the search process should be bounded by a constant. The smaller the constant, the better it is. This requirement is called *bottom-up* organization of search, which may have some biological confirmation. It is noted that parallel evaluation of different relations at no additional function evaluation is possible by exploiting the structural property of the set of relations, when ordered based on the *order* of relations. I identified this as implicit parallelism (Holland, 1975).

This chapter also rigorously defined problem difficulty in SEARCH. The notion of problem difficulty introduced here is philosophically very similar to the PAC-learning theory (Natarajan, 91). The SEARCH framework decouples problem difficulty along these dimensions:

1. problem size
2. success probability in making the right decision to choose a relation that properly delineates the search space
3. success probability in selecting the class containing the optimal solution
4. desired quality of relations and the solution

All of the above dimensions of difficulty, except problem size, came as a part of the development of SEARCH. Identifying the exact quantitative dependence upon the problem size requires

relating the size of the set of relations with problem size. I therefore considered a sequence representation as the source of relations and introduced the class of order-k delineable problems that can be solved in polynomial sample complexity. This class of problems will be further explored in the coming chapters.

The primary roles of the user of a BBO algorithm are to define the set of relations and to choose the perturbation operators based on the chosen set of relations. I first note three of the many possible ways to define relations over the search domain. The appropriateness of a set of relations is quantified in terms of the delineation-ratio. Computational benefits can be achieved by choosing a set of relations in which low-order relations satisfy the delineation requirement and also parallel evaluation of relations is possible by exploiting the structure of Ψ_r when organized based on the *order* of the relations. All these factors contribute to the efficacy of the set of relations. At the end, the role of perturbation operators is viewed from the relation construction perspective. Perturbation operators need to generate samples for those classes that are currently being evaluated. Therefore, search operators clearly need to know about the classes for which they are going to generate the samples.

The SEARCH framework and its implications offered an alternate perspective of different components of BBO. One of the main lessons of this framework is that the search for appropriate relations is essential in BBO. In the following chapter I consider messy genetic algorithms, one among the rare class of algorithms that searches for appropriate relations.

Chapter 4

Design of the Fast Messy Genetic Algorithm

The previous chapters introduced SEARCH—an alternate perspective toward BBO based on relations and classes—and its implications toward BBO problem solving. SEARCH emphasizes the need for searching for appropriate relations, since considering relations is essential to surpass the limits of enumerative search. Unfortunately, not many blackbox optimization algorithms realize this. In this chapter, I consider a rare class of algorithms that realize the need for relation search, known as messy genetic algorithms (Deb, 1991; Goldberg, Korb, & Deb, 1989), Historically, messy GAs came before the development of SEARCH, and therefore, messy GAs deserve the credit for the first step toward the right direction.

Messy GAs are a new generation of genetic algorithms that originated from the lessons learned from simple genetic algorithms (De Jong, 1975; Holland, 1975). Among others, poor search for relations is one of the main weaknesses of the simple GA. Section 4.1 discusses the strengths and weaknesses of simple GAs. Section 4.2 presents a brief review of the original version of the messy GA (Deb, 1991; Goldberg, Korb, & Deb, 1989). Although the original messy GA realized the role of the search for relations, one of the major problems of the original messy GA was its inability to take advantage of implicit parallelism. The fast messy GA (Goldberg, Deb, Kargupta, & Harik, 1993) was introduced to eliminate this bottleneck. The fast messy GA (fmGA) replaces the explicit evaluation of equivalence classes by a partially implicit evaluation that makes it computationally more efficient. Section 4.3 presents the design

of the fast messy GA. The choice of thresholding parameter during the building-block[1] filtering process plays a crucial role in the success of fmGA. The approach suggested by Goldberg, Deb, Kargupta, and Harik (1993) for designing a filtering schedule may create some problems in maintaining even growth of building-blocks during the late stages of the filtering process. A modified scheme for thresholding selection and iterative applications are added to improve its performance. Section 4.4 describes them. A detailed description of the new thresholding scheme is presented in Appendix C. The overall organization of the modified fmGA is described in Section 4.5. Finally, Section 4.6 discusses the main conceptual strengths of the fmGA and points out directions for further improvements.

4.1 Lessons from the Simple GA

Genetic algorithms (GAs) (Holland, 1975) are stochastic search procedures based on simplified mechanics of natural genetics. Although the actual algorithms within this family vary in different aspects, the simple genetic algorithm (sGA) (De Jong, 1975; Holland, 1975) captures the essence of the GA proposed by Holland. A brief description of simple GA can be found in Appendix B. Although the algorithmic implementation of the sGA is very simple, it is quite interesting to note that the fundamental motivations (Holland, 1975) behind designing the sGA share some common grounds with SEARCH. The first part of this section addresses some strengths of the sGA and the latter half points out the major bottlenecks.

The simple GA has many interesting features. The main strengths of the simple GA can be listed as follows:

1. It is possible to define a richer source of relations through representation.
2. Implicit parallel evaluations of relations can be attained.
3. Crossover does not change the decision made by selection for order-1 relations and it can be made more faithful to the decision of selection for higher order relations (Radcliffe, 1991).

Each of these points will be briefly discussed in the following.

[1] A building-block is an instance of a class within the top M_i classes of a relation r_i that properly delineates the search space.

The first important aspect is that GA emphasizes the role of representation in search that can be used to define a rich source of relations. From the very beginning, the design of GAs has been primarily viewed in the light of the schema or equivalence class processing within the partitions or relations defined by the chosen representation. Although the simple GA does not explicitly consider the relations and classes, the fundamental motivation shares the spirit of SEARCH.

Probably one of the most interesting observations that Holland made in his book (1975) is the idea of implicit parallelism. He pointed out that by processing a population of size n, the GA gathers information about $O(n^3)$ schemata. Although this argument was not rigorously laid in terms of computational arguments, the SEARCH framework identifies and confirms the concept in a quantitative manner. As I noted in the previous chapters, as long as the poset $\gamma(\Psi_r)_{<_o}$ is not linearly ordered, the parallel structure can be exploited to evaluate different relations simultaneously from the same set of samples. The computational benefits from the parallel evaluations of relations have been identified as the benefits of implicit parallelism.

The crossover operator in GA offers some unique features. Unlike the random mutative perturbation operators, crossover has some degree of capability to control the perturbation. If an order-1 class is eliminated from the population by selection, crossover never regenerates that class. Since crossover just swaps values at different loci, it cannot alter the overall distribution of the different values at any locus. In other words, crossover respects the decision making at the order-1 level by the selection operator. Unfortunately, this not true for higher-order relations. Radcliffe (1993) discussed related issues in details and suggested principles for designing more "respectful" crossover operators. Apart from this, we also noted that SEARCH requires resolution of relations that becomes computationally useful for problems that are delineable at a certain level. Crossover can also be considered as an operator that generates samples from an intersection set of different equivalence classes.

Despite these interesting features, the simple GA has several major problems. The main bottlenecks of the simple GA are as follows:

- Relation, class, and sample spaces are combined.
- A lack of precise mechanism for implicit parallelism occurs.
- Only a poor search for relations can occur.

Each of these points will be discussed in the following.

Since the relation, class, and sample spaces are combined, decision making in each of these spaces affect the other two. Only one common measure defined by selection is used to decide in both the relation and class spaces. Although the roles of each of these spaces in BBO are related, the decision makings in each of them are distinctly different. As a result, the overall decision making process becomes noisy and susceptible to error. The sGA does not have any way to explicitly decide about relations or classes without affecting the other spaces.

The simple GA considers only a small fraction of relations defined by the representation. A simple GA with one-point crossover (De Jong, 1975) favors those relations in which positions in sequence space defining equivalence are closer to each other and neglects those relations that contain equivalence defining positions far apart. One-point crossover also fails to generate samples for the intersection set of two equivalence classes in which fixed bits are widely separated. For example, in a 20-bit problem, single-point crossover is very unlikely to generate a sample from the intersection set of $1\#\#\cdots\#$(first bit is fixed) and $\#\cdots\#1$ (last bit is fixed). In biological jargon, this is called the *linkage problem*. Unfortunately, this is a major bottleneck of sGA. Although Holland (1975) realized the importance of solving this problem and suggested use of the *inversion* operator (Holland, 1975), it has been shown elsewhere (Goldberg & Lingle, 1985) that inversion is very slow and unlikely to solve this problem efficiently. One-point crossover is not the only type to suffer from this problem. Uniform crossover is another kind of crossover (Syswerda, 1989) often used in the simple GA. In uniform crossover, the exchange of bit values among the two parent strings takes place based on a randomly generated binary mask string. If the value of this mask string at a particular locus is 1, the corresponding bits in the parent strings get exchanged; otherwise they do not. Unlike one-point crossover, uniform crossover does not have any preference bias toward the closely spaced partitions. Since the relation space and the sample space are combined, random perturbation of the sample strings also result in disrupting proper evaluations of the relations. Uniform crossover should also fail to accomplish proper search in the relation space. In fact, this is exactly what Thierens and Goldberg (1993) reported. Their analysis and experimental results showed that the sample complexity grows exponentially with the problem size for solving bounded deceptive problems (Thierens & Goldberg, 1993) using a simple GA with uniform crossover.

This discussion clearly points out that the search in the relation space is very poor in the case of a simple GA with either one-point or uniform crossover.

In the following section I discuss the messy genetic algorithm (Deb, 1991; Goldberg, Korb, & Deb, 1989) that made serious efforts to eliminate one of the major bottlenecks of the simple GA—poor search for relations.

4.2 Messy Genetic Algorithms: A Review

Messy GAs (mGAs) are a class of iterative optimization algorithms that make use of a local search template, adaptive representation, and a decision theoretic sampling strategy. The work on messy genetic algorithms (mGAs) (Deb, 1991; Goldberg, Korb, & Deb, 1989; Goldberg & Kerzic, 1990; Goldberg, Deb, & Korb, 1990b) was initiated to eliminate some major problems of the simple GA, as described in the previous section. The main strong points of mGAs are listed in the following:

- The mGA eliminates the undesirable bias of sGA toward those relations that have all of their equivalence defining positions close together.
- It adaptively modifies the representation-operator interaction;
- It precisely evaluates the equivalence classes.

The search for good relations is highly restricted in the sGA. The mGA addresses this bottleneck of the sGA. The messy GA uses a relaxed, variable locus representation to solve the linkage problem. Precise evaluation of equivalence classes is also emphasized in the mGA. To keep things simple and well-grounded, the original mGA adopted an explicit enumeration technique for evaluating the equivalence classes. This, of course, sacrificed all the benefits of implicit parallelism. Construction of partial ordering among the equivalence classes is, again, an important issue that the mGA took quite seriously.

This section reviews different aspects of the original version of messy GA (Deb, 1991). Subsection 4.2.1 discusses the representation in mGA. Equivalence classes are explicitly enumerated and evaluated in the context of a template. Subsection 4.2.1 reviews this. Subsection 4.2.2 discusses the messy GA operators. Thereafter, Subsection 4.2.3 describes the overall organization of the mGA. Finally, Subsection 4.2.4 discusses some strong and weak points of the mGA.

4.2.1 Representation

The messy GAs relax the definition of representation used in the simple GA. Unlike the simple GA, mGAs define the chromosome, i.e., the sequence-represented members of the population as a collection of position-independent genes. A single gene is an ordered pair, (*locus, value*); in a ℓ-loci representation, the *locus* can be any value between zero and $\ell - 1$, and *value* is any letter from the chosen alphabet. The *locus* identifies the actual position of the gene. For example, consider a problem of length $\ell = 3$. A sample messy string for this problem could be ((0 1)(2 0)(1 1)). This corresponds to the string 110 in a fixed-locus representation of the simple GA. Although mGAs allow redundant and missing genes, the representation in mGAs is richer than that of simple sequence representation even without considering them. Considering only the final encoded string ready for evaluation, representation in mGA can be formally denoted as

$$I_m : \mathcal{X} \rightarrow S_\ell \times \Lambda^\ell,$$

where S_ℓ is the set of all different permutations of integers zero through $\ell - 1$. Note that the same member of $\mathcal{X}$ can be represented in different ways in this representation. To be specific, every member of $\mathcal{X}$ has $\ell!$ distinct representations. This gives the mGAs an additional advantage in searching for proper equivalence relations. Since the locus of a gene is explicitly specified, related genes can be adaptively clustered during the search to minimize the effect of operator disruption.

Another interesting aspect of the mGA is that it explicitly represents equivalence classes. Every member of the mGA population represents an equivalence class of certain order. Unlike the GA, where a population is comprised of only order-ℓ classes, i.e., the ground members, mGA strings can be of any arbitrary non-zero order. The messy GA strings can be underspecified or overspecified. For example, both ((0 1)(2 0)(1 1)(2 1)) and ((0 1)(2 1)) are valid messy strings for a problem of length $\ell = 3$. The first string is overspecified, whereas the second is underspecified. Overspecified strings are mapped to Λ^ℓ by left-to-right scanning of the strings on the basis of first-come first-served preference. On the other hand, an underspecified string of length k defines an equivalence class of order-k.

Equivalence classes are explicitly represented by underspecified strings. Underspecification is handled using a local search template string. A local search template is a locally optimal string that remains unchanged during a particular iteration of the messy GA. Evaluations of equivalence classes are performed in the context of this search template. The template has all of its genes specified. The missing genes of an underspecified string are filled in by the template. In our example problem, a template may resemble ((0 1)(1 0)(2 0)). The incompletely specified string ((0 1)(2 1)) can be evaluated in the context of this template. The missing gene is filled by the template, resulting in string ((0 1)(1 0)(2 1)).

The following section reviews the mGA operators.

4.2.2 Messy operators

The messy GA uses two main operators: (1) *thresholding selection* and (2) *the cut and splice* operator. Thresholding selection constructs the ordering between the equivalence classes and gives more copies to the underspecified strings representing the better classes. On the other hand, the cut and splice operators are used to construct the intersection among the equivalence classes. In the following discussion, I briefly describe the working of these two operators.

4.2.2.1 Thresholding selection

Selection generates more copies of the strings with higher objective function values. In a tournament selection (Brindle, 1981; Goldberg, Korb, & Deb, 1989), construction of the ordering among the equivalence classes is accomplished by pairwise comparison of class members. Thresholding selection tries to ensure that only classes belonging to a particular equivalence relation are compared with one another. Consider the strings ((1 0)(0 0)), ((1 1)(0 1)), and ((1 0)(2 1)). The first two strings define equivalence classes 00#, 11# over the relation $ff\#$. On the other hand, the last string defines the class #01 over the relation $\#ff$. Clearly, comparing the first two makes sense, because they are from the same relation; on the other hand, the last string must be restricted from competing with the other two strings, since it belongs to a different relation. Thresholding selection tries to accomplish exactly this, aside from the duty of selecting better strings. A similarity measure θ is used to denote the number of common genes among two strings. Two strings are allowed to compete with each other only if their θ is greater than some threshold value $\overline{\theta}$. The messy GA (Deb, 1991) used $\overline{\theta} = \ell_1\ell_2/\ell$, where ℓ_1, ℓ_2,

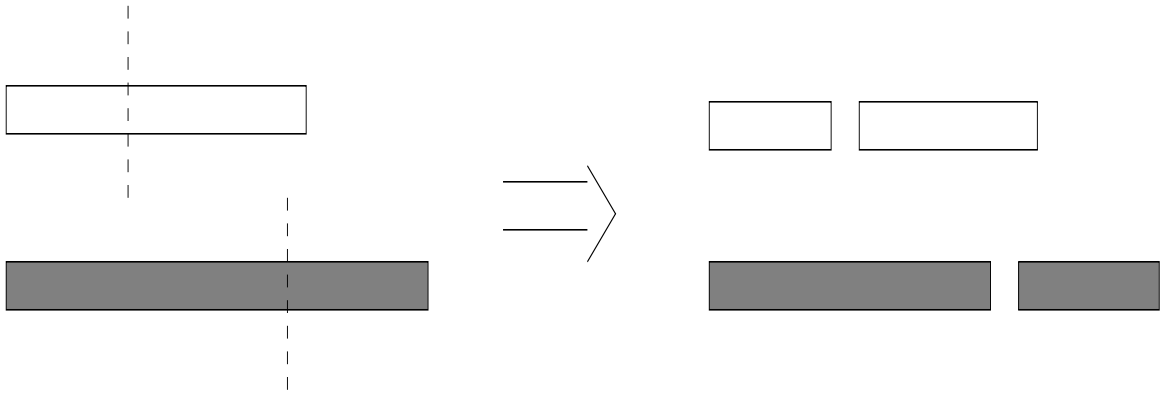


Figure 4.1: Cut operation.

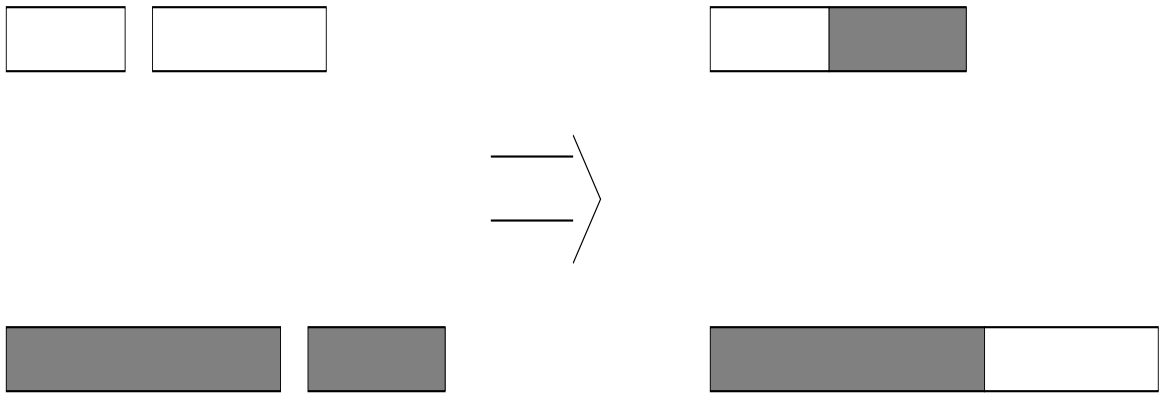


Figure 4.2: Splice operation.

and ℓ are the length of the two strings and the problem length, respectively. This expression is the expected value of θ when the two strings are randomly picked from a randomly initialized population (Deb, 1991). Thersholding selection basically implements a relaxed version of the class comparison process within a particular relation.

The following subsection reviews the cut and splice operators in the messy GA.

4.2.2.2 Cut and Splice operation

The cut and splice operation simulates the behavior of crossover for different length strings. Consider the strings ((1 1)(0 1)(2 0)) and ((2 1)(1 0)(0 1)(2 1)). The cut operation randomly picks two points, one for each string. Let us say that it picks 2 and 3 for the first and second string, respectively. The cut operation then splits the first string into ((1 1)(0 1)) and ((2 0)).

The second string is also divided into the strings ((2 1)(1 0)(0 1)) and ((2 1)). The splice operation swaps the split parts and generates new strings. In the current example, splice operation generates the strings ((1 1)(0 1)(2 1)) and ((2 1)(1 0)(0 1)(2 0)). Figures 4.1 and 4.2 graphically depict the operation of cut and splice operators, respectively.

The following subsection describes the overall organization of the messy GA.

4.2.3 Organization of the messy GA

The messy GA works by iterating within two loops—the outer and inner loops. The variable of the outer loop is the order of the equivalence relations considered. The inner loop constructs the ordering among the equivalence classes of different relations of the same order and produces a locally optimal solution by taking the intersection of the good equivalence classes using the cut and splice operations.

Table 4.3 presents pseudo-code explaining the operation of the mGA. In the following discussion I explain the overall working procedure of the mGA.

- **Outer loop begins:** The outer loop iterates over the order of equivalence relation k. At the initial iteration of the outer loop, k may be chosen as 1 if no other prior information is available. Initially, the template is randomly generated.

- **Inner loop:**

  1. **Initialization:** The population is randomly initialized with each string of length k, where k is the order of the currently considered equivalence relations. Precisely speaking, the initialization of the messy population is not exactly random. Rather, it makes sure that all $2^k \binom{\ell}{k}$ different possible order-k equivalence classes are present in the initial population. The objective function value of all these strings are evaluated in the context of the template.

  2. **Primordial phase:** The primordial phase constructs the ordering among the equivalence classes by applying thresholding selection alone. A binary tournament selection operator (Brindle, 1981; Goldberg, Korb, & Deb, 1989) gives on average two copies of the best string. Often, this results in continuing the primordial phase for several generations. No function evaluation is performed during this phase.

```
/* Initialization */
level = 1; template = random_string; // Random initialization of the template
While(mga_termination_criterion == TRUE) )
{
  t = 0; // Set the generation number to zero.
  Initialize(Pop(t), level); // Initialize the population at random
  Evaluate(Pop(t), template); // Evaluate the fitness values
  Repeat // Enter primordial phase
  {
    Selection(Pop(t)); // Select better strings
    Evaluate(Pop(t)); // Evaluate fitness
    t = t + 1; // Increment generation counter
  }
  Until (primordial_termination_criterion == TRUE)

  t = 0;
  Repeat // Enter juxtapositional phase
  {
    Selection(Pop(t));     Cut_and_Splice(Pop(t)); // Apply cut and splice operator
    Mutation(Pop(t)); // Apply mutation
    Evaluate(Pop(t), template);
    t = t + 1;
  }
  Until ( juxtapositional_termination_criterion == TRUE)

  template = optimal_string(Pop(t), level); // template remains locally optimal
  level = level + 1;
}
```

Figure 4.3: A pseudo-code for original version of the messy GA.

3. **Juxtapositional phase:** During this phase both thresholding selection and cut-&-splice operators are used. Good strings are picked, cut, and then spliced to generate better strings. Since new strings are generated during this phase, evaluation of the objective function values is required in every generation.

4. **Inner loop ends:** The template is set to the best solution found at the end of the juxtapositional stage of the previous iteration of the inner loop.

- **Outer loop ends.** The algorithm stops when the order of relations considered exceeds a certain value or some other stopping criterion is satisfied.

The following section summarizes the strengths and weaknesses of the mGA.

4.2.4 Lessons from the mGA

The messy GA took the first step in the right direction. It realized the need for searching appropriate relations. Explicit enumeration of equivalence classes and selection in the presence of thresholding aid precise evaluation and ordering of classes. This made the messy GA applicable to a larger class of problems, compared to sGA. The messy GA has been applied to different kinds of problems, including several real-life applications (Deb, 1991; Dymek, 1992; Merkle & Lemont, 1993). The main strong aspects of the messy GA are listed in the following:

1. It decomposes of the search space into two different spaces—(1) the sample space and (2) the class and relation space.

2. Less noisy decision making occurs compared to the simple GA because of this decomposition.

3. A better search for relations occurs.

Each of these points will be briefly discussed here.

The messy GA considers two distinctly different spaces—the population of strings and the template space. During the primordial stage every string in the string population has a length less than the problem length. Each of them defines an equivalence class. Therefore, during the primordial stage the string population represents the class space. On the other hand, the template is always a full string, and this defines the sample space in the mGA. During the

juxtapositional stage, however, the strings in the population become larger and gradually start becoming full strings. The string population can still be viewed as the class space, precisely the space of classes for order-ℓ relations, where ℓ is the problem length.

Since the main decision making in the mGA is made during the primordial stage, decomposition of the search into class and sample spaces makes the decision making less noisy compared to that of the sGA. The mGA uses an explicit enumerative initialization of the population. This makes the decision making more accurate.

One of the most important aspects of the mGA is that it realizes the importance for finding appropriate relations. Unlike the sGA, the search for proper relation in the mGA is more accurate and less susceptible to error because of the explicit enumeration and the use of a locally optimal template for class evaluations.

Despite these interesting features, the original mGA has some problems:

1. decomposition of the search space
2. single local search template
3. lack of implicit parallelism

I discuss each of them in the following.

The mGA takes one step ahead by decomposing the sample space from the relation and class spaces. However, further decomposition may be necessary. The rationale behind this observation originates from the role of thresholding selection in mGA. Thresholding selection tries to find the better classes and also the better relations. The thresholding parameter is always chosen less than the string length for allowing the elimination of bad relations. However, since the relations and classes are implicitly defined in the string population, the construction of ordering among relations influences that among the classes. For example, when thresholding parameter $\theta = 2$ for a 4-bit problem, strings like ((0 1)(2 1)(1 1)) and ((0 1)(2 1)(3 0)) compete with each other. Although the idea is to eliminate bad relations, physically this means comparing classes from different relations, which is inappropriate. The undesirable consequence of this is that it also allows competition among classes from good relations. This needs further study and experimentation.

By definition, the class comparison statistic of any algorithm is chosen in an ad hoc manner. It is a source of algorithmic bias. It is the particular aspect of an algorithm that may make

a class of problems either easy or difficult for itself. The messy GA is no exception to that. Unlike sGA, in the messy GA, the population of strings explicitly represents the class space. The template defines the sample space. Although the template is by definition a locally optimal solution, at least in the initial stage, at the order-1 level of outer loop the template is randomly generated. It is not clear how general the idea of comparing two classes based on a single sample is.

The messy GA explicitly enumerates the $2^k \binom{\ell}{k}$ order-k equivalence classes and evaluates all of them. As we noted in Section 3, parallel evaluation of different equivalence relations can be done at the cost of evaluating one relation. The messy GA does not take advantage of this possibility at all. On the other hand, the simple GA takes advantage of this at the price of high noise during evaluation of the classes. This lack of implicit parallelism is a major problem of the messy GA. Although the sample complexity is still polynomial, for problems with even a moderate order of decomposability, the population size becomes very large for any practical purpose.

Among the above three, lack of implicit parallelism is the main computational bottleneck of the original messy GA. The following section introduces the fast messy GA that preserves the mGA's search for good relations but returns some of the benefits of implicit parallelism.

4.3 The Fast Messy Genetic Algorithm

The simple GA harnesses the power of implicit parallelism, but at a higher price of noisy evaluation of equivalence classes. The sGA assigns credit at the string level, and the evaluation of classes is completely implicit. The original messy GA went to the other extreme. It emphasized the accurate evaluation of better classes by explicitly enumerating all-possible classes within the all possible relations at some order-k and evaluating them in the context of a locally optimal solution. The fast messy GA (fmGA), initiated by Goldberg, Deb, Kargupta, and Harik (1993), takes a moderate strategy to balance both the ends.

The fmGA brings some of the benefits of implicit parallelism to the mGAs while retaining some degree of accuracy in detecting good relations. An earlier work (Goldberg, Korb, & Deb, 1989) reported discouraging results for overcoming this bottleneck. However, the fmGA overcame this problem of the mGA using the following techniques:

1. probabilistically complete initialization

2. building-block filtering in presence of thresholding selection

The probabilistically complete initialization replaces the explicit enumerative evaluation of all $\Lambda^k\binom{\ell}{k}$ classes. The modified population-sizing becomes $O(\ell)$. The basic idea is to statistically gather information about order-k equivalence classes by evaluating some order-ℓ' classes, where $k \leq \ell' \leq \ell$.

The building-block filtering in presence of thresholding selection offers a way to use the primordial stage for gradually detecting the order-k classes from strings of length ℓ'. During this phase, strings are selected in presence of thresholding and genes are occasionally randomly deleted for reducing the string length from ℓ' to k.

The objective of this section is to present a detailed description of the fmGA. Section 4.3.1 discusses the probabilistically complete initialization technique that ensures proper decision-making by selection. Section 4.3.2 describes thresholding selection. The fast messy GA gradually reduces the string length using thresholding selection and random gene deletion. Section 4.3.3 describes the string length reduction process.

4.3.1 Probabilistically complete initialization

In the original messy GA, all the order-k equivalence classes are explicitly enumerated. This initialization is deterministic. The fast messy GA replaces this with a probabilistic technique. Initial string length ℓ' and the population size are the two design factors of the initialization process.

The basic idea behind the probabilistically complete initialization is that all the $2^k\binom{\ell}{k}$ equivalence classes can be defined using a much smaller number of strings, when the string length is much higher than k. In other words, multiple combinations of classes can be defined by the same string. The number of ways a string of size $\ell' > k$ contains a gene combination of size k may be calculated by assigning k genes to the string and then choosing the total number ways $\ell' - k$ genes can be created from $\ell - k$ genes. This is simply $\binom{\ell-k}{\ell'-k}$. Note that the total number of strings of size ℓ' created with ℓ genes is $\binom{\ell}{\ell'}$. Thus if we take $n_g = \binom{\ell}{\ell'}/\binom{\ell-k}{\ell'-k}$ string samples each of length ℓ' randomly, on average there will be one string that will have the

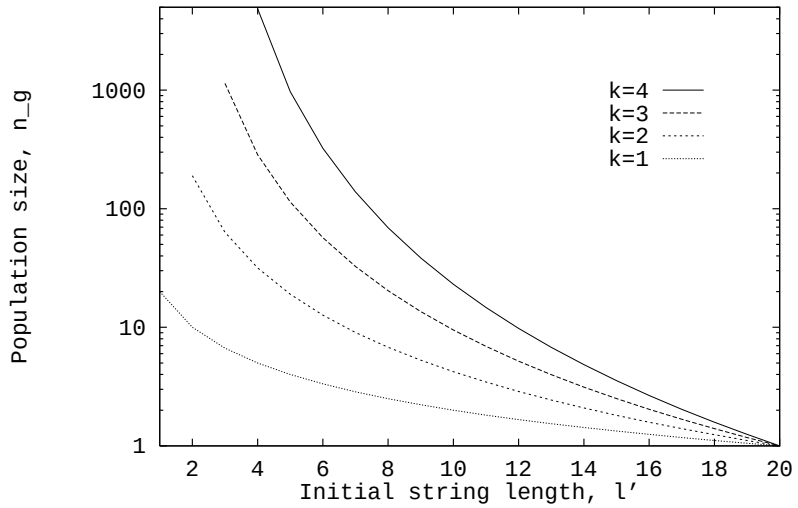


Figure 4.4: The population size required to have one expected instance of a building block of size k in strings of length ℓ' is plotted against ℓ'. The problem size $\ell = 20$ is assumed (from [Goldberg, Deb, Kargupta, Harik, 1993]).

desired gene combination of size k; n_g decreases exponentially as the string length ℓ' increases. Figure 4.4 shows the variation of n_g versus ℓ'. When $\ell' \approx \ell$, n_g is a constant.

Detecting order-k building-blocks from strings of length $\ell' >> k$ requires several sample strings that contain the building-blocks. This is essentially a decision problem. Goldberg, Deb, and Clark (1992) proposed a parametric decision theoretic approach to choose a sample size that ensures that the building-blocks are detected with high confidence. They proposed the population-sizing equation

$$n_a = 2c(\alpha)\beta^2(m-1)2^k,$$

where m is the number of subfunctions and $c(\alpha)$ is the square of the ordinate of a normal random deviate whose tail has an error probability α; $c(\alpha)$ will be denoted by c in short. β is the ratio of the variance and mean of the convolution of the distribution of the two competing classes (Goldberg, Deb, & Clark, 1992). The overall population size can be computed by multiplying

n_g and n_a:

$$\begin{aligned} n &= n_g n_a \\ &= \frac{\binom{\ell}{\ell'}}{\binom{\ell-k}{\ell'-k}} 2c_f\beta^2(m-1)2^k. \end{aligned} \tag{4.1}$$

The choice of ℓ' can be viewed as a trade-off between reduction in population size and increase in error probability. As ℓ' is increased, n_g decreases quickly, but error probability increases, since additional noise is introduced by additional bits. For all the experimental results presented in this chapter, I used $\ell' = \ell - k$.

This probabilistically complete initialization process makes sure that all the order-k classes are present in the initial population and they can be detected with high probability. Thresholding selection increases the proportion of good classes. The following section discusses the design of thresholding selection in the fmGA.

4.3.2 Thresholding selection

Thresholding selection is a selection operator that minimizes the competition among classes from different relations. In this section, I describe the design of this operator in the fmGA.

Ordinary selection gives more copies to the better string. This, however, cannot restrict competition among classes that do not belong to the same equivalence relation. Thresholding selection is a kind of selection that restricts such undesirable competition. The thresholding selection (Deb, 1991) is based on the *tournament selection* (Brindle, 1981; Goldberg, Korb, & Deb, 1989). In the binary tournament selection, population members are ordered by pairwise comparison. Two members are randomly picked from the population and their objective function values are compared with each other. The winner gets a copy in the next generation population. On average, the best member of the population gets two copies in every generation. Thresholding selection restricts the way members are picked from the population. In a binary thresholding selection, the first competitor is picked up randomly. The next competitor is the first randomly-picked string from the population that has a certain threshold, θ number of genes in common with the previous string. For example, consider a problem of length $\ell = 10$. Let us say that the first candidate is the string ((0 1)(3 1)(5 0)(2 1)). If the

chosen threshold parameter $\theta = 4$, then the string ((4 1)(3 1)(7 0)(5 1)) cannot compete with it; ((3 0)(2 0)(0 0)(5 0)) can however, since it has 4 common genes with the first competitor. When θ is equal to the string length, only the classes that belong to a particular relation can compete with each other. Ideally this should be what we want. However there is another issue—the search for appropriate relations. As we know in a sequence space of ℓ genes $\binom{\ell}{k}$ different order-k equivalence relations can be defined. The reader may recall from a previous section that the probabilistically complete initialization of the fmGA generates classes for all possible order-k combinations. In other words, the initial population contains classes from all possible order-k equivalence relations. However, for any problem, it is likely that some relations delineate the search space properly and some of them do not. The undesirable relations need to be eliminated from future considerations. The fast messy GA tries to accomplish that by allowing slight competition among the classes created by different relations. This is done by making the thresholding parameter θ less than the current string length. The idea is that this relaxation will introduce competition among classes belonging to relations which share some common genes. Such competition can eliminate the classes of bad relations. When θ is made less than the string length, the set of equivalence relations is divided into different overlapping subsets. In the previous example, if we make $\theta = 3$, then the string ((0 1)(3 1)(5 0)(2 1)) can match with classes of other relations, such as ((0 1)(3 1)(7 0)(2 0)).

The question is, How much should we relax the value of θ? Deb (1991) noted that in a randomly generated population, the probability that a randomly chosen string is also a member of the thresholding niche of a different class follows hypergeometric distribution. He computed the expected number of common genes between two randomly chosen strings of length ℓ_1 and ℓ_2 as

$$\theta = \frac{\ell_1 \ell_2}{\ell}, \tag{4.2}$$

where ℓ is the original problem length. Using θ worked fine with the original messy GA. However, in the fmGA, the string length is gradually reduced from ℓ' to order-k. As the selection is applied to the population, the number of members of a thresholding niche drastically changes. From our experiments it was clear that a choice of $\theta = \frac{\ell_1 \ell_2}{\ell}$ is too relaxed when $\ell >> \ell_1, \ell_2 >> k$. This resulted in large amount of cross-competition among the classes and elimination of some

good classes. A conservative value for θ can be chosen following the studies of Goldberg, Deb, Kargupta, and Harik (1993). They suggested using,

$$\theta = \lceil \frac{\ell_1 \ell_2}{\ell} + c'(\alpha')\sigma \rceil, \tag{4.3}$$

where σ is the standard deviation of the number of genes two randomly chosen strings of possibly differing lengths have in common, and the parameter $c'(\alpha')$ is simply the ordinate of a one-sided normal distribution with tail probability α. The variance of the hypergeometric distribution may be calculated as follows (Deb, 1991):

$$\sigma^2 = \frac{\ell_1(\ell - \ell_1)\ell_2(\ell - \ell_2)}{\ell^2(\ell - 1)}. \tag{4.4}$$

implementation of thresholding selection searching for a competitor that has at least θ genes in common with the string picked first. Strings from the population are randomly picked and checked for matching. If a match is found, the process stops, and a tournament among them takes place. Otherwise, the search for a match continues for a certain number of times, $n_s = \alpha_s n$, where α_s is a constant between 0 and 1 and n is the population size; n_s is called the shuffle number.

However, even this approach does not solve the problem completely. Addressing this issue requires a more fundamental approach toward understanding the thresholding process. Section 4.4.1 sketches an alternate approach to design the filtering schedule. However, before we discuss that, it is essential that we first understand the process of string length reduction in fmGA. This is described in the following section.

4.3.3 Building-Block filtering

During the primordial phase of fmGA, thresholding selection increases the proportion of the better strings. However, in addition to that, the string length needs to be gradually reduced to order-k. At this stage, order-k classes can be accurately evaluated in the context of the common template, just as it is done in the messy GA. Gradual reduction of string length is accomplished by random deletion of genes. This process of detecting the good classes by thresholding selection and gene deletion is called building-block filtering. In this section I

describe the design of building-block filtering process in fmGA as suggested by Goldberg, Deb, Kargupta, and Harik (1993).

Reduction of string length by random gene deletion is a straightforward concept. Let us first introduce some symbols. Consider the sequences of string lengths generated by successive applications of gene deletion by $\lambda^{(0)}, \lambda^{(1)}, \ldots \lambda^{(i)} \ldots \lambda^{(\mathcal{N})}$. The initial string length $\lambda^{(0)} = \ell'$ and the final string length $\lambda^{(\mathcal{N})}$ are chosen to be some number close to k. Selection continues for some number of generations with constant string length $\lambda^{(i)}$ to produce more copies of the better strings. Note that these are selection-only generations and therefore no new function evaluation is needed. This is followed by random deletion of genes which reduce the string length to $\lambda^{(i+1)}$. These shorter strings are then evaluated and the same process of thresholding selection and gene deletion continues until $\lambda^{(i)} = \lambda^{(\mathcal{N})}$.

The gene deletion rate should be chosen so that it is on average less than the rate at which better strings get more copies by selection. To correctly choose a building block of size k from strings of length $\lambda^{(i-1)}$ by picking $\lambda^{(i)}$ genes at random, we need a building-block repetition factor $\gamma = \binom{\lambda^{(i-1)}}{\lambda^{(i)}} / \binom{\lambda^{(i-1)}-k}{\lambda^{(i)}-k}$ strings to have one expected copy remaining of the desired building block. For large values of $\lambda^{(i-1)}$ and $\lambda^{(i)}$ compared to k, γ varies as $(\lambda^{(i-1)}/\lambda^{(i)})^k$. We may choose $\lambda^{(i)}$ so that γ is smaller than the number of duplicates generated by the selection operator. Another way to design a gene deletion schedule is to fix γ to a constant value much less than 2^{t_s}, where t_s is the number of selection repetitions per length reduction. This is done because we expect binary tournament selection to roughly double the proportion of the best individuals during each invocation. Define the ith length-reduction ratio as $\rho_i = \lambda^{(i)}/\lambda^{(i-1)}$. Using the asymptotic relation for $\gamma = (\lambda^{(i-1)}/\lambda^{(i)})^k = \rho_i^{-k}$, we recognize that the assumed fixed γ roughly implies a fixed length-reduction ratio $\rho = \rho_i$, for all i, and we calculate the total number of length reductions required to reduce the string length to $O(k)$. Assuming final string length equal to ζk, where $\zeta \geq 1$, the number of length reductions (t_r) required is given by the equation

$$\ell'/\rho^{t_r} = \zeta k. \tag{4.5}$$

Simplifying, we obtain $t_r = \log(\ell/\zeta k)/\log \rho$. This suggests that if the $\lambda^{(i)}$ values are chosen to make the deletion so that the length-reduction factor ρ is a constant, t_r varies as $O(\log \ell)$.

At the worst case when only one gene is deleted at a time, the complete filtering process is bounded by $O(\ell)$.

Unfortunately, the growth of building-blocks can be uneven due to cross-competition among the different blocks. Although the purpose of thresholding is to restrict such cross-competition, the method of choosing a thresholding parameter suggested in Section 4.3.2 does not provide a completely satisfactory solution. The following section suggests some further modifications that bolster the performance of the fmGA.

4.4 Some Modifications

The success of the fmGA depends upon the ability of the building-block filtering process to filter out the order-k building-blocks. Rigorous testing of the fmGA on different test function pointed out that the methodology for designing the building-block filtering schedule described in the previous section, does not give satisfactory performance all the time. This resulted in taking a two-pronged strategy to improve the performance of fmGA as listed in the following:

1. Study the fundamental processes in the building-block filtering process and develop an alternate methodology for designing the filtering schedule.

2. Apply fmGA iteratively at each level.

Each of them is discussed in the following sections.

4.4.1 Designing building-block filtering: An alternate approach

The approach for designing the building-block filtering schedule described in the previous section makes a simplistic assumption. It assumes that binary tournament thresholding selection increases the number of strings containing good building blocks by a factor of 2^{t_s}, where t_s is the number of times selection is applied. Although it is correct for binary tournament selection (Goldberg & Deb, 1991a), it is indeed a very idealized picture of thresholding selection. In this section, I take a more realistic approach to model the effect of thresholding selection. The objective of the following discussion is to outline an alternate methodology for designing the filtering schedule, which will be detailed in Appendix C.

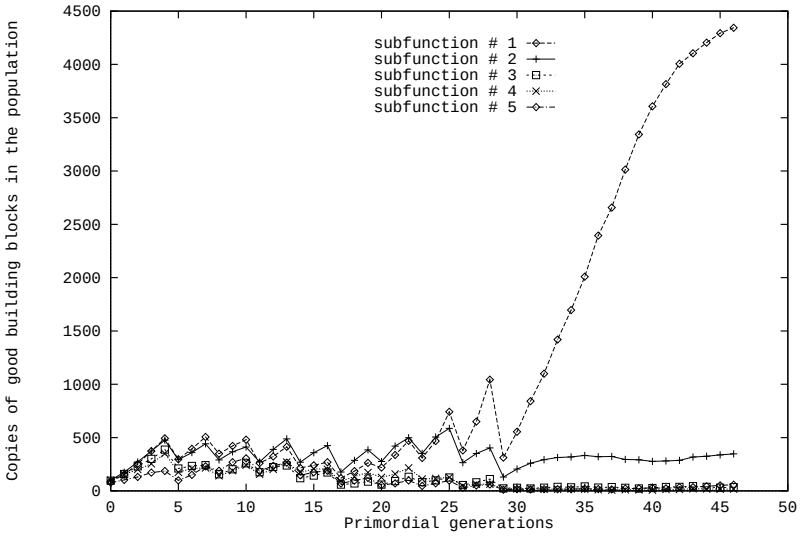


Figure 4.5: This figure shows the number of copies of building-blocks in the primordial generations for a 25-bit bounded deceptive problem (Goldberg, 1992e) with 5-bit subfunction size. The filtering schedule is designed using the methodology described in previous section, originally presented in (Goldberg, Deb, Kargupta, Harik, 1993). This figure shows that during the late stages of the primordial stage this approach suggests weaker thresholding that may lead to uneven growth of building-blocks. Length reduction ratio, $\rho = 0.5$, population size $n = 5000$.

The objective of designing a filtering schedule is to make sure that all the strings containing better building-blocks grow evenly. Unfortunately, a filtering schedule designed based on the assumption that binary thresholding selection continues to evenly grow all the building-blocks can lead to serious trouble. This can be illustrated by observing the growth of building-blocks during the primordial stage with a filtering schedule designed based on the approach described in the previous section. Figure 4.5 shows the growth of the five building-blocks corresponding to the five subfunctions of a 25-bit problem. The population size is 5000 and the length reduction ratio is 0.5. The chosen value of $c'(\alpha')$ is 2.0. Figure 4.6 shows the corresponding filtering schedule. As we see from Figure 4.5, the filtering schedule developed using this simple approach does not accomplish even growth of all the building-blocks during the late stages of the primordial stage. In this case, one of the building-blocks starts growing at a rate much higher than those of others. This eventually leads many building-blocks to the verge of extinction.

Clearly, the approach described in the previous section makes the thresholding too relaxed during the late episodes of the primordial stage. Instead of simply increasing the thresholding

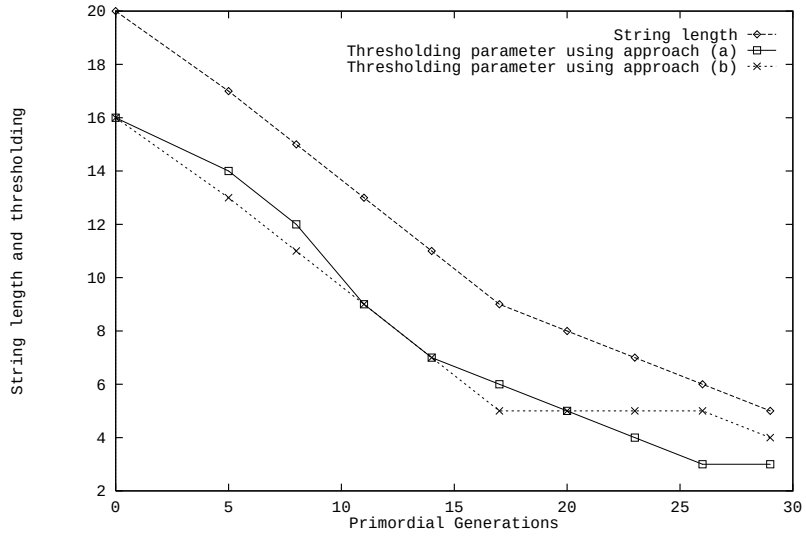


Figure 4.6: This figure shows three graphs. The topmost one shows the string length reduction schedule. The other two graphs show the thresholding parameter values during the different episodes of the primordial stage designed using two methodologies. Approach (a) denotes the scheduling procedure originally suggested by Goldberg, Deb, Kargupta, Harik (1993). Approach (b) represents the alternate methodology developed in Section 4.3.4. Note that approach (b) offers a more restricted value of thresholding parameter during the late episodes of the primordial stage, and this reduces the uneven growth of building-blocks.

parameter value by an arbitrary amount during the late episodes, let us first take some time to understand the fundamental processes in thresholding selection and building-block filtering of fmGA. There are primarily three design variables in building block filtering, namely

- threshold parameter value, θ for a given string length,
- the duration for which selection continues before gene deletion is applied, and
- the number of genes deleted during a particular gene deletion operation.

Although thresholding minimizes comparison among classes from different partitions, it allows a certain degree of cross-competition for eliminating the instances of bad partitions. Unfortunately, this cross-competition can also act against our main objective—even growth of building-blocks. The cross-competition among the instances of classes from different but good partitions may result in eliminating one another. At the initial stage with a random population, the effect of such cross-competition is minimal; however, this may become an acute problem

at the later stages of the primordial phase unless the filtering schedule is properly designed. We need a model of selection in the presence of cross-competition to address this problem. Once we do that, we can decide how many times selection should be applied without making cross-competition a serious problem.

Thresholding restricts a string from competing with some strings in the population. In other words, thresholding creates a niche of a string within the population. Two strings are members of the same niche if they can compete with each other. The whole population is divided into such overlapping niches. The gene deletion operator of the fmGA removes some members of a niche and also introduces some new members. One possible way to view the combined effect of gene deletion and thresholding is to look at their effect on the thresholding niche. If the size of the thresholding niche of a string is too small, its growth will be restricted; on the other hand, if the size is too large, cross-competition among the strings from different partitions will be very high. Therefore, one possible design objective for choosing a filtering schedule is to minimize drastic changes in the size of the niche.

The number of genes deleted can be decided using a chosen length reduction factor as described in the previous section. This approach does not suggest any change in this aspect.

It is possible to come up with a technique for designing a filtering schedule that uses a more realistic model of thresholding selection in the presence of cross-competition among different building-blocks and minimizes the changes in the size of the thresholding niche. This formulation is somewhat involved, and it is presented in Appendix C. The building-block filtering schedules designed using this approach have been more successful in maintaining all the different building-blocks in the population. Figure 4.7 shows the growth of the five building-blocks for the same problem considered in Figure 4.5. All the fmGA parameters are kept the same. This apparently demonstrates that this approach offers a more successful way to maintain the growth of all the building-blocks especially during the late stages of the filtering phase.

4.4.2 Iteration within each level

As we noted earlier, the messy GAs work level-wise, where each level corresponds to the order of possible decomposability of the problem. Iterative application of the fmGA within each level increases the chance of success. Those building-blocks that could not be detected during

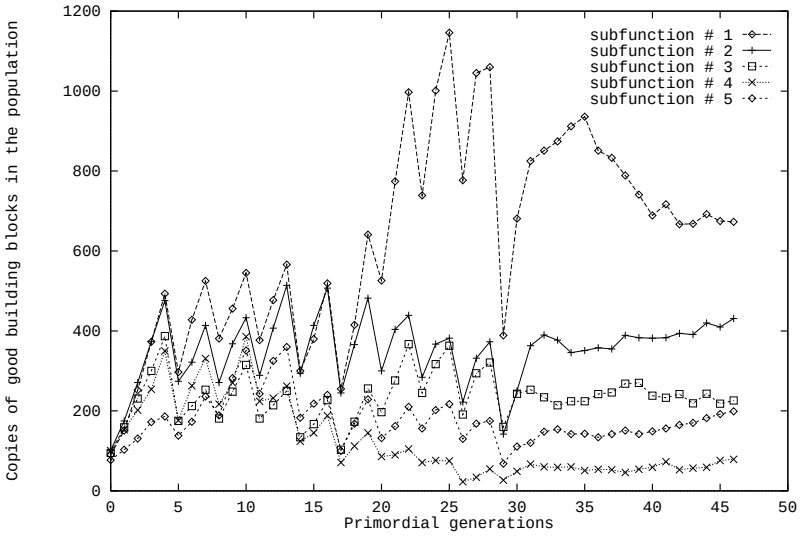


Figure 4.7: This figure shows the number of copies of building-blocks in the primordial generations for a 25-bit problem with 5-bit subfunction size. The filtering schedule is designed using the methodology described in this section that tries to minimize the change in the size of the subpopulations. This approach appears to be much more successful than the previous approach in maintaining an even growth of all the building-blocks.

the first iteration may be detected in the later iterations, since the template now contains the correct building-blocks identified in the previous iterations.

Iterative application of the fmGA within each level also becomes useful when very large optimization problems need to be solved. For very large problems, the population size required by fmGA becomes considerably large. When computer hardware is a constraint, iterative application of the fmGA with suboptimal population sizing becomes a necessity.

This completes our discussion on different components of the fmGA. The following section presents the overall algorithmic picture by combining all these different components.

4.5 Organization of the fmGA

Like the original messy GA, the fmGA goes through a level-wise processing of equivalence classes. The selection-only primordial stage is replaced by a primordial stage that uses thresholding selection and gene deletion for filtering out the building-blocks along with the probabilistic initialization of population. The pseudo-code describing different steps of the fmGA is

shown in Table 4.8. The working of the fmGA is very similar to that of the original messy GA. The main differences of the fmGA are the following:

1. probabilistically complete initialization,
2. gradual reduction of string length by random deletion of genes during primordial stage, and
3. iteration within the each level.

The following section summarizes the conceptual strengths and weakness of the fmGA.

4.6 Major Conceptual Underpinnings of the fmGA

The fmGA replaced the enumerative initialization ($O(\ell^k)$) of the original messy GA by an $O(\ell)$ probabilistically complete initialization. Since the building-block filtering schedule can have at most $O(\ell)$ steps, the overall sample complexity of the primordial phase is $O(\ell^2)$.

The fast messy GA is one among the rare class of algorithms that realized the need for detecting appropriate relations. The simple GA fails to work on many problems where its assumptions about good relations does not match with the problem. The fast messy GA eliminates this problem. Moreover, the fmGA takes a serious decision-theoretic approach for proper evaluation of equivalence classes. It sizes the population to guarantee correct evaluation with a certain degree of confidence. Another important aspect of the fmGA is that unlike the original version of messy GA, it brings in some of the benefits of implicit parallelism.

However, the fmGA has some weak points too. First of all, the fmGA addresses only one problem of mGA—the lack of implicit parallelism. Therefore, it still suffers from other problems of the mGA, as noted in the previous chapter. In the following, I list some of the conceptual weak points that the design of fmGA suffers from:

- Relation comparison is accomplished by comparing classes from different relations. As we noted earlier, this is a general problem of thresholding selection. This leads to cross-competition among classes from good relations, and as a result, maintaining all the building-blocks during the filtering stages may become difficult.

Since the primordial phase does not have any mechanism to construct strings with good building-blocks, maintaining even growth of all the building-blocks is sometimes (especially for large problems) difficult. However, the iterative level-wise working of fmGA can be used to gradually improve the solution.

- The thresholding selection during the building-block filtering process may also make the algorithm slower, compared to the running time of the sGA and mGA.
- When string length is close to the problem length ℓ, evaluation of strings on the basis of a single template may not be accurate, and as I pointed out earlier, the whole idea of comparing classes in the context of a single template should be investigated more thoroughly.

The following chapter presents the results of testing the fmGA on different classes of problems.

```
/* Initialization */
level = 1;
template = random_string; // Random initialization of the template
While(fmga_termination_criterion ≠ TRUE) )
{
  Repeat // Iterative application of fmGA within each level
  t = 0; // Set the generation number to zero.
  Probabilistic_Initialize(Pop(t), level); // Initialize the population
  Evaluate(Pop(t), template); // Evaluate the fitness values
  Repeat // Enter primordial phase
  {
    episode = 0;
    Repeat
    {
      Thresholding_Selection(Pop(t)); // Select better strings
      episode = episode + 1;
    }
    Until (episode < episode_max(t))
    GeneDeletetion(Pop(t)); // Delete Genes
    Evaluate(Pop(t)); // Evaluate fitness
    t = t + 1; // Increment generation counter
  }
  Until (primordial_termination_criterion == TRUE)

  t = 0;
  Repeat // Enter juxtapositional phase
  {
    Selection(Pop(t));
    Cut_and_Splice(Pop(t)); // Apply cut and splice operator
    Mutation(Pop(t)); // Apply mutation
    Evaluate(Pop(t), template);
    t = t + 1;
  }
  Until ( juxtapositional_termination_criterion == TRUE)

  template = optimal_string(Pop(t), level); // template remains locally optimal
 }
  level = level + 1;
}
```

Figure 4.8: A pseudo code for the iterative fast messy GA.

Chapter 5

Testing of the Fast Messy Genetic Algorithm

The emphasis on searching for better relations and classes makes messy GAs attractive for problems in which little knowledge is available about the good relations defined by the chosen representation. However, the theoretical conclusions should also be backed by adequate experimental results on sufficiently difficult problems. Unfortunately, like many other issues in BBO, problem difficulty is yet another topic that is not well understood. Chapter 2 presented the SEARCH perspective of problem difficulty in BBO. In this chapter I first design a testbed for the fmGA using our understanding of problem difficulty and then report the test results.

Section 5.1 discusses different classes of problems with bounded difficulties. A testbed is designed following the conclusions of this section. Section 5.2 presents the test results. Section 5.3 discusses the qualitative significance of the experimental results. Finally, Section 5.4 summarizes the main observations of this chapter.

5.1 Design of Experiments

Designing difficult problems requires some understanding about the problem difficulty in BBO. The objective of this section is to discuss the rationale behind the design of experiments for testing the fmGA. The fast messy GA is tested against the following facets of problem difficulty:

1. problem size,

2. bounded inappropriateness of the representation, the source of relations, and

3. sampling-noise.

Each of them is described in the following sections.

5.1.1 Problem size

Unfortunately, for most of the interesting BBO problems, the search space grows exponentially with the problem dimension ℓ. This does not necessarily mean exponential sample complexity in SEARCH, since it tries to solve a problem using low-order relations in a probabilistic and approximate way. However, the cardinality of the set of relations needed to solve a problem in SEARCH depends on the size of the search space. Therefore, as we expect, the sample complexity in SEARCH increases as the problem dimension increases. The later sections of this chapter that present experimental results observe the growth of sample complexity with problem size for certain classes of problems.

The following section considers the difficulty due to the inappropriate source of relations. I consider one way to introduce bounded inappropriateness in representation—order-k deception.

5.1.2 Inadequate source of relations: Deception

Inadequate sources of proper relations that do not satisfy the delineation requirement can create major trouble for a blackbox search algorithm. When the set of relations provided to the algorithm does not have a sufficient number of appropriate relations, a search for good relations is useless and therefore success in solving the problem efficiently is very unlikely. The delineation-ratio provides a way to measure the adequacy of the source of relations. As the delineation-ratio decreases, searching for appropriate relations become more and more difficult. Deceptive problems (Goldberg, 1987) offer one way to introduce problem difficulty by controlled inappropriateness of the representation. The objective of this section is to discuss this class of problems and explain the difficulty introduced by these problems from the SEARCH perspective.

Although the fundamental idea of deception is quite general and can be used to quantify inappropriateness of any arbitrary representation, traditionally deceptive problems have been defined in the context of sequence representation with either binary (Goldberg, 1987) or n-ary

alphabet (Kargupta, Deb, & Goldberg, 1992). Here we shall be restricted to binary representation. There exist several different definitions of deceptive problems (Bethke, 1981; Goldberg, 1987). Although all of them capture the fundamental idea, we shall use Goldberg's definition here.

Definition 4 (k-th order deception) *Let $\hat{x} \in \mathcal{X}$ be a suboptimal point in the solution domain such that $\Phi(\hat{x}) < \Phi(x^*)$, where x^* is the optimal solution. Let $C_{h,i}$ and $C_{*,i}$ be the schemata or classes in partition i that contain $\hat{x}$ and x^*, respectively. A partition i is deceptive if $\overline{\Phi}(C_{h,i}) > \overline{\Phi}(C_{j,i}), \forall j$, where $\overline{\Phi}$ denotes the average objective function value of the distribution within the class. A problem is called partially deceptive in the chosen representation if there exists a partition that is deceptive. A problem is fully deceptive in the chosen representation if all the partitions are deceptive. A problem is order-*k *fully deceptive if all the partitions of order less than* k *are deceptive.*

Deceptive problems provide a way to increase or decrease inappropriateness in representation by controlling the delineation-ratio. As we increase the value of k, by definition of deception, all the relations of order less than k do not properly delineate the search space. Therefore, the number of relations that do not properly delineate the search space, and thereby the delineation-ratio, decreases as k increases, and vice versa. When k is a constant, a representation with k-th order deception becomes an instance of the class of order-k delineable problems and hence is solvable in polynomial sample complexity. However, as we increase the value of k, solving deceptive problems become more and more expensive, and the success probability decreases due to the decrease in delineation-ratio.

Deceptive problems seem quite appropriate for testing an algorithm against bounded inadequacy in representation. They will play a major role in testing fast messy GAs. The following section considers the possibility of making decision error in detecting good classes and the resulting source of difficulty.

5.1.3 Inappropriate decision making: Signal and noise

SEARCH realizes the importance of decision making in BBO, and it takes a non-parametric approach to compute the probability of decision error. However, this is not the first time it has been addressed. Several efforts (Goldberg, Deb, & Clark, 1992; Holland, 1975; Kargupta,

1995b; Rudnick & Goldberg, 1991) have been made for quantifying the sampling error in the decision making processes in search. In this section I first briefly review these efforts. Finally, I pose a generalized signal-to-noise measure using a Bayesian approach developed elsewhere (Kargupta, 1995b) and define *crosstalk* (Goldberg, Deb, & Thierens, 1993) as a possible source of decision error.

Holland (1975) used the bandit framework for developing an "optimal" trial allocation strategy in genetic algorithms. Holland considered the decision-making in a single bandit and developed a sampling strategy that minimize the overall loss.

A similar parametric approach was taken by Rudnick and Goldberg (1991) and by Goldberg, Deb, and Clark (1992). They explicitly computed the error probability for determining the sample size (population size) in GA. In a parametric approach, the underlying distribution is assumed. Therefore, to compute the error probability in choosing a better class out of the two competing classes using parametric approach, we need to know the underlying distribution. By central limit theorem (Feller, 1959), the random variable, representing the objective function value of the observed samples from a class, is normally distributed. This observation lies at the heart of their parametric formulation of decision error probability. The mistake probability depends on the distribution of the convolution of two competing classes (say, $\hat{C}_{j,i}$ and $\hat{C}_{k,i}$). By central limit theorem, both $\hat{\Phi}_{j,i}$ and $\hat{\Phi}_{k,i}$ are normally distributed with mean $\mu_{j,i}, \mu_{k,i}$ and variance $\frac{\sigma_{j,i}}{n}, \frac{\sigma_{k,i}}{n}$ respectively. The mean and variance of the underlying distribution $\hat{\Phi}_{j,i}$ are $\mu_{j,i}$ and $\sigma_{j,i}$ respectively. The convolution of $\hat{\Phi}_{j,i}$ and $\hat{\Phi}_{k,i}$ is also normally distributed. The mean and variance of the convolution distribution are $(\mu_{j,i} - \mu_{k,i})$ and $(\frac{\sigma^2_{j,i}}{n} + \frac{\sigma^2_{k,i}}{n})$, respectively. The error probability can be approximated by the tail of the normal distribution. The sample size can then be appropriately chosen for a desired level of error probability. This can be done by finding an error probability α such that

$$z^2(\alpha) = \frac{(\mu_{j,i} - \mu_{k,i})^2}{\frac{\sigma^2_{j,i}}{n} + \frac{\sigma^2_{k,i}}{n}},$$

where $z(\alpha)$ is the ordinate of the unit, one-sided normal deviate. The corresponding sample size can be found by

$$n = z^2(\alpha)\frac{\sigma^2_{j,i} + \sigma^2_{k,i}}{(\mu_{j,i} - \mu_{k,i})^2}$$

$$= \quad z^2(\alpha)\left(\frac{\sigma}{d}\right)^2.$$

The term d/σ is often called the *signal-to-noise* ratio of the convolution. Goldberg, Deb, and Clark (1992) considered the evaluation of different partitions independent of each other. Holland's analysis on optimal allocation of trial also considered decision making for a single partition. Grefenstette and Baker (1989) pointed out some of the restrictions of the independent partition perspective of a GA.

A more general version of the signal-to-noise framework that accommodates mutually dependent simultaneous decision making has been recently proposed elsewhere (Kargupta, 1995b). This approach can be explained by considering a simplified case. Consider two relations r_i and r_j. Assume that both of them have an index value of 2. Let us denote the two classes, defined by relation r_i, by $C_{k,i}$ and $C_{*,i}$. When they are compared with each other using some statistic $\mathcal{T}$, then $C_{k,i} \leq_{\mathcal{T}} C_{*,i}$ defines a convolution variable. Let us denote this by s_i. Similarly, for the relation r_j, the comparison among $C_{k,j}$ and $C_{*,j}$ can define a convolution variable s_j. For some κ number of relations, a vector of such convolution variables can be defined. Let us denote this convolution vector by $\mathbf{s}$. The overall decision error probability (p) depends on the joint distribution of these convolution variables. As the search algorithm generates new samples, the state of the convolution vector $\mathbf{s}$ changes. This in turn changes the error probability p. If we denote the change in $\mathbf{s}$ by $\delta\mathbf{s}$, and expand $p(\mathbf{s}+\delta\mathbf{s})$ along any particular s_i using the Taylor series, then it can be shown that $p(\mathbf{s}+\delta\mathbf{s})$ depends on $E[\delta s_i]$ and covariance$(\delta s_i, \delta s_j)$. Higher order terms of the Taylor series can also be considered. However, we choose to neglect them for the sake of simplicity. The generalized signal-to-noise measure can now be defined as

$$\text{Signal-to-noise measure} = \frac{\mathrm{E}[\delta s_i]}{\sum_{i,j} \text{covariance}(\delta s_i, \delta s_j)}. \tag{5.1}$$

The expected fluctuation (numerator of Equation 5.1) toward the desired decision is viewed as the *signal*, and the random effects represented by the covariance term in the denominator is called the *noise*.

This signal-to-noise measure can be used to understand the decision-making process from a different angle. Both a wrong signal and increased noise can cause decision error. Noise can be introduced into the convolution kernel from two different sources:

1. *Intra-partition noise* originates from the individual variance terms (the diagonal elements) of the covariance matrix. This is essentially very similar to the *collateral noise* defined in (Rudnick & Goldberg, 1991).

2. *Inter-partition noise* originates from contributions from the off-diagonal terms of the covariance matrix.

Intra-partition noise has been addressed in (Goldberg, Deb, & Clark, 1992; Holland, 1975; Rudnick & Goldberg, 1991) The role of inter-partition noise was also noted by Goldberg, Deb, and Clark (1992). Kargupta (1995b) explicitly addressed inter-partition noise from the signal-to-noise framework. The difficulty in decision making, caused by the cross-covariance terms (i.e. off-diagonal terms), is called *crosstalk*.

Decision making can be made difficult by several means. Following Goldberg, Deb, and Thierens (1993), I list some of the possibilities here:

- multimodality of the objective function
- noisy objective function
- operator introduced noise
- intra-partition noise
- crosstalk

Among them, problems with crosstalk have received little attention. However, there is enough reason (Forrest & Mitchell, 1993; Kargupta, 1995b) to believe that crosstalk can offer a great degree of difficulty by increasing noise during the late stages of search. Problems with multimodality intra-partition noise and crosstalk will be used to test fmGA.

The following section presents the experimental setup and the results.

5.2 Experimental Results

The fast messy GA is used to solve different test functions. The test problems are designed using our understanding of problem difficulty, as described in the previous section. The fast messy GA is primarily tested against three aspects: (1) growth of search space, (2) bounded

inappropriateness in representation, and (3) sampling-noise. This section presents the results of the experiments.

The test set up is briefly described in Section 5.2.1. Results for large order-3 and order-5 deceptive problems are reported in Section 5.2.2. This is followed by results for several bounded deceptive problems with different scaling and mixed sizes of the building-blocks. I consider the problems used in Deb and Goldberg (1993) and present the results in Section 5.2.3 through 5.2.6. Problems with crosstalk are considered in Section 5.2.7.

5.2.1 Experimental setup

Two different classes of problems are used to test against the different aspects of problem difficulty listed earlier. These two classes are

- bounded deceptive problems
- royal road functions

Deceptive problems test an algorithm against inappropriateness of representation up to a certain order. Trap functions (Ackley, 1987) are known to be deceptive (Deb & Goldberg, 994b). I use trap functions to construct several classes of deceptive problems of different sizes and different scaling. Bounded deceptive problems are also multimodal. This will be pointed out later when we construct them.

The royal road functions (Forrest & Mitchell, 1993) are reported to have crosstalk elsewhere (Kargupta, 1995b). Problems with crosstalk offer difficulty by making the decision process noisy.

Before I present the experimental results, I would like to remind the reader that unlike the simple GA, messy GAs do not assume any prior knowledge about the linkage, i.e. what the good relations are. Therefore, they have to pay the computational price for it which may sometime require more function evaluations compared to an algorithm like the simple GA which assumes linkage information. The following section present the test results for bounded deceptive problems.

5.2.2 Test function 1: Bounded deceptive problems

Order-k delineable, bounded deceptive problems of any arbitrary length can be constructed by concatenating fully deceptive subfunctions of size k one after another as described by Goldberg,

Deb, and Clark (1992). This section presents the test results for such deceptive problems with different values of k, in which each of the subfunctions is uniformly scaled.

5.2.2.1 Order-three deceptive problems

This test function is constructed by concatenating multiple numbers of order-3 subfunctions. Each of these subfunctions is an order-three trap function. The particular version of the trap function used can be defined as follows:

$$\begin{aligned} f(x) &= \ell \quad \text{if} \quad u = \ell \\ &= \ell - 1 - u \quad \text{otherwise,} \end{aligned}$$

where u is the number of 1s in the string x and ℓ is the string length. The chosen problem size $\ell = 90$. Therefore, the number of subfunction is $m = 30$. If we carefully observe this trap function, we shall note that it has two peaks. One of them corresponds to the string with all 1s and the other is the string with all 0's. Since $\ell = 90$, the overall function contains 30 subfunctions; therefore, this function has 2^{30} local optima, and among them, only one is globally optimal.

The population-sizing equation developed elsewhere (Goldberg, Deb, & Clark, 1992) is used for choosing the population size. By noting that $\beta = 1$ for order-3 trap function, this equation can be reduced to $n = 16z^2(m-1)n_g$, where n_g is the factor contributed from the probabilistically complete initialization of the fmGA. For an order-3, 90-bit problem with the initial string length of 87, $n_g = 1.1$ and $m = 30$. With $z^2 \approx 9$, the population size becomes approximately 4500. This was the chosen population size. Figure 5.1 shows the performance of fmGA along generations for a 90-bit order-3 deceptive trap function. Three sets of data are presented, with each corresponding to a different level of fmGA. Figure 5.2 shows the corresponding building block filtering schedule, specified by the sequence of thresholding parameter and the string length. The total number of function evaluations needed is 256,500. For all the runs of this section, the initial template is randomly generated, unless otherwise noted. Binary thresholding selection with a shuffle number equal to the population size is used for all the experiments reported in this chapter. For every experiment, the cut and splice probabilities are chosen to be 0.04 and 1.0, following Deb (1991).

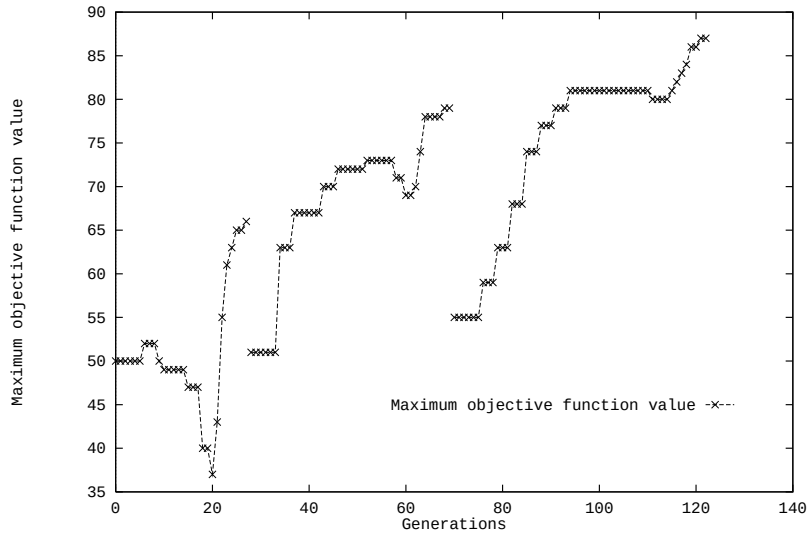


Figure 5.1: Maximum objective function value in different generations for a 90-bit, order-3 deceptive trap function. The fast messy GA found the best solution at the end of the third level. Population size, $n = 4500$

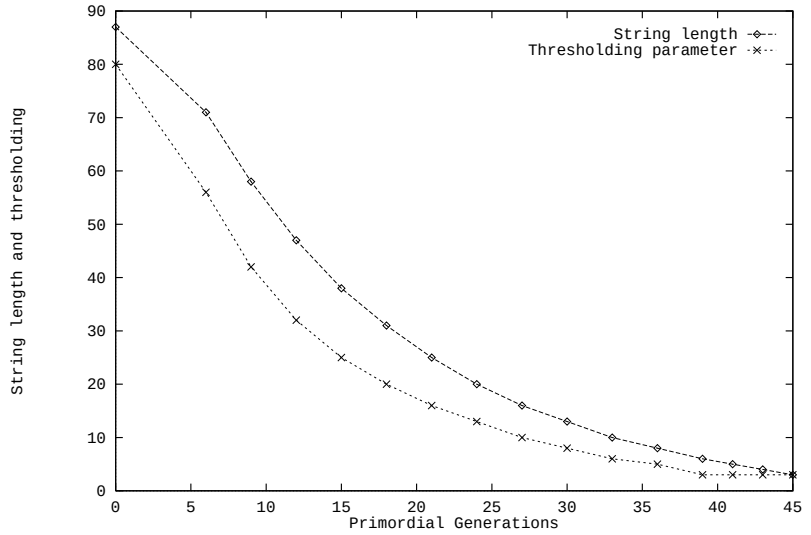


Figure 5.2: Building-block filtering schedule for order-3 level of the fmGA.

The following section considers order-five deceptive functions.

5.2.2.2 Order-five trap functions

Problems comprised of order-five trap functions are constructed in a way similar to the construction of order-three deceptive functions. The same trap function is used to generate bigger functions by concatenating order-five subfunctions together. For order-5 trap function, $\beta = 1.4$, and therefore, population size $n = 90z^2(m-1)n_g$. The population size for an order-5, 100 and 150-bit problems is around 7500 and 8500, using values of 4 and 3 for z^2 respectively. Figure 5.3 and 5.5 show the building-block filtering schedules for 100 and 150-bit problems. Figure 5.4 and 5.6 present the performance of fmGA for these two problems. The fast messy GA found the best solution for the 100-bit problem. A confidence factor of 3 corresponds to an error probability of approximately 0.03 in normal tail. Since $(1-0.03)*50 \approx 147$, the performance of the fmGA matches the prediction. The total number of function evaluations for the 100-bit and 150-bit problems are $100,5000$ and $425,000$ respectively. Note that the number of function evaluations for the 100-bit problem is larger than that for the 150-bit problem because in the former case, the fmGA found the best solution after several iterations; on the other hand the fmGA could not improve the best solution of the first iteration using additional iterations. Since these are big problems and population sizes are quite large, only order-5 level of the fmGA is run with a locally optimal template in order to reduce the computation time.

The next immediate step is to test the fmGA for even larger problems. However, since the population sizing equation is exponential in k, optimal population size becomes very large for larger problems even for order-5 deceptive problems. For example, in a 200-bit problem with initial string length of 195 the population size $n \approx 3900z^2$. For any reasonable choice of z^2 population size becomes very large. Therefore, we choose to experiment the degradation in the performance of the fmGA with the increase in problem size, keeping the population size constant. Again, practical consideration of the available hardware leads us to assume a locally optimal template with all zeros and only the order-5 level of the fmGA is run.

Figure 5.7 shows the degradation of performance for a constant population size of 5000 in even larger problems. The best solution found by several level of regular fmGA iterations is reported. Let us spend some time on this graph. When the number of subfunctions is 20 (i.e., a $5 \times 20 = 100$-bit problem), a population size of 5000 has a confidence factor $c =$

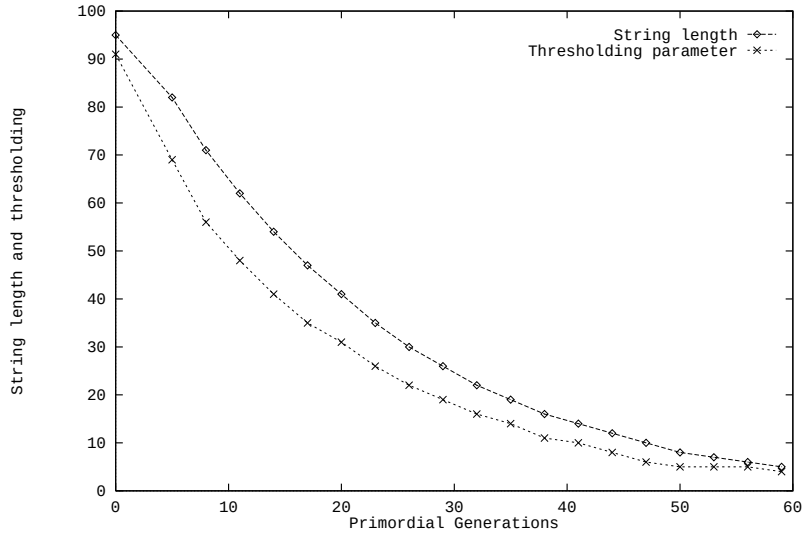


Figure 5.3: Building-block filtering schedule for order-5 level of a 100-bit problem.

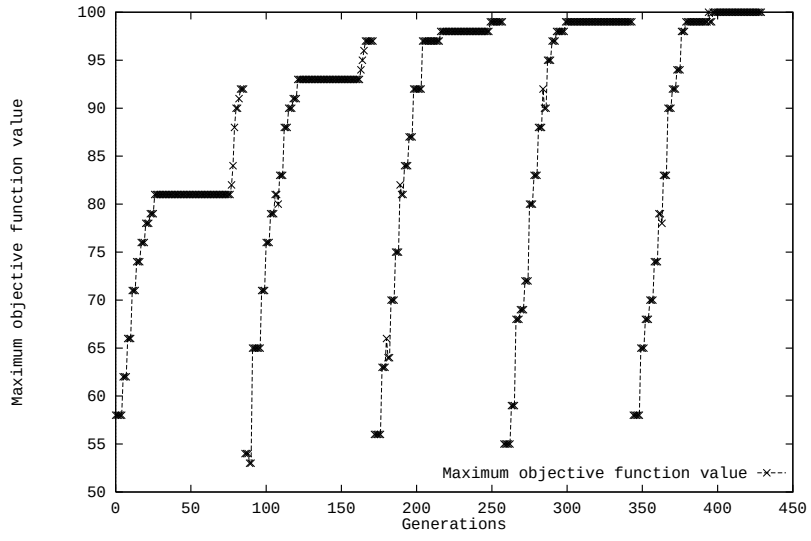


Figure 5.4: Maximum objective function value in different generations for a 100-bit, order-5 deceptive trap function. The fast messy GA found the best solution. Population size, $n = 7500$

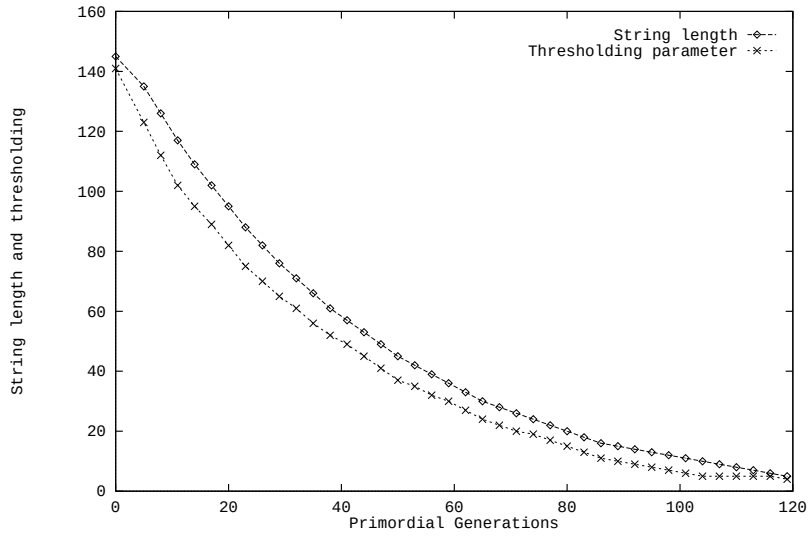


Figure 5.5: Building-block filtering schedule for order-5 level of a 150-bit problem.

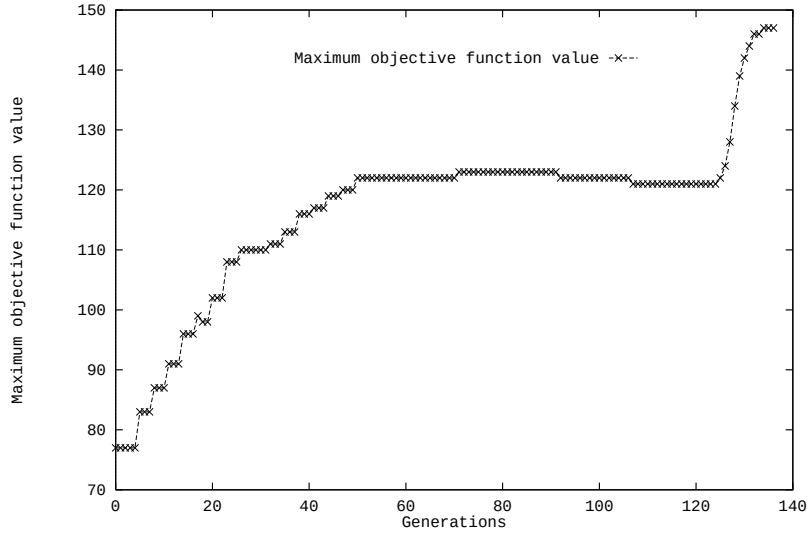


Figure 5.6: Maximum objective function value in different generations for a 150-bit, order-5 deceptive trap function. The fast messy GA found the correct solution for 27 out of the 30 subfunctions. Population size, $n = 8500$

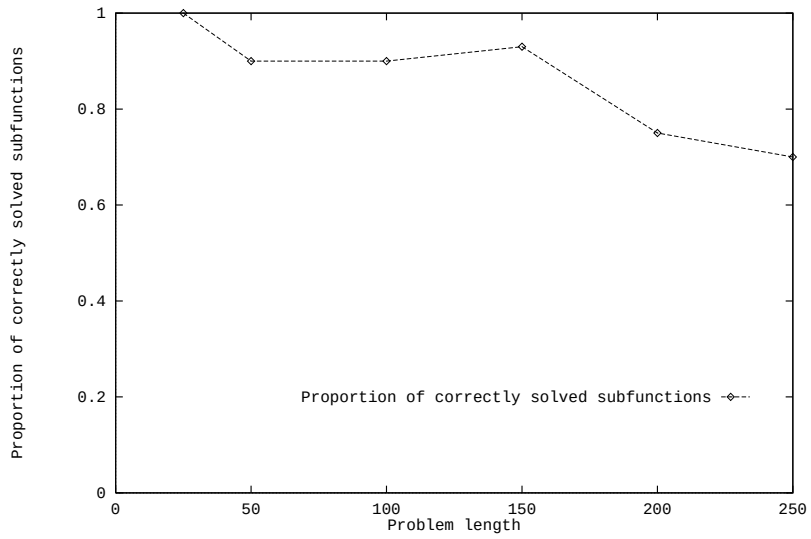


Figure 5.7: Quality of the best solution found for different size of problems. Population size is 5000 for all the experiments.

$\frac{5000}{2052} \approx 2.4$, according to the population sizing equation developed by Goldberg, Deb, and Clark (1992). A confidence factor of 2.4 corresponds to an error probability of 0.07 in the tail of normal distribution. The performance of the fmGA matches with this prediction. For the 40 subfunction problems the corresponding error probability is approximately 0.25. This again seems to match with the prediction of the population sizing equation.

In real applications different parts of the problem may be differently scaled and the subfunctions may be of different sizes. The original messy GA (Deb & Goldberg, 1993; Goldberg, Korb, & Deb, 1989) was tested on many test functions with nonuniform scaling and with subfunctions of mixed sizes. In the following sections, I present the experimental results of testing the fast messy GA against those functions.

5.2.3 Test function 2: Nine up, one down

This test function is designed using order-3 deceptive trap functions. Ten 3-bit deceptive functions are concatenated to form a 30-bit problem. The leftmost subfunction is scaled down by a factor of three, and the remaining subfunctions are scaled up by a factor of seven. The

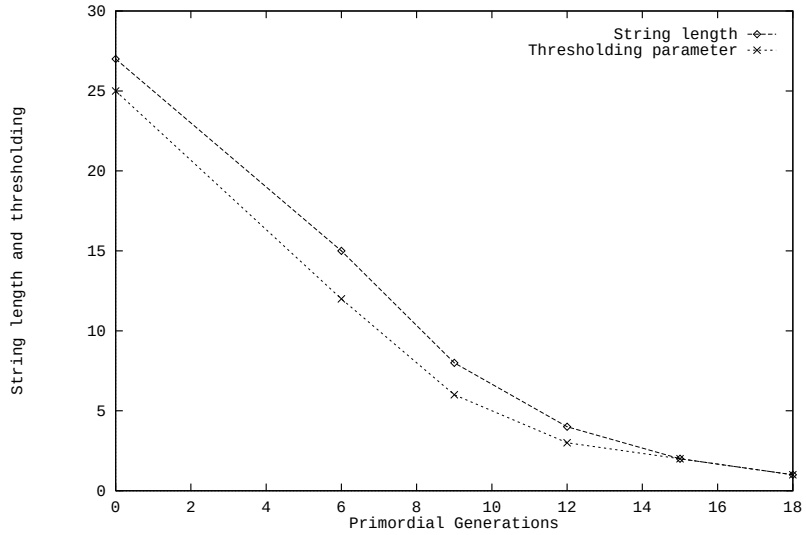


Figure 5.8: Building-block filtering schedule for an order-3 deceptive, 30-bit problems.

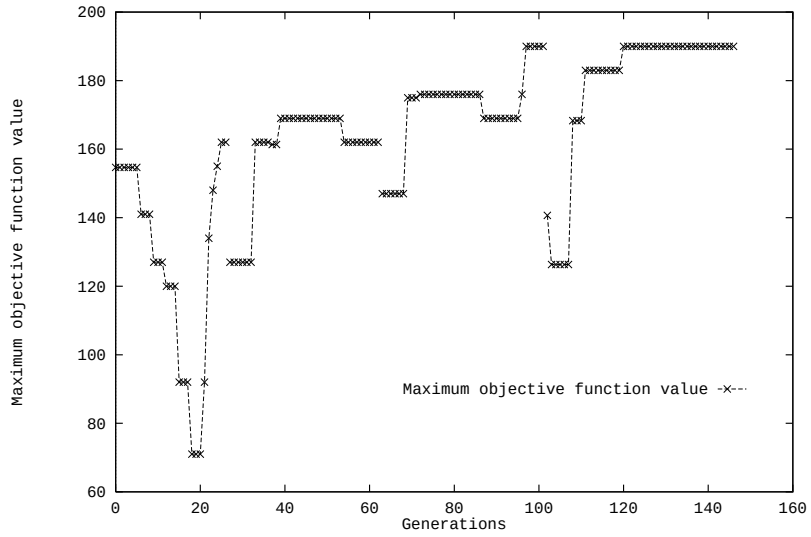


Figure 5.9: Test function 2: Best solution of different generations. The optimal solution is found at the end of the third level.

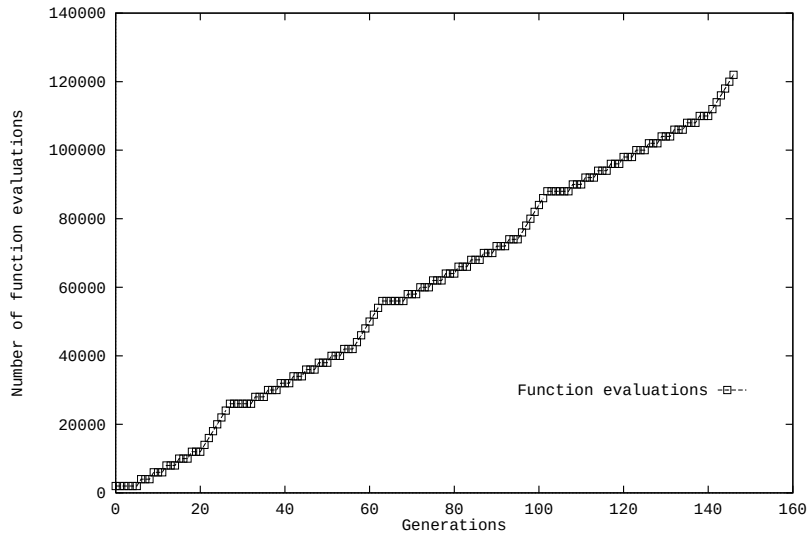


Figure 5.10: Test function 2: Number of function evaluations along generations, starting from the order-1 level.

objective is to test the fmGA against problems in which different variables of the problems are differently scaled.

Since the problem size is smaller than that of the previous test problems, using the simpler approach for designing the building-block filtering schedule as suggested in Goldberg, Deb, Kargupta, and Harik (1993) is appropriate. I therefore take this simpler approach to investigate how well it works for problems with small sizes. For all Sections 5.2.3 through 5.2.6, the level-wise fmGA is used starting from the order-1 level, initialized with a randomly chosen template. No additional iterations within each level was required for any of these test cases.

Figure 5.8 shows the string length reduction schedule and the corresponding threshold parameter values. The chosen population size for this and all the following test problems is 2000. Figure 5.9 shows the performance of the fmGA on this test function. The fast messy GA successfully found the correct solution for all the subfunctions by the end of the third level. Note that the four sets of data are plotted in the same figure. Each of these data sets comes from an fmGA run at a particular level, starting from one through four. Figure 5.10 shows the growth of the number of function evaluations along generations.

5.2.4 Test function 3: One up, nine down

This test function is designed again using order-3 deceptive trap functions. Ten 3-bit deceptive functions are concatenated to form a 30-bit problem. The leftmost subfunction is scaled up by a factor of seven, and the remaining subfunctions are scaled down by a factor of 3. Figure 5.11 shows the performance of the fmGA on this test function. The fast messy GA successfully found the correct solution for all the subfunctions, by the end of the third level. Figure 5.10 shows the growth of the number of function evaluations along generations, starting from the order-1 level.

5.2.5 Test function 4: A linear scaling

The same 30-bit problem is used to construct this test function, with the only difference in scaling. In this case a linear scaling is used. All the subfunctions are numbered, starting from left to right. The leftmost subfunction is numbered 1, the next one gets 2, and so on. Denote this number by α. The scaling factor of the subfunction with number α is 10α. The first subfunction is scaled by a factor of 10, the second one is scaled by 20, and so on. Figure 5.13 shows the performance of the fmGA for this test problem. The fast messy GA successfully found the correct solution for all the subfunctions by the end of the third level iteration. Figure 5.14 shows the growth of the number of function evaluations along generations.

5.2.6 Test function 5: Problems of mixed sizes

This test function is constructed by combining different sized subfunctions. A 31-bit problem is designed by concatenating one 3-bit subfunction and seven 4-bit subfunctions. All the subfunctions are fully deceptive trap functions. Figure 5.15 shows the performance of the fmGA. The fast messy GA successfully found the correct solution for all the subfunctions by the end of the fourth level iteration. Figure 5.16 shows the growth of the number of function evaluations along generations. The following section considers the royal road functions.

5.2.7 Test function 6: Royal Road functions

Forrest, Mitchell, and Holland developed a class of functions known as *royal road functions*, as shown in Table 5.1. Royal road functions introduce difficulty for Simple GA by introducing

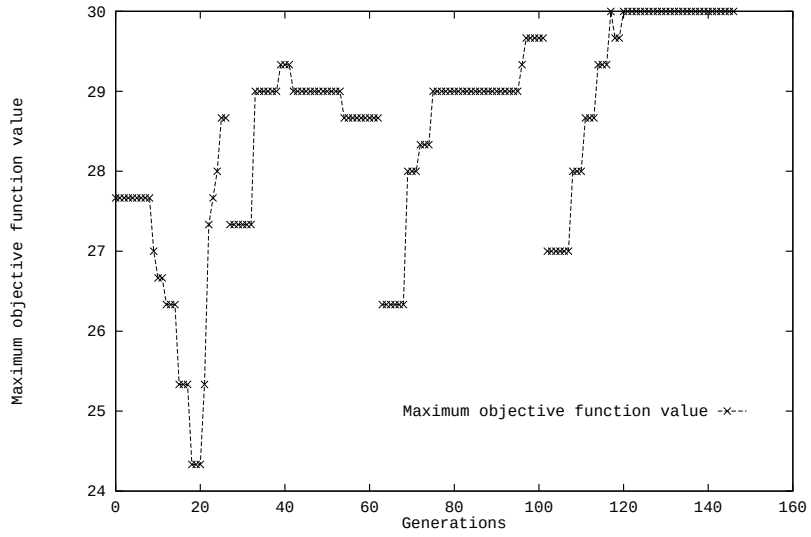


Figure 5.11: Test function 3: Best solution of different generations. The optimal solution is found at the end of the third iteration.

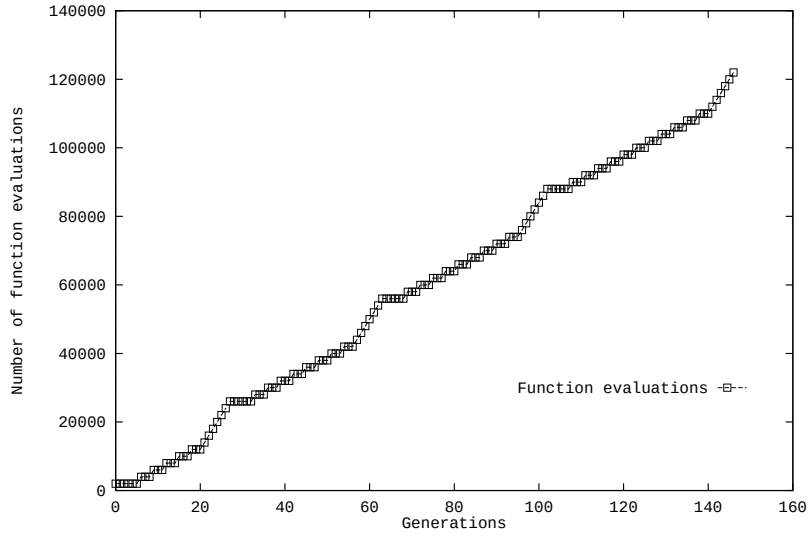


Figure 5.12: Test function 3: Number of function evaluations along generations, starting from the order-1 level.

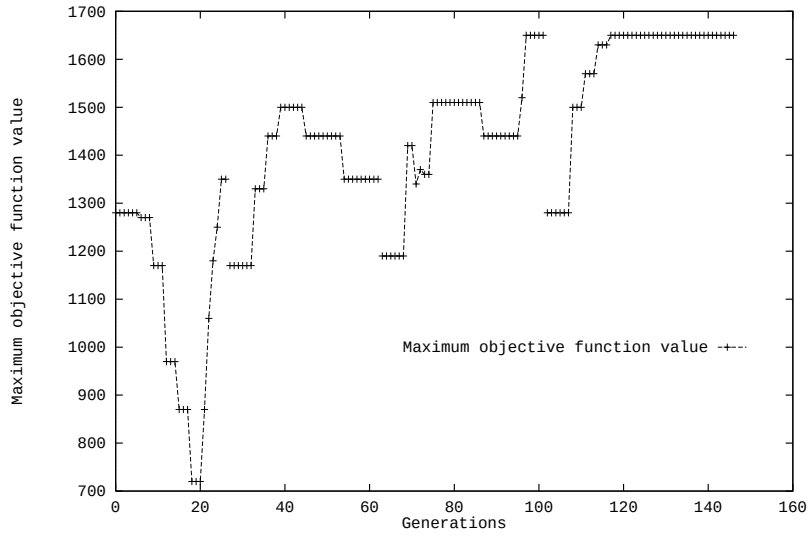


Figure 5.13: Test function 4: Best solution of different generations. The optimal solution is found at the end of the third level.

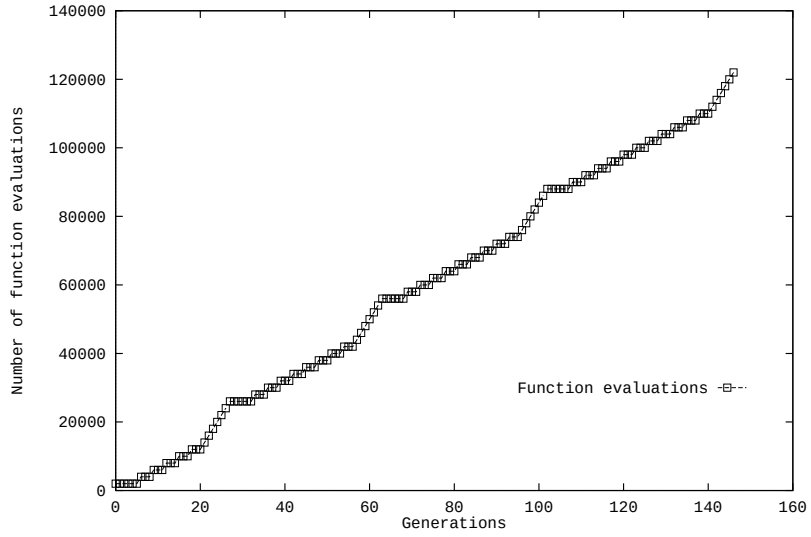


Figure 5.14: Test function 4: Number of function evaluations along generations, starting from the order-1 level.

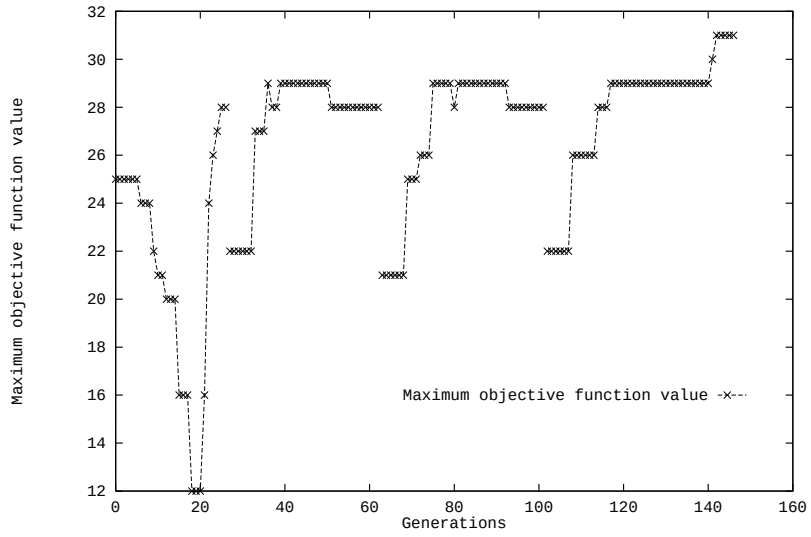


Figure 5.15: Test function 5: Best solution of different generations. The optimal solution is found at the end of the fourth level.

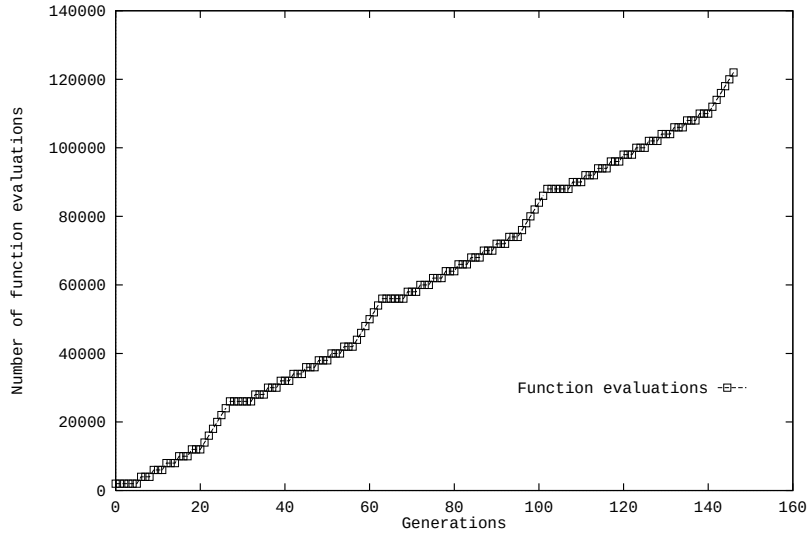


Figure 5.16: Test function 5: Number of function evaluations along generations, starting from the order-1 level.

String	Fitness (R1)	Fitness (R2)
1#######	8	8
#1######	8	8
##1#####	8	8
###1####	8	8
####1###	8	8
#####1##	8	8
######1#	8	8
#######1	8	8
1 1 ######		+16
##1 1 ####		+16
####1 1 ##		+16
######1 1		+16
1 1 1 1 ####		+32
####1 1 1 1		+32

Table 5.1: R1 and R2: A single 1 and single # symbol represent 8 1s and 8 null characters respectively. If a string is an instance of any of the above hyperplanes, it gets the associated pay-off. For R1, the optimal solution has 64 1s, its objective function value is 64. R2 gives additional reinforcements (denoted by +) for multiple sets of eight consecutive 1s. The optimal solution for R2 is the one with 64 1s with a function value of 192.

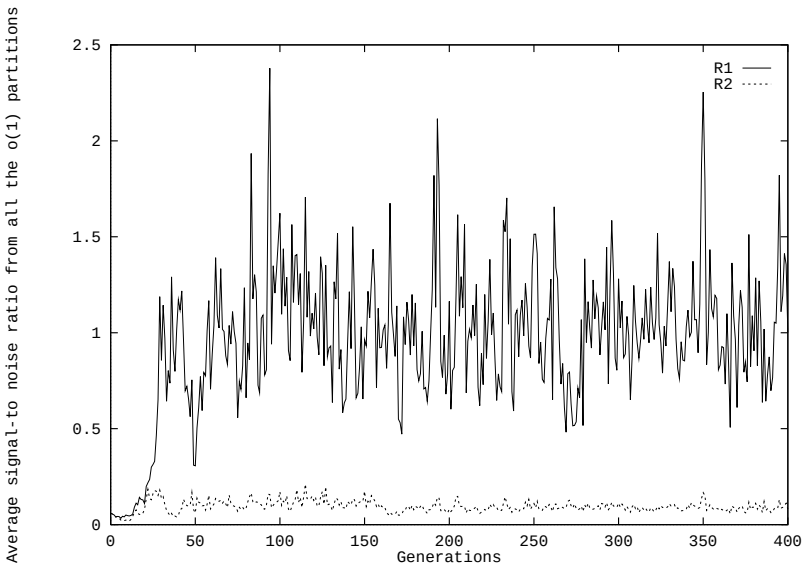


Figure 5.17: Variation of the signal-to-noise ratio along generations. Signal-to-noise ratio is averaged for all the order-1 partitions. This clearly shows that decision making in R1 should be easier than that in R2.

crosstalk noise into the convolution kernel. In the beginning of this chapter, I defined crosstalk and pointed out this property of royal road functions. The primary objective of this section is to present the experimental results for demonstrating the performance of the fmGA on royal road functions. However, before we do that, let us briefly review crosstalk in royal road functions.

R1 does not allocate bonus credit to the intermediate building blocks; on the other hand, R2 does. It has been reported (Forrest & Mitchell, 1993) that the sGA finds solving R2 difficult compared to R1. The question is: If detecting good relations and good classes is made easier by introducing these additional bonus credit, then why is it that the sGA finds R2 difficult? The answer to this question is quite straightforward. These additional credits do not make the decision-making any easier. It has been shown by Kargupta (1995b) that these additional credits introduce covariance noise. As we recall, we defined crosstalk as the aspect of problem difficulty introduced by the covariance among the different partitions. R2 introduces difficulty due to crosstalk. Figure 5.17 reproduced from (Kargupta, 1995b), shows that the signal-to-noise ratio is much lower in R2 compared to R1. Therefore, decision-making becomes more difficult in R2. Kargupta (1995b) constructed other instances of similar problems with crosstalk that may be difficult to solve, in which decision makings for different relations are not independent.

The simple GA does not have any mechanism to restrict cross-competition among different partitions (relations). However, the thresholding selection of the fmGA tries to minimize the competition among classes from different partitions. Therefore, it will be interesting to observe the performance of the fmGA for this class of problems. The following part of this section presents the test results.

The fast messy GA is used to solve both of these problems with a population size of 4000. Figure 5.18 shows the building-block filtering schedule, that is, used for solving both R1 and R2. Figure 5.19 and 5.20 show the best solution found in different generations for R1 and R2, respectively. The total number of function evaluations needed to solve R1 and R2 are 204,000 and 136,000, respectively. For all the runs on the royal road function, the chosen template was locally optimal, a string of all zeros. Again, like the large bounded deceptive problems, we skipped the low-order levels and used order-8 level fmGA with multiple iterations within. Note that the number of function evaluations is less in the case of R2. As we noted earlier, the fmGA seems to be more resistant to crosstalk because of the presence of thresholding selection.

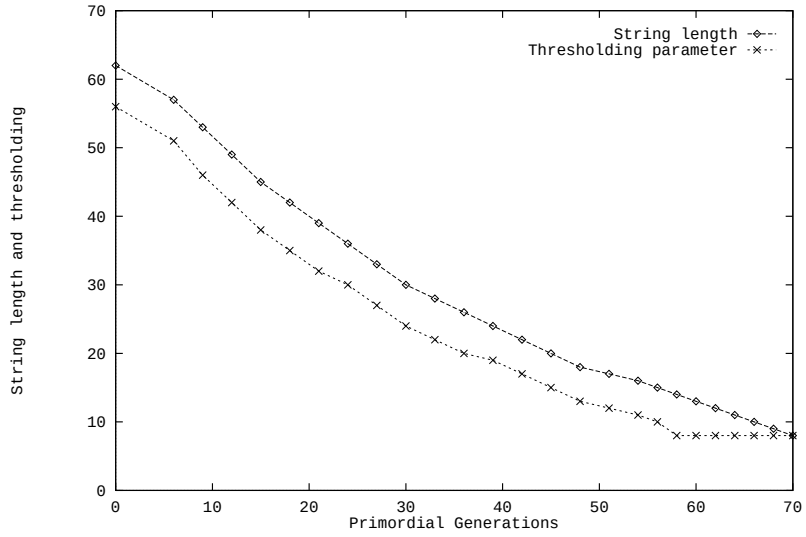


Figure 5.18: R1 & R2: Building block filtering schedule.

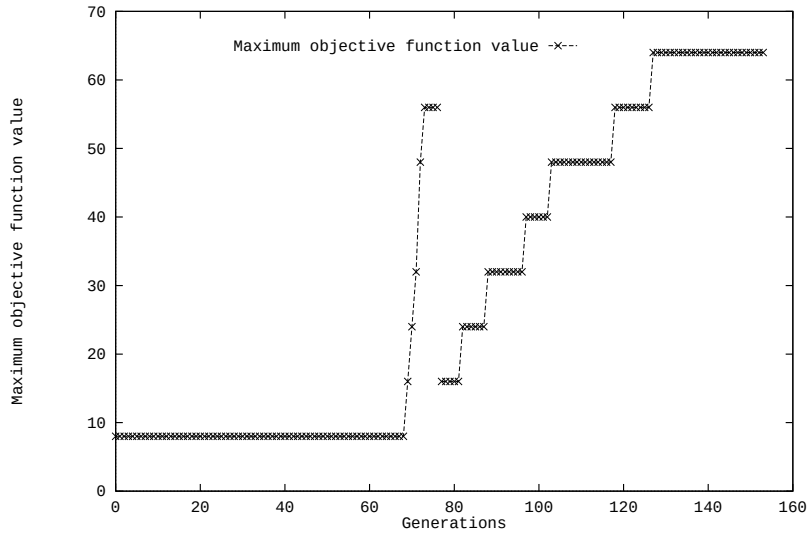


Figure 5.19: R1: Best solution found in different generations. The optimal solution is found at the end of the second iteration.

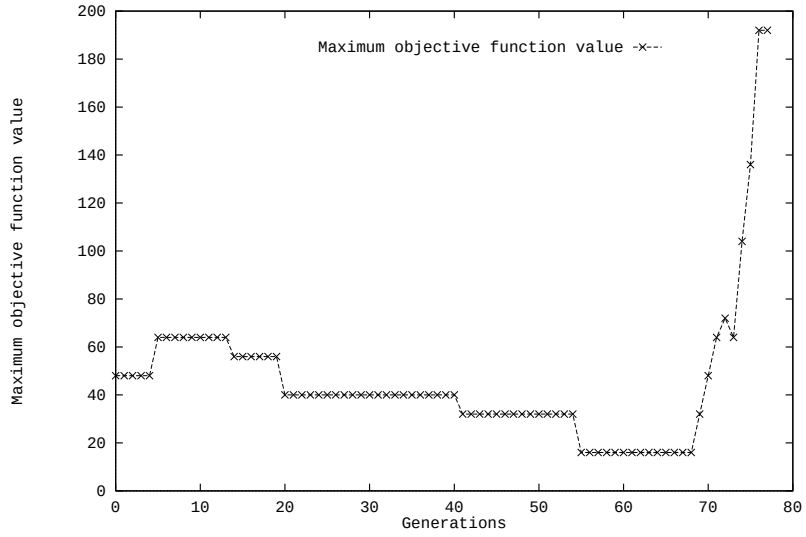


Figure 5.20: R2: Best solution found in different generations. The optimal solution is found at the end of the first iteration.

A different version of the royal road function has recently been proposed (Jones, 1994). I shall refer to this function by R3. R3 can be defined as follows. Let us assume that j and i index the levels of hierarchy (1 is the lowest level) and target classes (1 is at left), respectively. There are 2^k target classes at level 1 and 2^{k-j} target classes at level $j+1$. Each target class is defined over some b loci. Define

$$\begin{aligned} \Phi_1(j) &= u^* + (n(j)-1)u \text{ if } n(j) > 0 \\ &= 0 \text{ if } n(j) = 0, \end{aligned}$$

where $n(j)$ is the number of found targets at level j; u^* and u are parameters such that $u^* > u$. Now define

$m(i)$ = number of loci that have values common with the optimal solution in class i.

Then,

$$\Phi_2(i) = m(i)v \text{ if } m(i) < m^* + 1$$

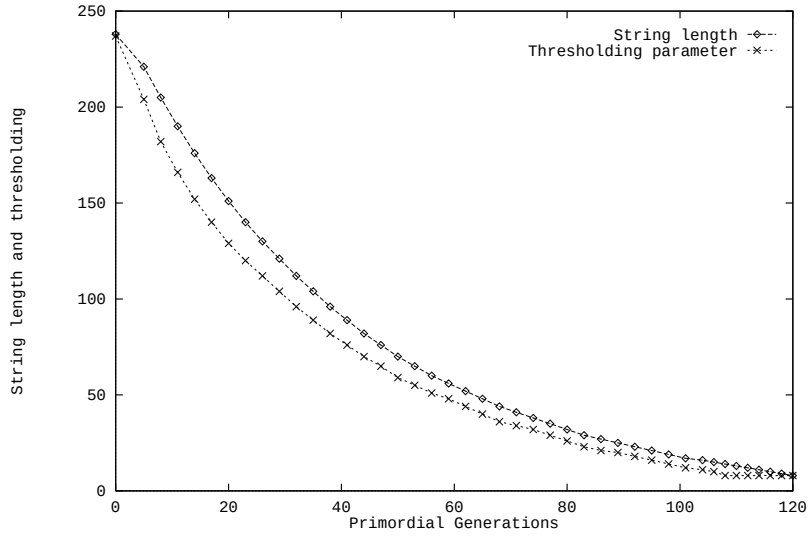


Figure 5.21: R3: Building block filtering schedule.

$$= \quad -(m(i) - m^*)v \text{ if } m^* < m(i) < b$$

$$= \quad 0 \text{ otherwise}$$

Now we can define the function R3:

$$R3(x) = \sum_j \Phi_1(j) + \sum_i \Phi_2(i)$$

The fast messy GA is used to solve R3 without any prior knowledge about the linkage information. The used parameters for defining R3 are as follows: $\ell = 240$, $b = 8$, $k = 4$, $u^* = 1$, $u = 0.3$, $v = 0.02$ and $m^* = 4$. Each target schema is separated by 7 intervening bits. For this parameter setting, the maximum possible objective function value is 12.8.

A population size of 2000 is used for this set of runs. Figure 5.21 shows the building block filtering schedule that is used for solving both R1 and R2. Figure 5.22 shows the best solution found in different generations for R3. The total number of function evaluations needed to solve R3 is 302,000. The fast messy GA found the correct solution at the end of third iteration. The following section analyzes the experimental results presented in this chapter.

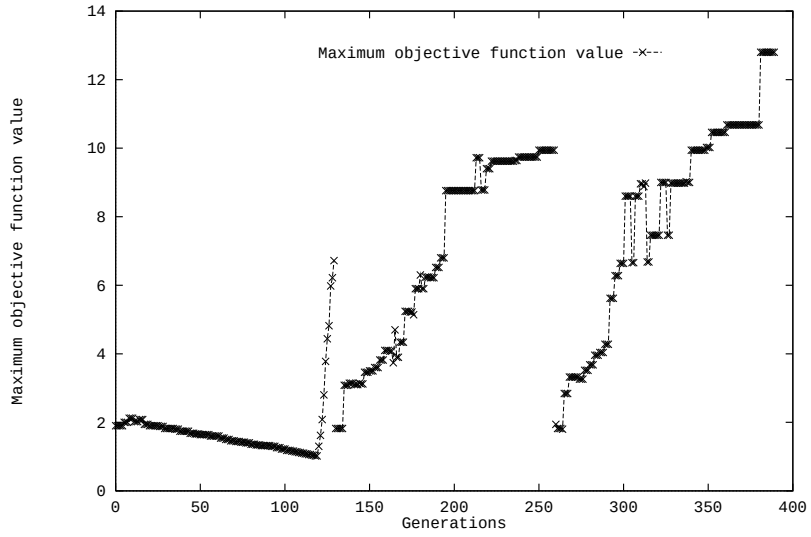


Figure 5.22: R3: Best solution found in different generations. The optimal solution is found at the end of the third iteration.

5.3 Analysis of Results

The fast messy GA has been tested against different kinds of order-k delineable problems that can be solved in SEARCH in polynomial complexity. These problems include,

1. bounded inappropriateness in representation,
2. multimodality,
3. different size building-blocks,
4. non-uniformly scaled building-blocks, and
5. crosstalk

Bounded deception provides a nice way to introduce order-k delineability in a problem. Therefore, bounded deceptive problems were used to design all the different test suites except for the last category—problems with crosstalk. The fmGA solved order-3 and order-5 deceptive problems. Section 5.2.2 presented the results on these problems. In addition to being boundedly inappropriate in representation, these problems also have multimodality. The order-3 deceptive

90-bit problem has around 2^{30} local optima. The largest problem that was used for testing the fmGA is an order-5 deceptive, 250-bit problem. It has even higher order of inappropriateness in representation and millions of local optima. Iterative application of the fmGA solved these problems.

In real optimization problems different building-blocks of the solution are likely to be differently scaled. Section 5.2.3, Section 5.2.4, and Section 5.2.5 presented the results for different test problems with non-uniform scaling. The fmGA successfully solved these problems. These test results also point out that thresholding selection is effective in minimizing the cross-competition among building-blocks up to a certain degree.

The fmGA has also been tested against problems in which building-blocks of different size are needed to solve them correctly. The results presented in Section 5.2.6 showed that the fmGA can solve this kind of problems.

Problems with crosstalk are also used in the test suite. Royal road functions served this purpose. Royal road function R2 offers difficulty by introducing covariance noise into the convolution kernel (Kargupta, 1995b). John Holland (Jones, 1994) recently developed Royal road function R3 which offers different kinds of difficulty including inappropriateness of representation and crosstalk. Iterative application of the fmGA soved this problem.

Despite these successes, the fast messy GA faces a problem. The problem is regarding its scalability to very large problems. Maintaining all the building-blocks together during the building-block filtering process is difficult since cross-competition among different building-blocks can eventually lead to the extinction of good building-blocks from the population. Although the modified thresholding technique improved the performance, this does not solve the problem completely. Multiple iteration of the fmGA is introduced because of this reason. Again this improved the performance and we have been able to solve up to 250-bit problems with millions of local optima and bounded inappropriateness in representation. However, this problem with cross-competition has a deep root in the design of the fmGA. As we noted in the previous chapter, although the sample space is separate, the relation and class spaces of the fmGA are implicitly defined together. This is also a characteristic of the original messy GA. Comparison between two relations is accomplished by relaxing the thresholding parameter and by physically comparing two classes from different partitions. This leads to cross-competition that can eliminate some classes from good partitions.

The test results of this chapter can also be interpreted from the following perspectives:

- success in detecting the correct relations,
- reliability of overall success in finding the optimal solution, and
- the accuracy of the solution found.

The performance of the fmGA along each of these directions will be discussed in the following paragraphs.

The fmGA uses the building-block filtering process to detect the correct relations. All the test problems considered in this chapter are order-k delineable, and therefore the complexity is polynomial, provided there exists an appropriate measure that can correctly detect the good relations. For all the chosen test problems, there is a selective advantage of the correct relations when evaluated in the context of the locally optimal template. Selection should be able to detect this, and it did. Therefore, it is fair to conclude that the fmGA has been successful in finding the right relations for the classes of test problems considered here. For different problems, different measures may be required; however, the fundamental mechanism should work.

Although the iterative version of the fmGA with modified thresholding scheme solved problems up to 250-bit with millions of local optima, thresholding selection has limitations in its capability to control cross-competition among building-blocks. This may lead to elimination of some building-blocks and thereby it may reduce the reliability of the fmGA.

The solutions found for most of the test problems are the optimal solution. When the population size was suboptimal, the fmGA showed a reasonable degradation of performance. Therefore, the fmGA is quite successful on this ground.

The SEARCH framework proved that the class of order-k delineable problems can be solved in polynomial sample complexity. The complexity of the original messy GA was polynomial; still for any practical purpose scalability of the original mGA was poor because of the exponential growth in the complexity with the order of delineability, k. The primary motivation behind the development of the fmGA was to exploit implicit parallelism and relax the process of decision making in relation and class spaces for reducing the complexity of search. The previous work by Goldberg, Deb, Kargupta, and Harik (1993) hypothesized that the fmGA might be able to solve order-k delineable problems reliably and accurately in subquadratic complexity. Although this

appears to be true for the size of problems considered in this dissertation, elimination of cross-competition among the building-blocks is a hurdle that needs to be crossed for materializing that hypothesis for even larger problems. As we noted earlier the implicit implementation of the relation comparison process is the fundamental reason behind this problem. Explicit decomposition, either spatially or temporally, of the relation and class decision making processes is essential to ovecome this difficulty.

The following section summarizes the main points of this chapter.

5.4 Summary

This chapter started with a note on problem difficulty from the SEARCH perspective. This is followed by a description of experimental setup and the test results. The fast messy GA is tested along three dimensions of problem difficulty: (1) bounded inappropriateness in representation, (2) problem size, and (3) decision error due to sampling noise. Order k deceptive problems offer bounded inappropriateness in representation. Both order-3 and order-5 bounded deceptive problems contain millions of local optima. Apart from that, they have bounded level of inappropriateness in the representation. These are difficult problems. The fast messy GA performed quite well against these problems. For large problems, multiple iterations were used, since the fmGA was not able to maintain all the building-blocks for all subfunctions in a single iteration. The test results for deceptive problems with mixed building-block sizes and non-uniform scaling showed that the fmGA is capable of solving such problems. Thresholding selection seems to be quite effective in controlling cross-competition among differently scaled building-blocks.

The fmGA is also tested against large deceptive problems. Iterative application of the fmGA has been successful, although maintaining all the building-blocks remains a problem. As pointed out earlier, the process of comparing relations in fmGA may be the fundamental reason behind this.

Sampling noise can also introduce difficulty for a particular algorithm. In this work, I restricted myself to problems with noise due to crosstalk–covariance contribution to noise kernel. The fast messy GA is tested against the class of royal road functions, which are known to contribute crosstalk noise in a GA (Kargupta, 1995b). The fast messy GA successfully solved all the Royal Road functions. It is also observed that the fmGA solved R2 with a smaller

number of function evaluations. This is because the effect of crosstalk is minimized in the fmGA because of the thresholding selection.

The following chapter presents the results of applying the fmGA for solving the so-called scan-to-scan correlation problem.

Chapter 6

Application of the Fast Messy GA to Target Tracking Problem

The issue of simultaneous tracking of targets is becoming increasingly important in military endeavors as the air defense system is becoming increasingly vulnerable to air breathing threats, such as bombers and air and sea-launched missiles. As the target indications (*blips*) are detected, the trajectories of the corresponding blips need to be determined using a series of frames of imaging data. As the number of targets increases, the number of possible trajectories (tracks) increases very rapidly. The tracking problem can be categorized into two different problems—track initiation and track continuation. In track initiation phase, three frames of imaging data are usually considered and a track initiated from each blip in the first frame is found. In track continuation phase, each track found by the track initiation phase is continued as new frames of imaging data arrives. The former problem is harder than the latter, because for N targets in each of three initial frames, there are a total of N^3 tracks possible and for N^3 tracks there are a total of $(N!)^2$ valid solutions possible. Even though there exists a number of pattern recognition approaches that uses some *windowing* techniques, they are computationally expensive. Recently, neural networks have been used successfully in this tracking problem with a reasonable number of evaluations (Elsley, 1990). This method uses a preprocessing technique that reduces the number of possible tracks from n^3 down to about $5n$ or so in order to keep the size of the neural network within manageable limits. But this drastic reduction may cause some needed tracks to be eliminated from consideration. In this work, we use a more relaxed

preprocessing technique that eliminates infeasible tracks and a fast messy genetic algorithms to find correct initiation tracks.

In the remainder of this section, I first present a brief formulation of the problem, followed by the simulation results.

6.1 Problem Formulation

Target tracking has been a problem of interest for a long time, mainly because of its importance in the field of air defense systems. Given a large number of potential target indications ("blips") the job of a target-tracking system is to determine the trajectories of the corresponding blips through a series of frames of sensory imaging data. This can be divided into two steps:

- trajectory initiation: generation of tracks from the imaging data at the initial stage.
- trajectory continuation: identifying the location of a blip in the incoming data frame, one time step at a time.

Trajectory initiation is known to be more difficult than trajectory continuation. This is not only because of the high computational effort required, but also because it demands a high degree of accuracy in the performance of the tracker; otherwise the next stage of trajectory continuation will be increasingly misleading. The sequel presents a target tracker, based on a fast messy genetic algorithm and its performance in the track initiation problem.

The first step for addressing this problem using a messy GA is to design a suitable coding scheme, which transforms the problem space into the messy string space. One of the interesting features of the track initiation problem is that different candidate tracks may share common blips. This may introduce a large number of redundant invalid strings—strings containing tracks with many shared blips. A coding scheme may worsen the situation if it is not designed carefully. The coding scheme used for the present work tries to minimize this problem. Every string is comprised of a sequence of *genes*, where every gene corresponds to a unique blip in the initial frame and the corresponding allele represents a candidate track initiated from that particular blip. In order to do this in a systematic way, every blip in the initial frame is numbered as $1, 2, 3 \cdots N$, each corresponding to a blip. In the present implementation, the first gene corresponds to the number 1 blip, second gene corresponds to the number 2 blip and so on.

After the preprocessing stage all candidate tracks initiating from each of these blips are serially labeled. Figure 2 depicts this clearly. Every allele is a label corresponding to a candidate track, initiated from the blip, denoted by the gene. If we denote a track corresponding to an allele by $\mathbf{x}_i$ then,

$$\mathbf{x}_i = x_i(t_0)x_i(t_1)\cdots x_i(t_f)$$

is a candidate track, generated by connecting $x_i(t_0)$, $x_i(t_1)$, $\cdots x_i(t_f)$ blips in the data frame corresponding to t_0, t_1 $\cdots t_f$ time respectively.

The objective function value of a string is measured by using the *acceleration* and *intensity* information obtained from the imaging data:

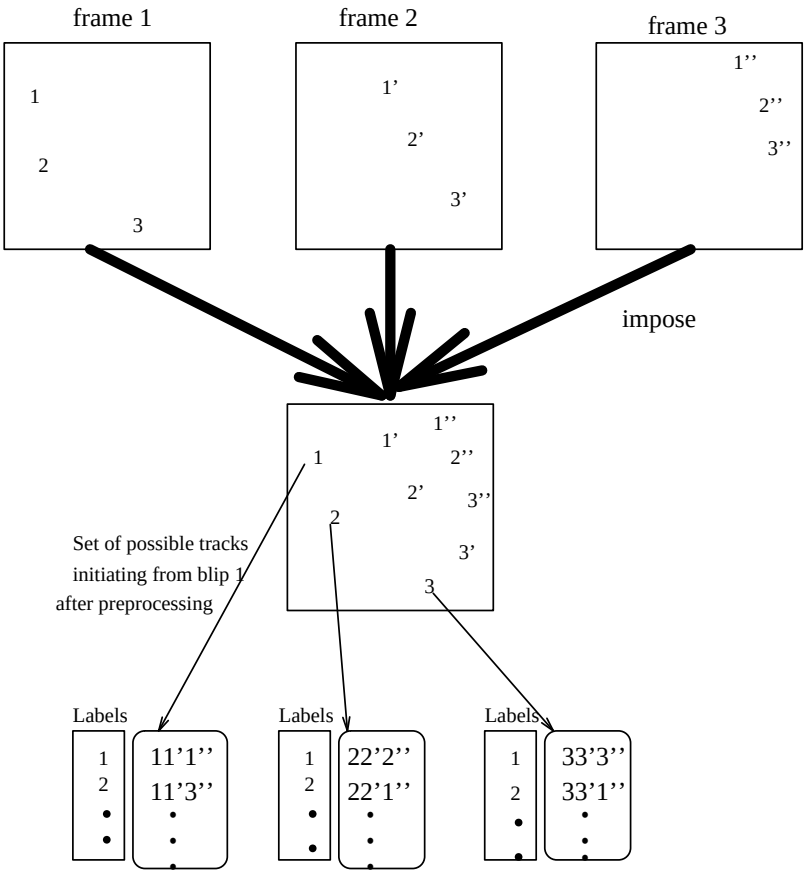
$$f(\mathbf{x}) = \alpha \sum_{i=1}^{N} A_i + \beta \sum_{i=1}^{N} dI_i + \gamma \sum_{t=t_0}^{t_f} \sum_{j=1}^{N} (1 - \sum_{i=1}^{N} \delta(blip_j(t), x_i(t))) \tag{6.1}$$

$$\begin{aligned} \delta(blip_j(t), x_i(t)) &= 1 \qquad \text{if } blip_j(t) = x_i(t) \\ &= 0 \qquad \text{otherwise} \end{aligned}$$

where, A_i is the acceleration of the i-th track and dI_i is the intensity variation along that track; α, β and γ are constants. This expression is similar to that used elsewhere (Elsley, 1990). The last term in equation 6.1 is a penalty term used for minimizing the sharing of blips; t_0 and t_f correspond to the initial and final time frame, considered for the track initiation. $blip_j(t)$ is the j-th blip in the data frame, corresponding to the t-th data frame.

6.2 Preprocessing

Out of the $O(N^2)$ possible candidate tracks for each missile, many can be discarded by conventional preprocessing of the data. In the present implementation, the preprocessor makes use of a windowing technique. A spatial window is constructed around every blip in the initial frame and all the blips in the later time frames, falling within the window are considered. All possible tracks generated from these blips are considered to be candidate tracks for that particular blip in the initial frame.



String ((1 2) (2 1) (3 1)) corresponds to the track:
11'2'', 22'2'' and 33'3''

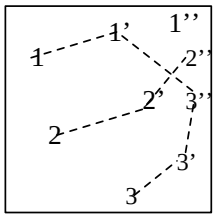


Figure 6.1: Coding scheme.

6.3 Simulation Result

6.3.1 Experiments on unclustered missiles

A simulation is run with 100 moving targets. Initial locations of the targets are randomly generated and each of them is assigned with random initial velocity and acceleration. Intensity of every target has a constant and a fluctuating component, which makes the problem more difficult. Imaging data from the first 3 frames are used for initiating tracks. Figure 3 shows the actual tracks of the targets in the first 3 data frames, with 100 blips in each frame. There are a total of $(100)^3$ or 10^6 different tracks possible and a total of $(100!)^2$ or $8.7(10^{315})$ different solutions (valid or invalid) possible. After preprocessing on the basis of the spatial windowing technique, the search space is reduced to about $3.5(10^{59})$ alternatives.

Figure 4 shows the tracks corresponding to the best string in the initial random population. Notice that this solution has ignored a number of blips from the second and third frame and instead included a number of tracks with shared blips. Figure 5 shows the tracks corresponding to the best string in generation 25. The solution is the global best and only required a total of 50,689 function evaluations, roughly $1.5(10^{-55})$ of the search space. Figure 6.5 summarizes the important parameters of the simulation and the results.

Track continuation may be achieved in at most $O(N)$ evaluations. For each successive frame at time t, preprocessing similar to that adopted here may be performed for correlating the blips in the previous frame at time $t-1$. Messy GA or some local search technique may be used to follow up the preprocessing in order to find the optimal track continuations.

6.3.2 Clustered missile simulations

In the previous subsection we presented the encouraging performance of messy GA tracker for 100 randomly dispersed missiles. In order to test the performance further, we made testbed even harder. This time the missiles were initialized in such a way that they form several spatial clusters. The light intensity of the missiles have random fluctuation as before. The performance of the messy GA tracker for 150 and 300 clustered missiles problems are as follows:

- **150 clustered missile problems:** Figure 6.6 shows the parameters of simulation. All the missiles are launched in such a way that they are distributed among five spatially different clusters. Our messy GA tracker has been able to find out correct tracks for 149

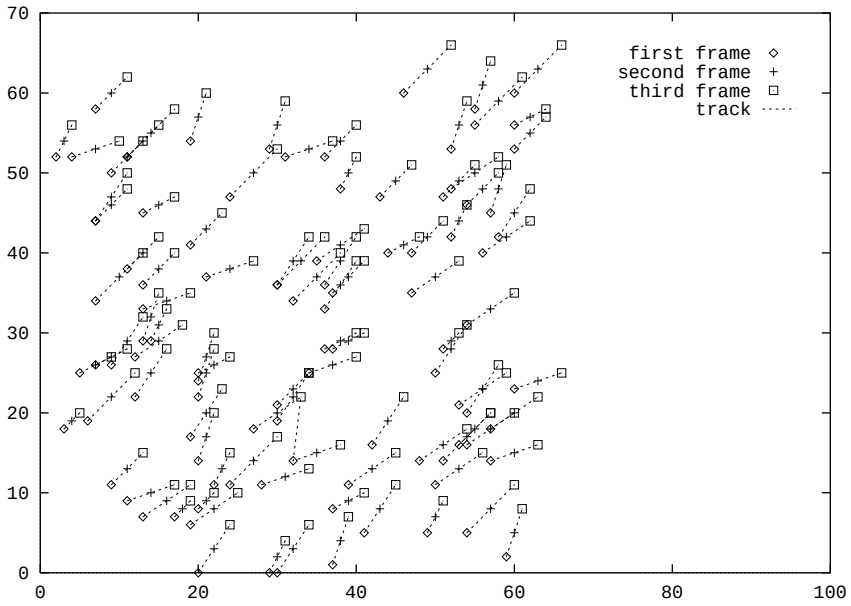


Figure 6.2: Blip locations in 3 data frames.

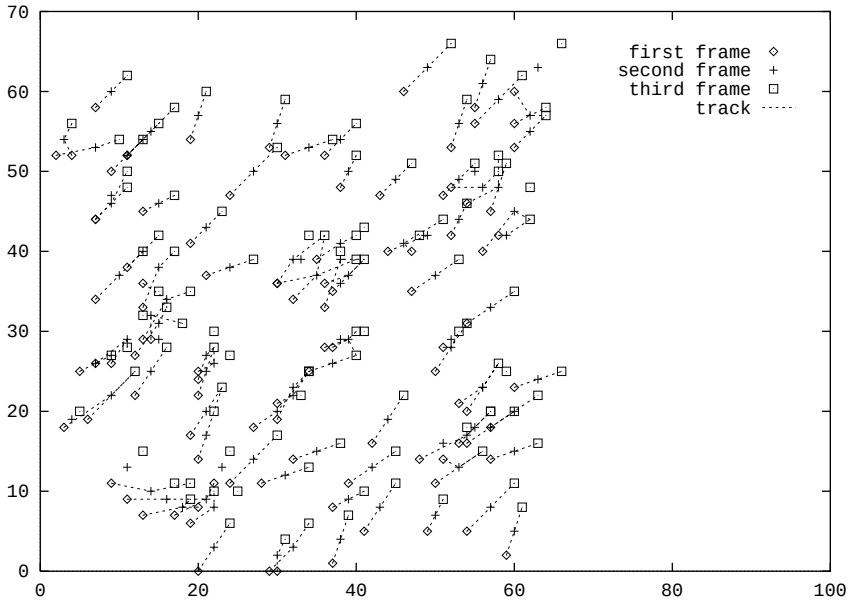


Figure 6.3: Tracks found by fmGA at generation 0

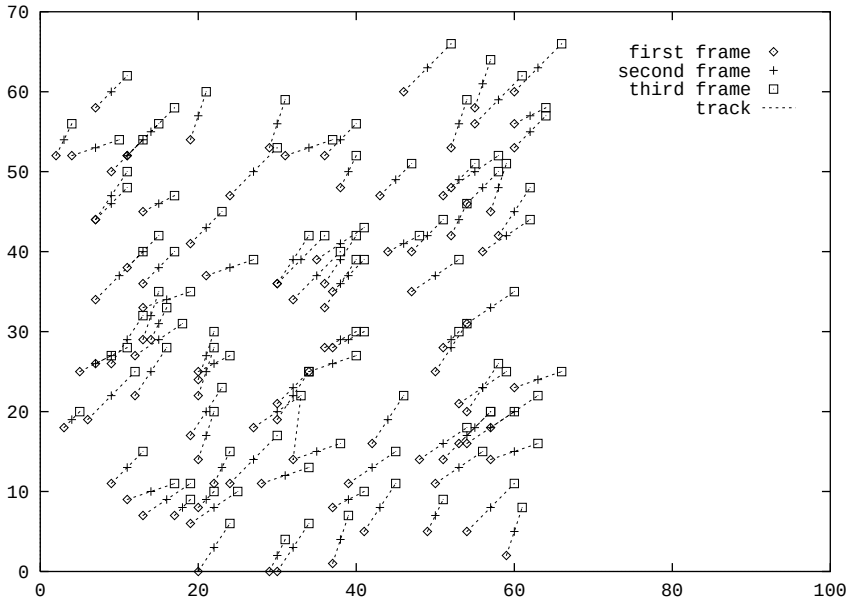


Figure 6.4: Tracks found by fmGA at generation 25

Simulation features:
number of targets = 100
varying intensity
$\alpha = \beta = 1.0$
$\gamma = 2.0$

Messy GA parameters:
random template
population size = 4224
splice probability = 1.0
cut probability = 0.02

Results:
- number of mistakes = 0
- function evaluations = 50689

Figure 6.5: Important particulars about the messy GA tracker and simulation for 100 unclustered missiles problem.

Simulation features:
number of targets = 150
varying intensity
clustered problem
$\alpha = \beta = 1.0$
$\gamma = 2.0$

Messy GA parameters:
random template
population size = 2432
splice probability = 1.0
cut probability = 0.02

Results:
- number of mistakes = 1
- function evaluations = 102140

Figure 6.6: Important particulars about the messy GA tracker and simulation for 150 clustered missiles problem.

missiles out of 150 missiles even for a modest population size of 2432. Figure 6.7 shows the variation of quality of solution along generations.

- **300 clustered missile problems:** In this simulation we used 300 clustered missiles, keeping everything else as before. Figure 6.8 shows the relevant parameters for the simulation. Figure 6.9 shows the correct tracks of the 300 missiles. This is a really difficult problem, considering the fact that the missile trajectories are so closely spaced. Messy GA tracker found right tracks for 295 missiles out of 300 with a population size of around 2900. The variation of the quality of solution along generations is given in 6.10.

6.4 Conclusions

In this section, I have presented the results of applying fmGA for solving a target tracking problem—a problem of high interest in the field of air defense systems. On a hundred target-

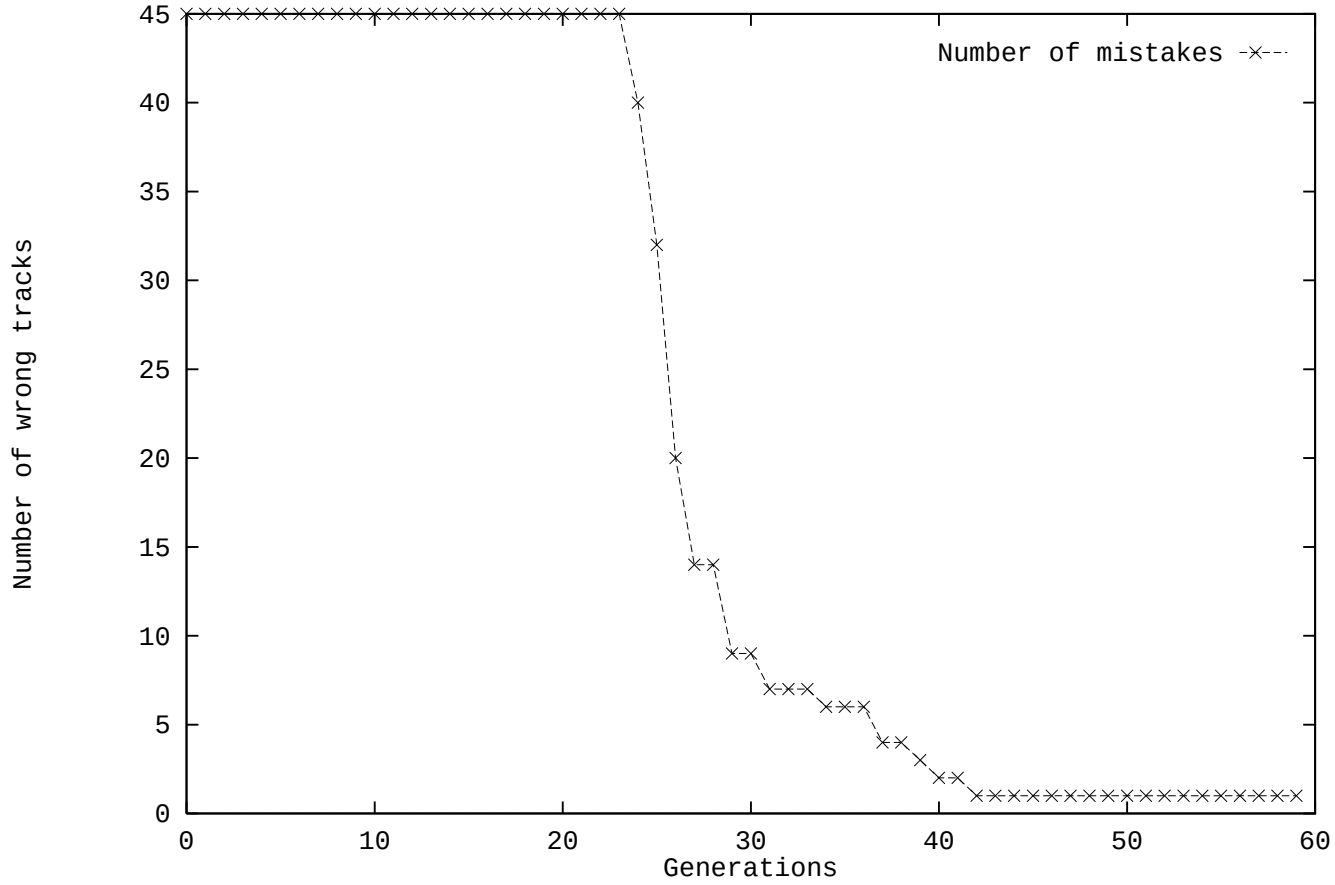


Figure 6.7: Variation of number of wrong tracks along generations for 150 clustered missiles problem.

Simulation features:
number of targets = 300
varying intensity
clustered problem
$\alpha = \beta = 1.0$
$\gamma = 2.0$

Messy GA parameters:
random template
population size = 2919
splice probability = 1.0
cut probability = 0.02

Results:
- number of mistakes = 5
- function evaluations = 105,080

Figure 6.8: Important particulars about the messy GA tracker and simulation for 300 clustered missiles problem.

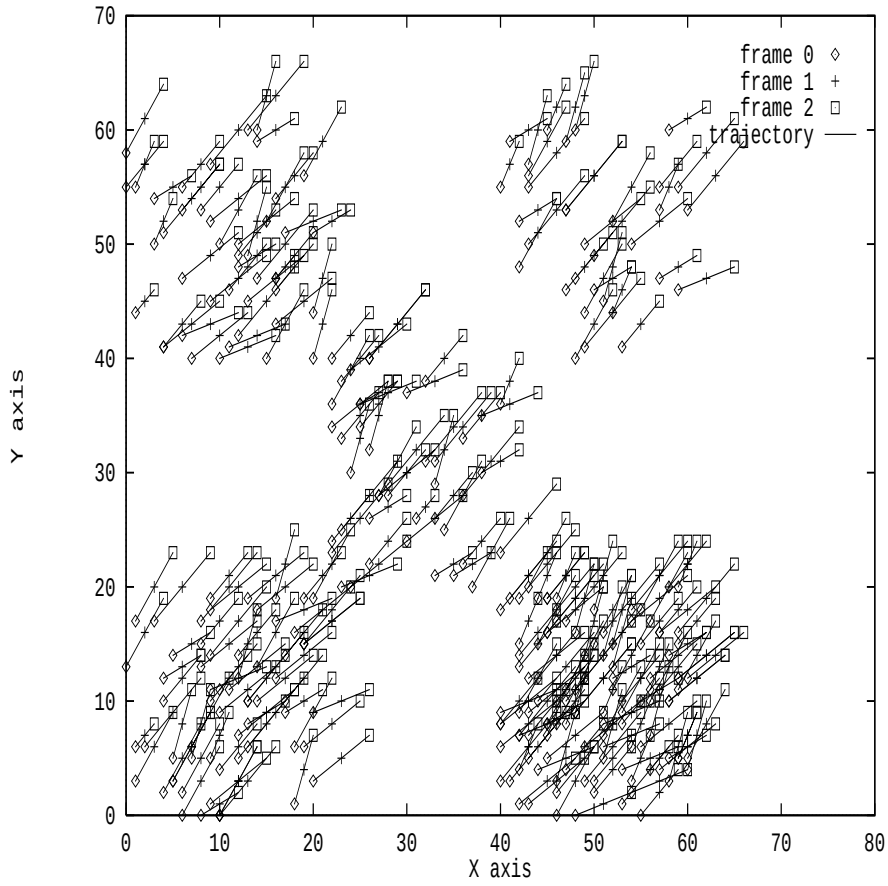


Figure 6.9: Three overlapped frames for the 300 missiles, with 5 clusters.

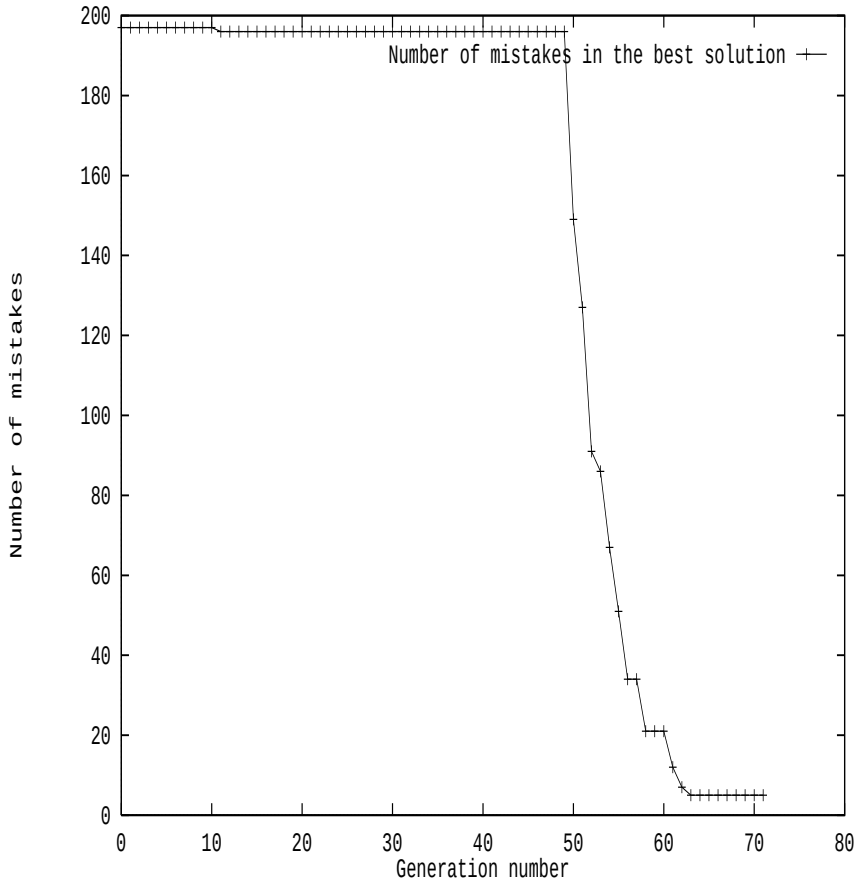


Figure 6.10: Variation of number of wrong tracks along generations for 300 clustered missiles problem.

tracking problem, the fmGA has found all tracks correctly after a reasonable amount of evaluations. This application encourages the use of fmGA in more difficult tracking problems.

Chapter 7

Conclusions and Future Work

This chapter discusses the main conclusions of this dissertation and identifies future directions of research. The following section presents the main conclusions.

7.1 Conclusions

This dissertation has dealt with two major aspects of blackbox optimization:

1. understanding the fundamental principles that transcend a blackbox search beyond random enumeration and developing a general framework to quantify them
2. designing and testing blackbox optimization algorithms that share some of these fundamental principles

The achievements of this dissertation along each of these dimensions are discussed in the following sections.

7.1.1 SEARCH and its implications

The SEARCH framework decoupled BBO into three spaces, namely the relation, the class, and the sample spaces. It realized the importance of the decision makings in both relation and class spaces. SEARCH establishes that the role of relations in BBO is essential to surpass the limits of random enumerative search in both qualitative and quantitative manner. The appropriateness of relations in solving a BBO is quantified in terms of the delineation constraint. The quantitative analysis of the decision processes in SEARCH provided bounds on success

probability and sample complexity. I considered different blackbox algorithms and showed that they can all be viewed from the SEARCH perspective. I also presented natural evolution in the light of SEARCH. Some of the major computational aspects of natural evolution are studied using the basic principles of SEARCH. This led us to an alternate perspective of evolution that establishes the computational role of gene expression in evolution.

SEARCH provides a unified approach toward understanding the role of different participants in solving BBO problems: (1) algorithm, (2) problem, and (3) user. It identifies the essential components of a blackbox optimization algorithm. The bounds on sample complexity and success probability in SEARCH are used to quantify problem difficulty in BBO. This led to the identification of one particular class of BBO problems that can be solved in polynomial sample complexity—the class of order-k delineable problems. Parallel evaluation of relations from the same set of samples can be accomplished by exploiting the structural properties of the set of relations considered by the algorithm. This observation is used to quantify so-called *implicit parallelism* (Holland, 1975). SEARCH also pointed out the potential computational benefits from considering low-order relations first in the course of search. This observation may find some rationale behind the widely observed bottom-up process of pattern formation in natural and artificial complex systems. An algorithm in SEARCH requires the user to define the relation space. This appears to be one of the most critical responsibilities of the user. Several other principles for designing the search operators are also noted.

Before I move on to the accomplishments of the second part of this thesis, let me list the main findings of this part of the thesis and explain their implications:

1. It is possible to view blackbox optimization from a common unified perspective. The SEARCH framework offers one possible way to do that in an ordinal, probabilistic, and approximate sense.

2. Unless we assume relations among the members of the search space, blackbox search cannot be any better than random search. Therefore the role of relations in BBO is fundamental.

3. The appropriateness of relations in solving a BBO problem can be quantified in terms of delineation, and it can be taken into account while deriving bounds on success probability and sample complexity in BBO.

4. Relations can be evaluated in parallel from a common set of samples by exploiting some structural properties of the set of relations provided to the algorithm. This observation can be used to quantify *implicit parallelism*.

5. Explicit bounds on success probability and sample complexity in BBO are derived.

6. Problem difficulty in BBO can be rigorously quantified using the SEARCH framework.

7. The class of order-k delineable problems can be solved with sample complexity growing polynomially along ℓ, q, q_r, $1/d^*$, and $1/d_r^*$.

8. SEARCH offers an alternate perspective of natural evolution that establishes the computational role of gene expression in evolution.

Each of these is discussed in somewhat more detail in the following paragraphs.

The continued arrival of more new blackbox optimization algorithms has caused the need for a common, unified perspective toward understanding BBO. The SEARCH framework developed in this thesis took a step toward this goal by presenting an alternate perspective toward BBO in terms of relations, classes and ordering. The foundation of SEARCH is laid on a decomposition of BBO into the following items:

- searching for relations
- sampling
- searching for better classes defined by relations
- resolution, which exploits the delineation property of relations

This dissertation proposes that these are the fundamental processes in any probabilistic, adaptive sampling optimization algorithm. This proposition is also supported by explicitly considering different popular algorithms from the SEARCH perspective.

The performance of a BBO algorithm will be no better than random search when no relations are considered among the members of the search space. Every algorithm that aspires to transcend this limit has to consider some relations defined over the search space. Unfortunately, very few existing algorithms appear to realize the critical role of relations in blackbox search.

Most of the existing algorithms such as simulated annealing and simple genetic algorithms define relations implicitly and combine the fundamentally different decision making in relation and class spaces together. This leads to undesirable decision errors. Another problem with this implicit definition of relations is that the user hardly has any idea about the relations processed by the algorithms. Different kinds of relations may be envisioned among the members of the search space. For example, traditionally, the genetic algorithm has been viewed to pay attention to the relations defined by the representation. However, relations can also be defined in terms of search operators like crossover and mutation. Therefore, exact identification of relation space for the above-mentioned algorithms depends on the discretion of the individual. Since defining relations that properly delineate the search space is critical for success, and since often in practice the user may be able to help in defining the appropriate relation space, it is imperative that the algorithms explicitly define the relation space. The main conclusion is that regardless of someone's point of view about the dynamics of a search algorithm in the optimization domain, emphasis on relation processing is essential, if performance of the algorithm is an issue.

Not all the relations are appropriate for a given problem and algorithm. This dissertation provided a way to quantify the appropriateness of relations in terms of the delineation requirement. Relations can be defined in many ways, and representation is not the only way to do that. However, representation may be a useful way to define problem-specific relations. It is the primary responsibility of the user to define this space, since same set of relations may or may not be appropriate for different problems. The delineation-ratio—the ratio of the number of relations that properly delineate and the total number of relations in the relation space—reflects how suitable the relation space is. An algorithm is likely to fail if this ratio is very low. The bounds on success probability and sample complexity in SEARCH directly depends on delineation property of relations.

The SEARCH perspective points out an interesting possibility. Relations divide the same search space in a different manner. Therefore, relations can be evaluated in parallel using a common sample set. This observation is used to quantify the so-called implicit parallelism noted earlier by Holland (1975). This dissertation computed the bound on sample complexity for sequence representation and also noted the computational benefits of implicit parallelism for this particular representation.

SEARCH provides us a closed-form bound on the sample complexity for solving a BBO in terms of the number of relations considered to solve the problem, their indices, the desired quality of solution and relations, success probabilities in detecting an appropriate relation, and the class containing the optimal solution. The first two parameters often depend on the problem size. The SEARCH framework decouples problem difficulty along these dimensions:

1. problem size
2. success probability in making the right decision to choose a relation that properly delineates the search space
3. success probability in selecting the class containing the optimal solution
4. desired quality of relations and the solution

It is important to realize that this only presents a higher-level picture that directly interprets different aspects of problem and algorithm to the success. Different specific characteristics such as multimodality (Horn & Goldberg, 1995), crosstalk (Goldberg, Deb, & Thierens, 1993; Kargupta, 1995b) of the problem and algorithm can be identified within each of these aspects.

The definition of problem difficulty and the bound on sample complexity in SEARCH can be directly used to conclude that the class of order k delineable problems can be solved in polynomial sample complexity. The design and testing of fast messy GAs presented in the second part of this thesis were inspired to solve this particular class of problems.

SEARCH also provides insights in natural evolution. This dissertation briefly reviewed a possible mapping of the main characteristics of SEARCH into natural evolution developed in (Kargupta, 1995b). Understanding evolution in the light of SEARCH requires paying attention to the intracellular flow of genetic information—gene expression. An alternate computational model of evolution is proposed that extends the traditional model of evolution (Holland, 1975) by incorporating steps of gene expression. A direct correspondence between different components of gene expression process and SEARCH is hypothesized. The transciptional regulatory mechanisms are viewed as the relation space in evolution. This space also makes use of information from the sample space—the DNA. This work clearly showed that the amino acid sequences in proteins define equivalence classes over the DNA space. Therefore, nature appears to have a

distinct class space. Kargupta (1995b) also noted that in eukaryotes, construction of the new relation is possible because of the diploid chromosome.

The following section considers the achievements on the second ground of objectives of this dissertation.

7.1.2 Design and testing of the fast messy genetic algorithms

The second qualitatively distinct part of this thesis is the design and test of fast messy GAs initiated in Goldberg, Deb, Kargupta, and Harik (1993). Messy GAs (Deb, 1991; Goldberg, Korb, & Deb, 1989) are a rare class of BBO algorithms that emphasize on the search for proper relations that classify the search space in an appropriate manner. The original version of messy GA (Deb, 1991; Goldberg, Korb, & Deb, 1989) completely lacked the benefits of implicit parallelism. The fast messy GA makes use of a probabilistically complete initialization and a building-block filtering process to gain some of the advantages of implicit parallelism. The version of fmGA presented in Goldberg, Deb, Kargupta, and Harik (1993) appeared to have some degree of sensitivity toward the choice of particular filtering schedule. A modified technique for thresholding is introduced in this dissertation that appears to be able to maintain the building-blocks more appropriately. Use of this modified scheme and iterative application of fmGA minimized this problem. The fast messy GA has been tested on different kinds of instances of the class of order-k delineable problems. Although the fmGA pays the price of search for relations in terms of more function evaluations compared to algorithms like simple GA on average, fmGA is likely to solve problems in which little knowledge about the good relations defined by the representation is available.

The main conclusions of this part of the thesis are listed in the following:

1. The scope of the simple genetic algorithm is limited, since it does not properly search for appropriate relations. The overall decision process is very noisy, since relation, class, and sample spaces are all combined into a single population.

2. Messy genetic algorithms emphasize searching for the right relations. However, the original messy GA fail to exploit the benefits of implicit parallelism.

3. The fast messy genetic algorithm eliminated the enumerative initialization process of the messy GA by introducing a probabilistically complete initialization and a building-block filtering process.

4. The fast messy GA is likely to solve instances of the class of order-k delineable problems in polynomial sample complexity.

5. The fmGA is also tested against problems with crosstalk—royal road functions. Thresholding selection in fmGA restricts cross-competition among strings from different partitions. This seems to be effective in solving problems with crosstalk, like the Royal Road functions.

6. Like the original messy GA, the fmGA implictly defines the relation and class spaces together. Relation comparison is accomplished by comparing classes from different relations. This is inappropriate and may lead to cross-competition among classes from good relations.

Each of these is discussed in somewhat more detail in the following paragraphs.

Simple GA fails to properly search for relations that properly delineate the search space. The one point crossover highly disrupts the classes defined over loci that are not close to one another. In other words, it assumes that the relations defined over closer loci are better with no apparent rationale. Other crossovers like uniform crossover also do not work (Thierens & Goldberg, 1993). One of the fundamental problems is that relation, class, and sample spaces are all combined together, and as a result decision making in one space effects the others. Implicit parallelism was an interesting concept noted by Holland (1975). However, the computational benefit of implicit parallelism is outweighed by the error in decision making introduced by the lack of explicit evaluations of relations.

Messy GAs are one among the rare class of algorithms that emphasize searching for appropriate relations. Messy GAs use a competitive template and explicit enumeration of good classes—building-blocks—to ensure correct decision making. However, explicit enumeration of building-blocks essentially means a complete lack of the benefits of implicit parallelism.

The fast messy GA eliminated one major bottleneck of the messy GA—the enumerative initialization. The probabilistically complete initialization and the building-block filtering pro-

cess can be used to detect better classes from better relations without sparing the advantage of implicit parallelism completely.

Experimental results suggest that fmGA can be used to solve the class of order-k delineable problems in polynomial sample complexity. The fmGA has been tested on different kinds of deceptive problems with uniform scaling, non-uniform scaling, and with building-blocks of mixed sizes.

Kargupta (1995b) also showed that royal road function R2 offers crosstalk problem in simple GAs. I briefly reviewed the definition of crosstalk presented in Kargupta (1995b). The fmGA is also tested against royal road problems R1, R2, and R3. The fast messy GA seems to be effective in solving problems with crosstalk. Thresholding selection that restricts comparing classes from different partitions seems to contribute this feature.

One possible source of problem in messy GAs is the thresholding selection. The thresholding selection tries to find the better classes and also the better relations. Thresholding parameter is always chosen less than the string length for allowing the elimination of bad relations. However, since the relations and classes are implicitly defined in the string population, construction of ordering among relations influence that among the classes. Although the idea is to eliminate bad relations, physically, this means comparing classes from different relations, which is inappropriate. The undesirable consequence of this is that it allows competition among classes from good relations as well. This leads to cross-competition that may ultimately eliminate some building-blocks.

7.2 Ramifications

There are several possible ramifications of this work. They are listed in the following:

1. Explore the strengths and weaknesses of explicit consideration of relation, class, and sample spaces in a BBO algorithm.

2. Consideration of parametric analysis when relation or class evaluations are not independent because of the implicit definition of the spaces.

3. Study different class and relation comparison statistics.

4. Design new algorithms in a more constructive manner, by designing new instances of the components of the laundry list that defines an algorithm in SEARCH.

5. Find classes of BBO problems, other than the order-k delineable problems, that can be solved in polynomial complexity.

6. Explore the possibility of new relation construction.

7. SEARCH and evolution—Can it lead to a biologically plausible implementation of the main lesson from SEARCH?

8. Test the fast messy GA against noisy functions.

9. Eliminate the problem with thresholding selection in the fmGA by either designing it to solve a bounded number of building-blocks or by explicitly defining the relation and class spaces.

The following paragraphs present a brief account of them.

The foundation of SEARCH is laid on a decomposition of the search space into relation, class, and sample spaces. The analysis showed that the decision makings in each of these spaces is different from the others. We also saw some plausible evidence supporting the hypothesis that natural evolution may have such explicit decomposition of the search space. The immediate extension of this work is to explore the strengths and weaknesses of such explicit decomposition.

The SEARCH framework took a distribution-free ordinal approach to design class and relation comparison statistics. The comparison process for each pair in either the relations or class space is considered independent of other pairwise comparison process. Although this is true in SEARCH, it may not be true in algorithms like GAs. Since all three spaces are combined together, evaluations are not independent. When they are not not independent, a parametric approach needs to be taken. Kargupta (1995b) took a Bayesian approach to analyze decision error for such cases. One possible ramification is to introduce parametric analysis in SEARCH in order to have a better idea about the decision making for specific algorithms, that do not have explicit separation of spaces.

Although almost all the blackbox search algorithms make use of some kind of class and relation comparison statistic, the utilities of such statistics need to be constructively studied. We

need to know what kinds of statistics are "generally" suitable for relation and class comparisons, if such generalization is possible at all.

This work also outlined the main components that an algorithm needs in order to transcend the limits of random enumerative search. Now that we have a listing of the different components, development of new algorithms should be more constructive. Design of a new algorithm is now reduced to design of new instances of one or more items from this laundry list. Designing a new class or relation comparison statistic, designing a new ordering construction algorithm in either of these spaces, and defining a new order to consider relations are some examples.

The class of order-k delineable problems has been identified as solvable in polynomial sample complexity. Other such classes of problems need to identified.

Most of the blackbox optimization algorithms rely on the user-provided source of relations, such as representation, operators. SEARCH also analyze BBO when such a source of relation exists. An immediate extension is to explore the possibility of using new relation construction. When the set of relations Ψ_r does not have enough relations that properly delineate the search space, BBO is destined to fail. When we do not know this beforehand, the search will be executed anyway. However, the question is whether or not we can use the information gathered from this bad set of relations to construct a new set of relations that is more appropriate for the problem. One possible way to approach this may be to solve assuming some order-k delineability and then use that information to construct a new source of relations and observe the improvement of performance if any.

This dissertation pointed out a possible mapping between SEARCH and natural evolution. The proposed computational model of evolution, which realizes the importance of intra-cellular flow of information, offers some interesting possibilities. While most of the existing models of evolutionary computation focus on debating the relative importance of mutation and crossover, the SEARCH perspective points out that there is another important aspect of evolution that is almost unexplored. SEARCH proposes a computational model that hypothesizes the explicit decomposition of relation, class, and sample spaces in natural evolution. One possible extension is to implement the lessons of SEARCH in a biologically plausible manner. Precise evaluation of equivalence classes using an operator like transcription is possible. Construction of new relations using a diploid chromosome and a transcription operator can also be incorporated in a GA. Incorporating the main principles of SEARCH in a biologically plausible manner

is indeed an immediate possibility. A new class of algorithms, named *gene expression messy GA*, is currently in the stage of development and testing; it uses operators like transcription to accomplish both relation construction and precise evaluations of equivalence classes. More work needs to be done before presenting it to the users.

This dissertation also presented the design and testing of the fast messy GA (fmGA), introduced earlier in (Goldberg, Deb, Kargupta, & Harik, 1993). There are several possible extensions to this work. Along with others, the test results on problems with crosstalk are reported. By definition, crosstalk introduces noise in the decision making process that can be quantified by the covariance measure described earlier. However, the decision making process can be made noisy by several means. Using a noisy objective function is one possibility that was not included in the chosen test suite. The fast messy GA should be tested on such problems.

Thresholding selection introduces cross-competition among building-blocks. As I noted earlier, the problem originates from the effort to do relation comparison by explicit class comparison. The thresholding parameter is relaxed to allow the elimination of bad relations. However, this also introduces class-comparison from different good relations. Such cross-competition may result in elimination of some good building-blocks. There appear to be at least two possible extensions of the fmGA:

- Derive or estimate the bounds on the effect of such cross-competition, and then present the fmGA as an algorithm that solves a bounded number of building-blocks at a time.
- Explicitly define the relation and class spaces. In that case, relation comparison is completely separate from class comparison, and the problem of thresholding will not exist anymore. Note that this explicit decomposition does not necessarily has to be a spatial decomposition. It may be possible to decompose these spaces temporally.

These two approaches define the future directions of the messy GAs.

Appendix A

Relations, Orderings, and Computational Complexity: A Brief Review

Development of the SEARCH framework requires familiarity with some basic set theory. This appendix reviews these definitions. I start from the basic definition of a set, and proceed toward defining relations and orderings. I also review some definitions of bounds in computational complexity.

A.0.1 Set

A *set* is a collection of distinguishable objects, called its *members*. If an object x is a member of a set S, we write $x \in S$. A one-element set is called a *singleton*. Given two sets S_1 and S_2 such that for all $\forall x \in S_1$, we know that $x \in S_2$, then we call S_1 a *subset* of S_2. It is denoted by $S_1 \subset S_2$. The set of all subsets of S is called the *power set* of S and it is denoted by 2^S.

A.0.2 Relation

The SEARCH framework explores the possibility of defining relations among the members of the domain of optimization. Therefore, it is important to understand what relations are, and this section presents the set theoretic definitions of relation and its properties.

A *relation* is a set of ordered pairs. A binary relation r on a set S is a subset of the Cartesian product $S \times S$. For example, the *less than or equal to* relation on integer space is $\{x, y \in \mathbf{N} : x \leq y\}$, where $\mathbf{N}$ is the set of natural numbers. A relation r is *reflexive* if $x\ r\ x$, $\forall x \in S$. A relation r is *symmetric* if $x\ r\ y$ implies $y\ r\ x$, $\forall x, y \in S$. A relation r is *transitive* if $x\ r\ y$ and $y\ r\ z$ imply $x\ r\ z$. An *equivalence relation* is defined as a relation that is reflexive, symmetric, and transitive. For example, "=" is an equivalence relation on the natural numbers. If r is an equivalence relation on S then for $x \in S$, the equivalence class of x is $\{y \in S : x\ r\ y\}$. If $S = \bigcup_i s_i$, where every set s_i is not empty and every pair s_i and s_j are disjoint if $i \neq j$, then we say that the sets s_i partition S. If r is an equivalence relation on S, then the distinct equivalence classes of r *partition* S. The number of distinct equivalence classes of r is called its *index*. A relation r is *antisymmetric* if $x\ r\ y$ and $y\ r\ x$ imply $x = y$.

A.0.3 Partial order

An order is a particular kind of relation. For example, when the set of individuals, each with a distinctly different age, are ranked based on their age, we call this ranking an order. A partial order is again a special kind of order. For example, if the above set is now ranked based on the relation *is-ancester-of* then all the pairs in this set cannot be ordered using this relation. This kind of relations, in which every pair cannot be ordered, is called a partial order. The SEARCH framework makes use of partial orders to partially rank different regions of the search space. Therefore, this is again an important concept that we shall use frequently in this thesis.

A partial order can be defined using the set theoretic defintions introduced earlier. A relation that is antisymmetric and transitive is a *partial order*, and the set on which a partial order is defined is called a *partially ordered set*. For example, the relation *is-descendent-of* is a partial order on the set of all males. A partial order r on S is called a *total order* if $\forall x, y \in S$ we have either $x\ r\ y$ or $y\ r\ x$.

A.0.4 Bounds in complexity

The order of increase in the cost of running an algorithm characterizes its eficiency. In blackbox optimization the number of samples taken from the search space is a measure of the cost of running an algorithm. The SEARCH framework considers this measure of cost for solving BBO

problems. Although, there exists several ways to bound the growth of cost (Cormen, Leiserson, & Rivest, 1990) I only consider the *asymptotic upper bound*.

The *asymptotic upper bound* of a given function $f(\ell)$ is the set of functions, in which for any member $g(\ell)$ there exists constants c and ℓ_0 such that $0 \leq g(\ell) \leq cf(\ell)$ for all $\ell \geq l_0$. This upper bound is denoted by $O(f(\ell))$. A function $f(\ell)$ is polynomially bounded if $f(\ell) = O(\ell^k)$ for some constant k. On the other hand, $f(\ell)$ is exponential if $f(\ell) = O(a^\ell)$, for some $a \neq 0$. Further details about these definitions can be found elsewhere (Cormen, Leiserson, and Rivest, 1990).

Appendix B

Simple GA: A Brief Review

Genetic Algorithms (GAs) (Holland, 1975) are a class of stochastic search algorithms. They are motivated by the computational process in natural evolution. GAs emphasize the role of representation in search. The simple GA (De Jong, 1975; Goldberg, 1989; Holland, 1975) works from a population of samples defined using some representation and searches by *selection*, *crossover*, and *mutation* operators.

1. **Representation:** Simple GA sometimes uses a sequence representation. Binary representation and gray coding are some examples often used in GAs. A string usually represents a unique member of the search space. Strings are sometimes called *chromosomes*. Every locus of a string is also called *gene*; the corresponding value at a locus of a string is called the allele value of the gene.

2. **Selection:** The selection operator is responsible for detecting better regions of the search space. The "fitness' " of a member is its objective function value. Selection computes an ordering among all the members of the population and gives more copies to the better strings at the expense of less "fit" members. There exist various kinds of different tools of selection operators (Goldberg & Deb, 1991b). Some widely used selection operators are roulette wheel selection (Holland, 1975), tournament selection (Brindle, 1981; Goldberg, Deb, & Korb, 1990a) and stochastic remainder selection (Booker, 1982; Brindle, 1981). Although these selection operators are technically different from each other, all of them share the same feature giving more copies to the better strings. The amount of selection pressure given to the population varies among these different selection operators. However,

```
/* Initialization */
t = 0; // Set the generation number to zero.
Initialize(Pop(t)); // Initialize the population at random
Evaluate(Pop(t)); // Evaluate the fitness values
Repeat
{
    Selection(Pop(t)); // Select better strings
    Crossover(Pop(t)); // Cross better strings to produce offspring
    Mutation(Pop(t)); // Mutate strings
    Evaluate(Pop(t)); // Evaluate fitness
    t = t + 1; // Increment generation counter
}
Until ( t > t_max Or (termination criterion TRUE) )
```

Figure B.1: A pseudo-code for simple GA.

in absence of any other operators, all of them lead the population to convergence in polynomial time (Goldberg & Deb, 1991b; Mühlenbein, 992b).

3. **Perturbation operators:**

   - **Crossover :** Crossover works by swapping portions between two strings. Single point crossover (Holland, 1975), is often used in simple GA. It works by first randomly picking a point between 0 and ℓ. The participating strings are then split at that point, followed by a swapping of the split halves. The working of one point crossover is illustrated in figure B.2(left). Crossover has interesting search behavior. This leads to many investigations that resulted in different perspective of crossover (Booker, 1993; Culberson, 1994). Different kinds of crossovers have been suggested in the literature (Goldberg, 1989; Syswerda, 1989). Crossover is often applied with a high probability.
   - **Mutation:** Mutation randomly changes the entries of a string. Mutation is usually treated as a low profile operator in GA because of its random nature of perturbation. Figure B.2(right) shows an example of a point mutation operation.

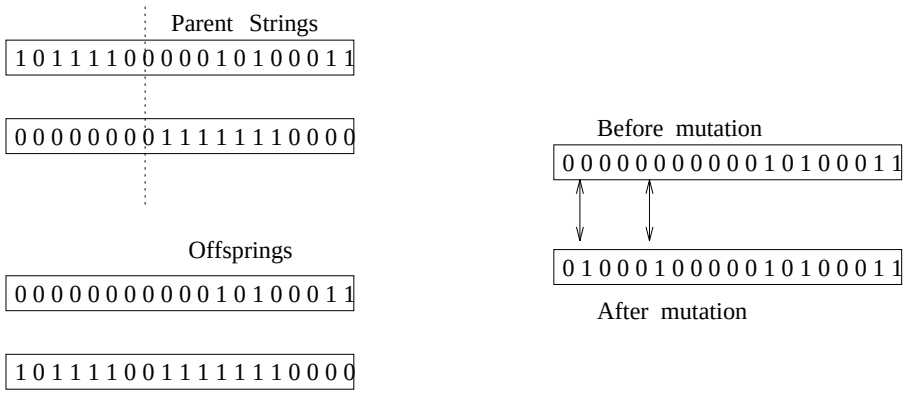


Figure B.2: (left)Single point crossover. (right) Point mutation for binary strings.

Appendix C

An Analysis of the Thresholding

A design procedure for choosing the parameters of the building-block (BB) flitering process can be developed by addressing the following issues:

1. tournament selection in presence of cross-competition;
2. random deletion of genes;
3. choosing the thresholding parameter.

Each of these is discussed in detail in the following sections. At the end, the overall lessons are summarized.

C.1 Selection in the Presence of Cross-Competition

The objective of this section is to develop a simple model of cross-competition. This model will help us choosing the number of times selection is applied before gene deletion.

In the fast messy GA (fmGA) the building-block filtering phase is divided into several stages. Selection is applied for few generations, then genes are deleted randomly. This process is repeated several times until the string length is in the order of the size of the building blocks (BBs). Each of these iterations corresponding to a different string length will be called an episode. Thresholding divides the population into several overlapping subsets, each corresponding to a unique partition. Each of these subsets will be called a niche. These niches are overlapping because the thresholding parameter is often less than the string length. To

determine how long selection should be applied in a particular episode we need to know the following things:

1. the initial distribution of strings within the niche;

2. how strings grow or become extinct because of selective pressure.

Determining the exact distribution of strings within a niche is hardly possible in absence of knowledge about the problem itself. Since we cannot afford that luxury, we consider a simplified model. In this model there are the following kinds of strings in a niche:

1. strings containing a building-block from a particular good partition; let us denote this partition by r_i and the group of strings containing the building-blocks in r_i will be represented by the subscript i.

2. strings containing a building-block from all other partitions except the partition r_i; I assume that building-blocks from m partitions are needed to solve the problem. The set of strings corresponding to each of these partitions except r_i will be denoted by the subscript j. Therefore j can take $m-1$ different possible values.

3. strings that contain no building blocks; these are the strings that should be eliminated during the course of building-block filtering stage.

The proportion of building-blocks in a population will be denoted by q_i and q_j with the subscripts chosen according the previous description. String containing building-blocks from different partitions will compete with each other as long as they belong to the same thresholding niche. The nature of the cross-competition among these building-blocks depends on their contribution to the objective function value of the corresponding solution. Let us define an *interaction matrix* α, to introduce the idea of cross-competition. α_{ij} the $(i,j)-th$ element of α is the expected number of copies a BB from partition i makes by competing with a BB defined over partition j. Every selection only generation within each episode is denoted by subscript t.

The growth equation for q_i can be written as,

$$q_{i,t+1} = q_{i,t}\left(2 - q_{i,t} - \sum_{j \neq i}(2-\alpha_{ij})q_{j,t}\right) \tag{C.1}$$

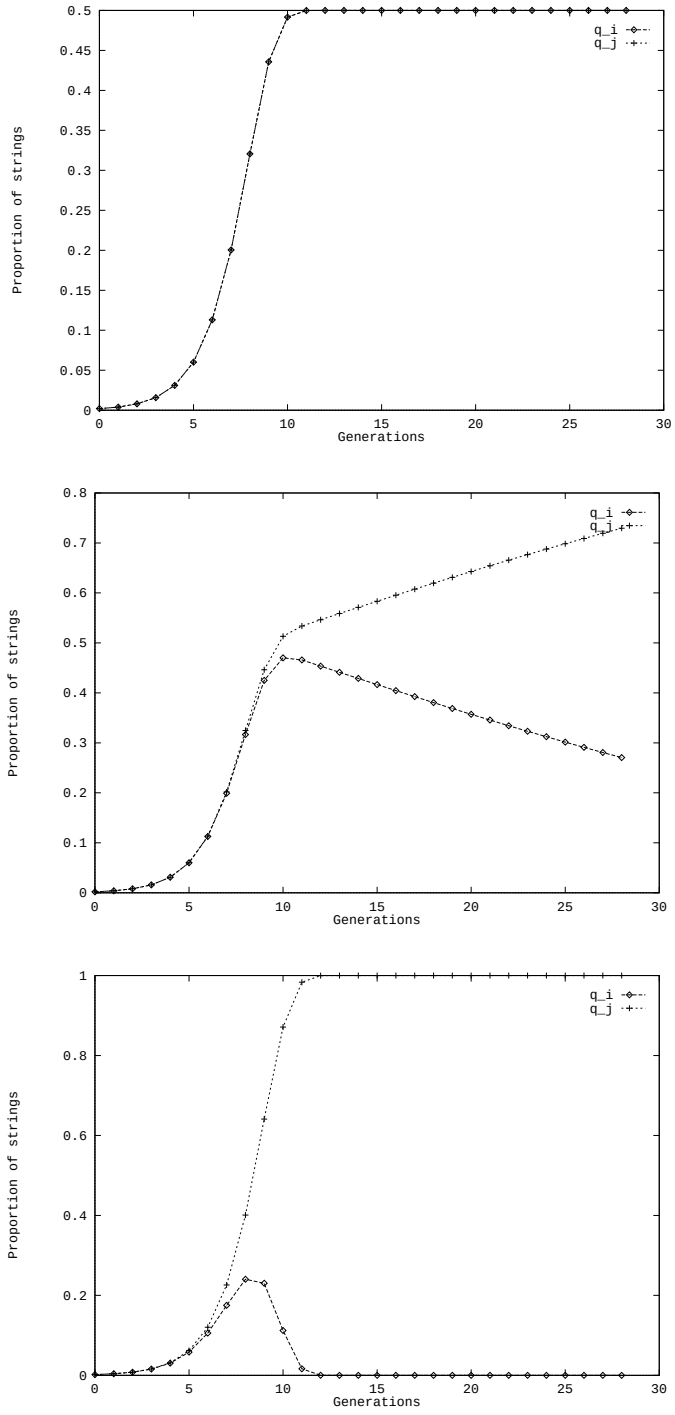


Figure C.1: Effect of selection in presence of cross-competition: α_{ij} is the $(i,j) - th$ element of the BB interaction matrix α gives the expected number of copies a BB defined over set i makes by competing with a BB defined over the another set of genes, j. **(Top)** $\alpha_{ij} = \alpha_{ji} = 1$. **(Middle)** $\alpha_{ij} = 0.95, \alpha_{ji} = 1.05$. **(Bottom)** $\alpha_{ij} = 0, \alpha_{ji} = 2.0$. As we see slight bias towards a particular BB can lead to quick extinction of the other BB after a certain stage.

where $q_{i,t}$ denote proportion of the BB from partition i within a niche at iteration t of an episode. This can be written in the differential form as,

$$\frac{\partial q_{i,t}}{\partial t} = q_{i,t}\left[1 - q_{i,t} - \sum_{j \neq i}(2 - \alpha_{ij})q_{j,t}\right] \tag{C.2}$$

There are m such differential equations, governing the growth of m good BBs.

The strings with no BBs are assumed to loose every competition with the strings with BBs. Therefore, the proportion of these junk strings,

$$\begin{aligned} q_{junk,t+1} &= q_{junk,t}^2 \\ &= \left(q_{junk,0}^2\right)^t \end{aligned} \tag{C.3}$$

In our simple model, equations C.1 and C.2 presents a simple picture of cross-competition. Let us now illustrate there physical interpretation. Consider a simple problem in which $m = 2$. For the purpose of illustration, let us assume that at the initial stage, $q_{i,0} = q_{j,0} = 0.001$. Figure C.1 shows the variation of $q_{ig,t}$ and $q_{jg,t}$ over time for different pair of values of α_{ij} and α_{ji}. The top figure shows the ideal case when $\alpha_{ij} = \alpha_{ji} = 1$. In this case, the competition is even and both the classes of strings continue to grow until their proportion converges to 0.5. Once the junk strings are eliminated, there is not really any selective pressure and therefore their proportion remains the same. The figure in the middle shows that if we perturb little bit by making $\alpha_{ij} = 0.95$ and $\alpha_{ji} = 1.05$ $q_{ig,t}$ eventually reduces down because of slight bias towards $q_{jg,t}$. The bottom most figure shows the extreme situation when $\alpha_{ij} = 0$ and $\alpha_{ji} = 2$. The two figures from the bottom shows that $q_{i,t}$ and $q_{j,t}$ practically remains same up to a certain stage and there after $q_{i,t}$ falls apart. These two figures show that when there is slight bias toward a certain BB, which could even be stochastic drift, after a certain stage one of the BBs takes over the other BB.

Our objective is to find the number of iterations of selection only phase, after which cross-competition starts playing a major role, resulting in eliminating some BBs. For a given interaction matrix, α, we should be able to solve the set of coupled differential equations at least numerically in the most general situation. However, such information is not likely to be available. Therefore, it will probably be more prudent to explore the extreme cases and let the

user choose a reasonable value within the bounds. The following subsections consider two such extreme cases.

C.1.1 Cross-competition without any bias toward a particular good BB

If all the good BBs cross-compete with each other without any bias towards a particular one, in the ideal situation, i.e. when the effect of drift is neglected $\alpha_{ij} = 1$ for every combination of i and j. Since both q_i and q_j-s grow identically, they can be treated as one variable, $q_{ig,t}$. All the growth equations for different good BBs become identical and we can combine them to form a differential equation,

$$\frac{\partial q_{ig,t}}{\partial t} = q_{ig,t}\left[1 - m q_{ig,t}\right] \tag{C.4}$$

which can be solved resulting,

$$q_{ig,t} = \left[m + \frac{1 - m q_{ig,t_0}}{q_{ig,t_0}} \exp^{-t}\right]^{-1}$$

$q_{ig,t}$ asymptotically converges to $1/m$; let us say a population is converged when $q_{ig,t} = \frac{\omega}{m}$, in order to solve for a finite convergence time. Finally we can solve for the convergence time,

$$t^* = \ln \frac{(1 - m q_{ig,t_0})\omega}{m q_{ig,t_0}(1 - \omega)} \tag{C.5}$$

where $1/\omega$ is a factor that weighs the degree of desired convergence to the maximum possible $q_{ig,\infty}$. This factor is basically used to get a practical value for t^* from the fundamentally asymptotic variation of $q_{ig,t}$.

C.1.2 Cross-competition in presence of strong bias

The model analyzed in this section tries to capture one of the worst case possibilities. In this extreme case, I assume that there exists strings with a BB defined over partition k that wins every competition with the strings with BB over other partitions. In reality this kind of situation may arise when a particular partition is highly scaled compared to other partitions.

In other words for all j, $\alpha_{nj} = 2$ and $\alpha_{jn} = 0$. If that be the case, the growth equation for $q_{ng,t}$ simplifies to,

$$\frac{\partial q_{ng,t}}{\partial t} = q_{ng,t}\left[1 - q_{ng,t}\right] \tag{C.6}$$

To make the situation even worse, we assume that the strings with BBs from partition i looses every cross-competition with other strings containing good BBs from other partitions.

We also assume that, there exists the other extreme kind of good BB ig, which looses every cross-competition with other good BBs.

In order to keep things simple, let us assume rest of the BBs behaves identically and they loose or win without any a priori bias, except when they compete with either k or i; they loose every time they compete with one of k and they win every competition with one of i-s. We can average the proportions of such BBs and define a variable $q_{j,t}$, representing the average proportion of such BBs. The growth equation for BB i is,

$$\frac{\partial q_{i,t}}{\partial t} = q_{i,t}(1 - q_{i,t} - 2(m-2)q_{j,t} - 2q_{k,t}) \tag{C.7}$$

Similarly we can write,

$$\frac{\partial q_{j,t}}{\partial t} = q_{j,t}(1 - (m-2)q_{j,t} - 2q_{k,t}) \tag{C.8}$$

Equation C.6 can be solved giving,

$$q_{k,t} = \left[1 + \frac{1 - q_{k,t_0}}{q_{k,t_0}} \exp^{-t}\right]^{-1}$$

Substituting this in equation C.8 we get,

$$q_{j,t} = \frac{\exp^{t - 2\log(1 - q_{k,t_0} + \exp^t q_{k,t_0})}}{\frac{1}{q_{j,t_0}} - \frac{2-m}{q_{k,t_0}} + \frac{2-m}{q_{k,t_0} + q_{k,t_0}^2 \exp^t q_{k,t_0}^2}}$$

Finally from C.7,

$$q_{i,t} = \frac{\exp^{t - 2\log p}}{\frac{1}{q_{i,t_0}} + \frac{1}{-2q_{j,t_0} + mq_{j,t_0} + q_{k,t_0}} - \frac{1}{p(-2q_{j,t_0} + mq_{j,t_0} + q_{k,t_0})}}$$

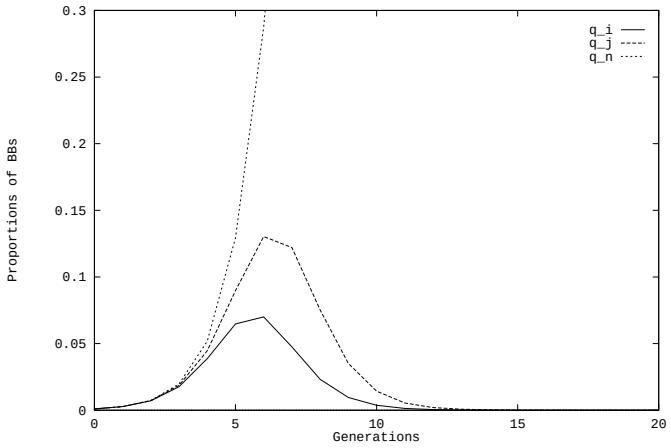


Figure C.2: Variation of $q_{i,t}, q_{j,t}$ and $q_{k,t}$ along time.

$$\text{where, } p \quad = \quad 1 + 2q_{j,t_0} - 2\exp^t q_{j,t_0} - mq_{j,t_0} + \exp^t mq_{j,t_0} - q_{k,t_0} + \exp^t q_{k,t_0}$$

The above closed form equations look involved. Any closed form solution for the optimum value of t^*, at which $(q_{k,t} - q_{i,t}) > \delta$ (where δ is some small value) for the first time, is likely to be even more complicated. Therefore, it may be more practical to find the optimal time iteratively. Figure C.2 shows the growth of each of these kinds, each having an initial proportion of 0.001. Figure C.3 shows the values of t^* for different values of m and $q_{i,0} = q_{j,0} = q_{k,0}$; all the t^* values are computed for $\delta = 0.01$;

This section pointed out that despite the presence of thresholding, binary tournament selection can introduce cross-competition that can eliminate some good strings. Therefore, selection only iterations within each episode cannot be continued for indefinite period. The bounding cases described in the above sections provide some idea about the possible range of values. For most of the experimental results presented in this thesis, the number of generations within each of these episodes have been less than 5. The following section considers the effect of gene deletion that takes the population from one episode to the next.

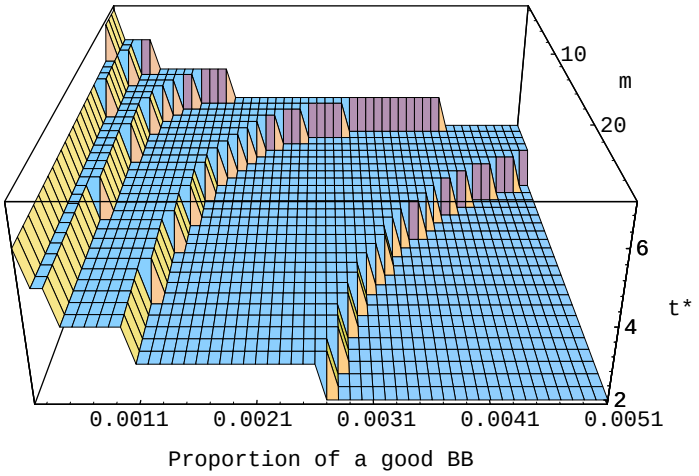


Figure C.3: t^* as a function of number of good BBs m and initial BB proportions within a subpopulation; $q_{i,0} = q_{j,0} = q_{k,0} = 0.001$.

C.2 Random Deletion of Genes

After selection is applied several times in a particular episode, denoted by subscript e, some genes are randomly deleted. As a result the string length is reduced from $\lambda_{(e)}$ to $\lambda_{(e+1)}$. Gene deletion can destroy a BB in a string. The probability of the survival of a BB, is

$$\eta_{e+1} = \frac{\binom{\lambda_{(e)}-k}{\lambda_{(e+1)}-k}}{\binom{\lambda_{(e)}}{\lambda_{(e+1)}}}$$

where k is the size of the BBs. Note that a junk string that does not contain any BB can only remain so; gene deletion cannot generate a BB. Let us use $q_{i,t,e}$ to denote the proportion of strings with BB from partition i at iteration t of episode e. Similar convention will be applied for other similar variables. This additional subscript e will be dropped wherever it is not necessary.

$$q_{i,0,e+1} = \eta_{e+1} q_{i,t^*,e} \tag{C.9}$$

$$q_{j,0,e+1} = \eta_{e+1} q_{i,t^*,e} \tag{C.10}$$

$$q_{junk,0,e+1} = (1-\eta_{e+1}) \sum_i q_{i,t^*,e} + q_{junk,t^*,e} \tag{C.11}$$

where t^* represents the final selection only iteration of episode e.

The following section considers the problem of choosing the thresholding parameter in an episode.

C.3 Choosing the Thresholding Parameter

So far we have developed an approach to decide how long selection should be applied in a particular episode. Design of filtering schedule is not complete until we address the effect of thresholding parameter on the niches. In this section we develop a methodology for choosing the thresholding parameter. One possible way to do so is to minimize disproportionate growth of building-blocks by controlling the size of their niches. Subsection C.3.1 presents a general overview of the approach. Subsections C.3.2 C.3.3 quantify the effect of string size reduction on the size of niches.

C.3.1 Overview

At the initial stage when the string length is close to problem length, most of the niches are very large in size, since almost every string matches with any other string. Let us denote the thresholding parameter of episode e by $\theta_{(e)}$. As the string length is reduced by random gene deletions, corresponding thresholding parameter is also gradually reduced. When $\theta_{(e)}$ is reduced to $\theta_{(e+1)}$ some strings are either thrown out or pulled inside of a niche. This changes the initial proportions of BBs in the $(e+1)$–th episode. The choice of $\Delta\theta_{(e)} = (\theta_{(e)} - \theta_{(e+1)})$ turns out to be a design parameter controlling the sizes of the niches which in turn determines the success in maintaining all the BBs in the population. Before we actually demonstrate that, let me first briefly present an overview of the fundamental ideas of this section.

Choosing the right thresholding parameter requires analyzing the behavior of thresholding niches under selection and gene deletion. Explicit tracking of all the niches appears very difficult if not impossible. The approach that I take here draws its motivation from the *Lagrangian* style of solving problems in fluid mechanics. I focus on the behavior of one arbitrarily chosen niche defined with respect to an arbitrarily chosen reference string containing a BB. I analyze the behavior of this niche as selection and gene deletion are applied. The niche is always defined with respect to the reference string and the reference string itself gets changed because of gene

deletion. Therefore this niche is not a fixed window to the population; rather it is a sliding window that changes itself as the reference string is modified. Such approach is often called a *Lagrangian approach* in contrast of the *Eulerian approach* in which the window remains fixed in the space that we are looking at. Let us define S_I the primary thresholding niche or simply niche of a particular reference string as the set of strings in which every member has at least $\theta_{(e)}$ common genes with the reference string. Define a secondary niche S_{II} of the reference string as the set of all strings that do not compete with the reference string, but do compete with at least one string in S_I. The behavior of the reference string is viewed as an interplay among its primary and secondary niche. The reference string can compete with any string in S_I but not with any of S_{II}. S_{II} can be viewed as a boundary region outside S_I. As the strings in S_I are reduced by gene deletion, some strings from S_{II} may become a member of S_I and some members of S_I may join S_{II}.

The objective of the building-block filtering episodes is to increase the proportions of the good BBs. Since the population size is assumed constant, this is possible only if there exist some bad junk strings within the niches, that can be eliminated to make more copies of the strings with BBs. After the initial stage, random deletion of genes is the only major source of such bad strings. Even if the population contains enough junk strings, their fullest exploitation depends on the choice of the thresholding parameter. If $\theta_{(e)}$ is very stringent, the size of the niche will be small and the number of niches will increase. This may result in poor growth and ultimate extinction of some BBs. On the other hand if $\theta_{(e)}$ is relaxed too much, the size of the niche may grow excessively and thereby neutralize the role of thresholding in restricting cross-competition. One possible way to minimize such undesirable consequences is to choose $\theta_{(e)} - s$ in such a way that, the size of the niches neither go below a desired limit nor does it increase beyond a limit.

To choose the right $\Delta\theta_{(e)}$ that minimizes the change in the size of the niches, we need to compute the following things:

1. the minimum required $\Delta\theta_{(e)}$ for keeping strings which were originally within $S_I^{(e)}$ into $S_I^{(e+1)}$.

2. the maximum allowable $\Delta\theta_{(e)}$ beyond which strings which were originally with $S_{II}^{(e)}$ would go into $S_I^{(e+1)}$.

The first item address the loss of strings from a niche. The second item considers the gain of strings from other niches. Each of these will be described in detail in the following sections.

C.3.2 Keeping strings within the original niche

As noted earlier, the size of a niche, created by thresholding depends on the chosen value of thresholding parameter. If the thresholding parameter is not reduced enough after the gene deletion, the size of a niche may become very small and may lead to extinction. In this section I compute the minimum reduction in θ, for which all the strings which were in $S_{I(e)}$ will also be in $S_{I(e+1)}$.

The expected value of the minimum required $\Delta\theta_{(e)}$ can be calculated as follows. Consider two strings with some $\theta_{(e)}+d$ common genes, where d may vary between 0 and $\xi_{(e)} = (\lambda_{(e)} - \theta_{(e)})$. Denote the number of genes to be randomly deleted from each of these strings by $\delta_{(e)} = \lambda_{(e)} - \lambda_{(e+1)}$. The expected value for $\min\Delta\theta_{(e)}$, which keeps two strings, having $(\theta_{(e)} + d)$ similar genes, together within the same niche, even after the random deletion of $\delta_{(e)}$ genes from both of them is,

$$
\begin{aligned}
E\left[\min\Delta\theta\right]_{(\theta_{(e)}+d)} &= \sum_{x=\max(g,0)}^{\min(\delta_{(e)},(\theta_{(e)}+d))} \frac{\binom{(\theta_{(e)}+d)}{x}\binom{\xi_{(e)}-d}{\delta_{(e)}-x}}{\binom{\lambda_{(e)}}{\delta_{(e)}}} \sum_{y=\max(g,0)}^{\delta_{(e)}} \frac{\binom{\xi_{(e)}-d}{\delta_{(e)}-y}}{\binom{\lambda_{(e)}}{\delta_{(e)}}} \cdot \\
&\qquad \sum_{j=\max((y-(\theta_{(e)}+d)+x),0)}^{\min(y,x)} \binom{x}{j}\binom{(\theta_{(e)}+d)-x}{y-j}\left[\min\Delta\theta(x,y,j)\right] \\
g &= \delta_{(e)} - (\lambda_{(e)} - \theta_{(e)}) \\
\min\Delta\theta(x,y,j) &= \max(((y+x)-j-d),0)
\end{aligned}
$$

The first summation from the left represents different ways of deleting $\delta_{(e)}$ genes from the reference string. The dummy variable, x denotes the number of genes picked up from the set of mutually common genes. The term on the immediate right of this summation denotes the probability associated with each of the different ways to delete $\delta_{(e)}$ genes. The next two summations denote the work on the second string. The rightmost summation is used to quantify the possible deletion of the same genes from both the strings.

$E\left[\min\Delta\theta\right]_{(\theta_e+d)}$ can be computed for all values of d and their average value, $E\left[\min\Delta\theta\right]$ weighted by the proportion of strings with corresponding number of similar genes tells us that

in order to keep all the strings of $S_{I(e)}$ within $S_{I(e+1)}$ in an average sense,

$$\Delta\theta_{(i)} \geq E\left[\min\Delta\theta\right] \tag{C.12}$$

This provides the average value of the reduction in $\theta_{(e)}$ required for keeping strings within the same niche in the next episode. The following section considers the possibility of gaining new strings by choosing a relaxed value of the thresholding parameter.

C.3.3 Restricting the influx

This section describes how to compute the maximum allowable reduction in θ that makes sure that the strings which were not originally in $S_{I(e)}$ are not allowed to be a member of $S_{I(e+1)}$. Equation C.12 can also be used to compute the maximum allowable reduction by letting d vary through $-1, \cdots -\xi_{(e)}$. The corresponding weighted average, say $E\left[\max\Delta\theta\right]$ gives us the other inequality,

$$\Delta\theta_{(e)} < E\left[\max\Delta\theta\right] \tag{C.13}$$

In the rest of the discussion we often refer to inequalities C.12 and C.13 together by **minimax criterion**. Figure C.4 shows the spectrum of values for every components of both the inequalities. The abscissa denotes the string length; On any particular vertical column diamond shaped points denote the spectrum of $E\left[\max\Delta\theta\right]_{(\theta_{(e)}+d)}$ for d varying through $-1 \cdots -\xi_{(e)}$. The dotted line portion on each vertical line denotes the spectrum of $E\left[\min\Delta\theta\right]_{(\theta_{(e)}+d)}$ for d varying through $0, \cdots \xi_{(e)}$. This set of data is computed for $l = 100$ and a string length reduction rate of 1.1. The initial string length, $\lambda_{(0)} = 95$ and the corresponding $\theta_{(0)} = ceil(95 * 95/100) = 91$. For every subsequent filtering stages, we compute the minimum required value of $\Delta\theta$ in order to keep every string in $S_{I(e)}$ with $(\theta_{(i)} + d)$ genes similar to the reference string of that subpopulation (for every possible d varying 0 through $(\lambda_{(e)} - \theta_{(e)})$), within the subsequent $S_{I(e+1)}$-s. Each of these values are connected through the dotted vertical lines. Similarly the diamond shaped points represent the maximum allowable value of $\Delta\theta$ in order to keep every string in $S_{II(e)}$ with $(\theta_{(e)} + d)$ genes similar to the reference string of that subpopulation(for every possible d varying through -1 to $-\xi_{(e)}$, out of the subsequent $S_{I(e+1)}$-s. The figure clearly shows that the two spectrum are almost non-overlapping. The only trouble seems to come from keeping

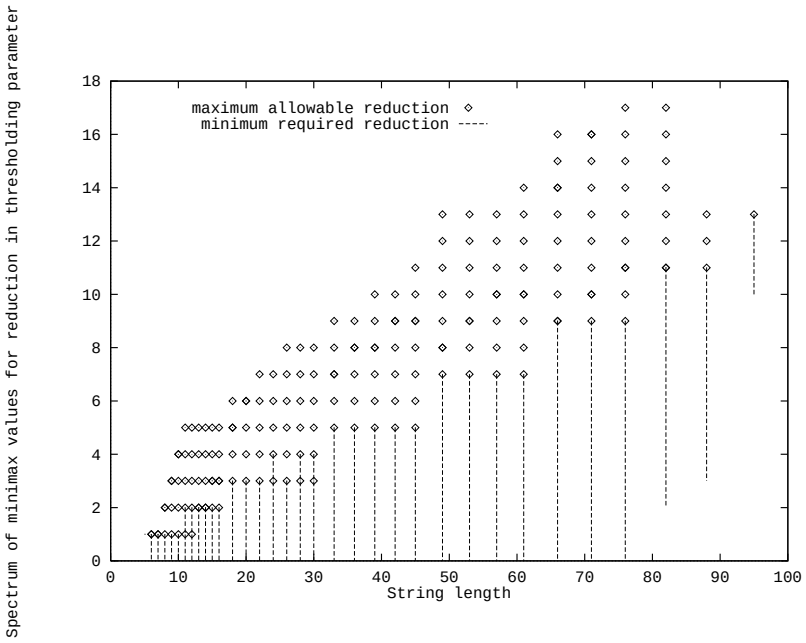


Figure C.4: The spectrum of values composing minimum required $\Delta\theta$ for keeping all the members of $S_I^{(i)}$ into $S_I^{(i+1)}$ and the same, composing the maximum allowable $\Delta\theta$ in order to keep the members of $S_{II}^{(i)}$ out of $S_I^{(i+1)}$.

strings which are at the boundary interface of $S_{I(e)}$, i.e. mainly the strings with just $\theta_{(e)}$ and $(\theta_{(e)} - 1)$ genes in common with the reference string.

This minimax spectrum of values provide different choices of threshold values with different degrees of qualitative behavior. A value from the upper region of the spectrum will tend to be less restrictive. In other words, it will allow some strings to go in and out from the niche. On the other hand a value from the lower spectrum is more restrictive.

The following section summarizes the results of this chapter.

C.4 Discussion

The results of the previous sections provide some idea about how to choose a building-block filtering schedule. The models of selection under cross-competition and the minimax criterion can be used to design a building-block filtering schedule. In this section, I briefly summarize the results.

The simple models of growth of building-blocks in presence of cross-competition, tells us about the approximate range for which thresholding selection should be applied in an episode.

In general, the growth of building-blocks under selection is problem dependent; however, since tournament selection is somewhat independent of the absolute objective function values, the ranges of values found using these models should work for different problems. For most of the reported experimental results on fmGA, the episode length have been restricted to less than 5.

Section C.3 provided us with some ideas about choosing the thresholding parameter. The minimax criterion can be used to choose a relatively stringent or relaxed thresholding value. However, determining the exact values of $E[\min\Delta\theta]$ and $E[\max\Delta\theta]$ are not possible unless we compute the distributions of the niches. Since this is not quite feasible, a more pragmatic and approximate approach may be adopted. At the initial stage of the fmGA, when the string length is close to problem length almost every string matches with others. As a result the size of a niche is large. Therefore, for relatively larger value of string length, loosing strings from the niche by adopting a relaxed strategy may be reasonable. On the other hand when the string size reduces, niche size also decreases. When the size of a niche is smaller than a threshold, a loosing strings should be restricted. These observations can be used to choose the thresholding parameters, as described in the following.

First, we need to divide the filtering schedule into two parts — (1) the portion with somewhat relaxed thresholding and, (2) the later part of the filtering process during which thresholding should be more conservative. For all the experiments reported here, the later part begins when the string length is reduced to βk, where β is a constant and k is the size of the building-blocks. We used $\beta = 2$. During the first part we choose a value from the upper region of the minimax spectrum. In other words a relatively larger value of $\Delta\theta$ is used. In this thesis, the average value of the upper spectrum is used as the thresholding parameter for the first part. For the last part of the filtering schedule a more conservative value of $\Delta\theta$ should be chosen. I used the average of the complete minimax (both upper and lower) spectrum for the second part of the schedule.

One of the important conclusions of this analysis of thersholding is that cross-competition among different building-blocks plays a significant role in thresholding selection. The fundamental problem originates from the implicit definition of the relation and class spaces. Comparison of relations is tried to accomplish by comparing classes from two different relations, which is inappropriate. For large problems, such cross-competition is likely to eliminate some of the building-blocks.

Bibliography

Ackley, D. H. (1987). *A connectionist machine for genetic hill climbing*. Boston: Kluwer Academic.

Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., & Watson, J. D. (1994). *Molecular biology of the cell*. New York: Garland Publishing Inc.

Archetti, F., & Schoen, F. (1984). A survey on the global optimization problem: General theory and computational approaches. *Annals of Operations Research*, *1*(1), 87–110.

Bagley, J. D. (1967). The behavior of adaptive systems which employ genetic and correlation algorithms. *Dissertation Abstracts International*, *28*(12), 5106B. (University Microfilms No. 68-7556).

Bethke, A. D. (1981). Genetic algorithms as function optimizers. *Dissertation Abstracts International*, *41*(9), 3503B. (University Microfilms No. 8106101).

Betrò, B. (1983). A bayesian nonparametric approach to global optimization. In Stähly, P. (Ed.), *Methods of Operations Research* (pp. 45–47). Atenäum Verlag.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1987). Occam's razor. *Information Processing Letter*, *24*, 377–380.

Blumer, A., Haussler, D., & Warmuth, M. K. (1990). Learnability and the vapnik-chervonenkis dimension. *Journal of the Association for Computing Machinery*, *36*(4), 929–965.

Booker, L. B. (1982). Intelligent behavior as an adaptation to the task environment. *Dissertations Abstracts International*, *43*(2), 469B. (University Microfilms No. 8214966).

Booker, L. B. (1993). Recombination distributions for genetic algorithms. In Whitley, L. D. (Ed.), *Foundations of Genetic Algorithms 2* (pp. 29–44). San Mateo, California: Morgan Kaufmann Publishers.

Brindle, A. (1981). *Genetic algorithms for function optimization.* Unpublished doctoral dissertation, University of Alberta, Edmonton, Canada.

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to algorithms.* Massachussets: MIT Press ; New York: McGraw-Hill.

Culberson, J. (1994). Mutation-crossover isomorphisms and the construction of discriminating functions. *Evolutionary Computation*, *2*(3), 279–311.

Dantzig, G. B. (1963). *Linear programming and extensions.* New Jersey: Princeton University Press.

Dasgupta, D., & McGregor, D. R. (1992). Designing neural networks using the structured genetic algorithm. *Artifical Neural Networks*, *2*, 263–268.

David, H. A. (1981). *Order statistics.* New York: John Wiley & Sons, Inc.

Davis, L. (Ed.) (1987). *Genetic algorithms and simulated annealing.* Los Altos, CA: Morgan Kaufmann.

De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. *Dissertation Abstracts International*, *36*(10), 5140B. (University Microfilms No. 76-9381).

Deb, K. (1991). *Binary and floating-point function optimization using messy genetic algorithms* (IlliGAL Report No. 91004). Urbana: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Deb, K., & Goldberg, D. E. (1993). Analyzing deception in trap functions. In Whitley, L. D. (Ed.), *Foundations of Genetic Algorithms 2* (pp. 93–108). San Mateo, CA: Morgan Kaufmann.

Deb, K., & Goldberg, D. E. (1994b). Sufficient conditions for deceptive and easy binary functions. *Annals of Mathematics and Artificial Intelligence*, *10*, 385–408.

Dixon, L. C. W., & Szegö, G. P. (1978). The global optimization problem: an introduction. In Dixon, L. C. W., & Szegö, G. P. (Eds.), *Towards global optimization 2* (pp. 1–15). Amsterdam: North-Holland.

Dreyfus, S. E., & Law, A. M. (1977). *The art and theory of dynamic programming*. New York: Academic Press.

Dueck, G., & Scheuer, T. (1988). *Threshold accepting–a general purpose optimization algorithm appearing superior to simulated annealing* (Technical Report No. 88.10.011). IBM Heidelberg Sci. Center.

Dymek, A. (1992). *An examination of hypercube implementations of genetic algorithms* (Masters thesis and Report No. AFIT/GCS/ENG/92M-02). Wright-Patterson Air Force Base: Air Force Institute of Technology.

Elsley, D. (1990). *The use of neural networks for simultaneous tracking of large numbers of targets* (Report No. SC5490.FR). Rockwell International Science Center.

Feller, W. (1959). *An introduction to probability theory and its applications* (2nd ed.). New York: John Wiley & Sons, Inc.

Ferreira, A. G., & Žerovnik, J. (1993, 10/11). Bounding the probability of success of stochastic methods for global optimization. *Computers, Mathematics, Applications*, *25*, 1–8.

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). *Artificial intelligence through simulated evolution*. New York: John Wiley.

Forrest, S., & Mitchell, M. (1993). Relative building-block fitness and the building-block hypothesis. In Whitley, L. D. (Ed.), *Foundations of Genetic Algorithms* (pp. 109–126). San Mateo, CA: Morgan Kaufmann.

Gibbons, J. D., Sobel, M., & Olkin, I. (1977). *Selecting and ordering populations: A new statistical methodology*. New York: John Wiley & Sons, Inc.

Glover, F. (1989). Tabu search - part I. *ORSA Journal on Computing*, *1*, 190–206.

Goldberg, D. E. (1987). Simple genetic algorithms and the minimal, deceptive problem. In Davis, L. (Ed.), *Genetic Algorithms and Simulated Annealing* (pp. 74–88). San Mateo, CA: Morgan Kaufmann. (Also TCGA Report 86003).

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

Goldberg, D. E. (1990). A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems*, *4*(4). (Also TCGA Report No. 90003).

Goldberg, D. E., & Deb, K. (1991a). A comparative analysis of selection schemes used in genetic algorithms. In Whitley, L. D. (Ed.), *Foundations of Genetic Algorithms* (pp. 69–93). San Mateo, CA: Morgan Kaufmann.

Goldberg, D. E., & Deb, K. (1991b). 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*, 333–362.

Goldberg, D. E., Deb, K., & Clark, J. H. (1993). Accounting for noise in the sizing of populations. In Whitley, L. D. (Ed.), *Foundations of Genetic Algorithms 2* (pp. 127–140). San Mateo, CA: Morgan Kaufmann.

Goldberg, D. E., Deb, K., Kargupta, H., & Harik, G. (1993). Rapid, accurate optimizaiton of difficult problems using fast messy genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 56–64). San Mateo, CA: Morgan Kaufmann.

Goldberg, D. E., Deb, K., & Korb, B. (1990a). *An investigation of messy genetic algorithms* (TCGA Report No. 90005). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

Goldberg, D. E., Deb, K., & Korb, B. (1990b). Messy genetic algorithms revisited: Studies in mixed size and scale. *Complex Systems*, *4*.

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.

Goldberg, D. E., & Kerzic, T. (1990). *mGA1.0: A common LISP implementation of a messy genetic algorithm* (TCGA Report No. 90004). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

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 89003).

Goldberg, D. E., & Lingle, R. (1985). Alleles, loci, and the traveling salesman problem. In Grefenstette, J. J. (Ed.), *Proceedings of an International Conference on Genetic Algorithms and Their Applications* (pp. 154–159). Hillsdale, NJ: Lawrence Erlbaum Associates.

Gomulka, J. (1978). Deterministic vs probabilistic approaches to global optimization. In Dixon, L. C. W., & Szegö, G. P. (Eds.), *Towards global optimization* (pp. 19–29). Amsterdam: North-Holland.

Grefenstette, J. J., & Baker, J. E. (1989). How genetic algorithms work: A critical look at implicit parallelism. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 20–27). San Mateo, CA: Morgan Kaufmann.

Hart, P. E., Nilson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Hart, W. E. (1994). *Adaptive global optimization with local search*. Doctoral dissertation, Department of Computer Science, University of California, San Diego.

Haussler, D. (1989). Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, *2*(36), 177–222.

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

Hollstien, R. B. (1971). Artificial genetic adaptation in computer control systems. *Dissertation Abstracts International*, *32*(3), 1510B. (University Microfilms No. 71-23,773).

Horn, J., & Goldberg, D. E. (1995). Genetic algorithm difficulty and the modality of fitness landscapes. In Whitley, L. D., & Vose, M. (Eds.), *Foundations of Genetic Algorithms* (pp. 243–269). San Mateo, CA: Morgan Kaufmann.

Jacob, F., & Monod, J. (1961). Genetic regulatory mechanisms in the synthesis of proteins. *Molecular Biology*, *3*, 318–356.

Jones, D. R., & Stuckman, B. E. (1992). Genetic algorithms and the Bayesian approach to global optimization. *Proceedings of the 1992 International Fuzzy Systems and Intelligent Control Conference*, 217–235.

Jones, T. (1995). *Evolutionary algorithms, fitness landscapes and search.* Doctoral dissertation, Department of Computer Science, University of New Mexico, Albuquerque, NM.

Jones, T., & Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In Eshelman, L. (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 184–192). San Mateo, CA: Morgan Kaufmann.

Jones, T. C. (1994). A description of Holland's royal road function. *Evolutionary computation*, *2*(4), 411–417.

Kargupta, H. (1995a). *An alternate model of evolution: The SEARCH perspective.* In preperation.

Kargupta, H. (1995b). Signal-to-noise, crosstalk and long range problem difficulty in genetic algorithms. In Eshelman, L. (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 193–200). San Mateo, CA: Morgan Kaufmann.

Kargupta, H., Deb, K., & Goldberg, D. (1992). Ordering genetic algorithms and deception. In Männer, R., & Manderick, B. (Eds.), *Parallel Problem Solving from Nature* (pp. 47–56). Amsterdam: Noth-Holland.

Kirpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the travelling salesman problem. *Operations Research*, *21*, 498–516.

Mahfoud, S. W., & Goldberg, D. E. (1992). A genetic algorithm for parallel simulated annealing. In Männer, R., & Manderick, B. (Eds.), *Parallel Problem Solving from Nature* (pp. 301–310). Amsterdam: Noth-Holland.

Merkle, L. D., & Lemont, G. B. (1993). Comparison of parallel messy genetic algorithm data distribution strategies. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 191–205). San Mateo, CA: Morgan Kaufmann.

Michalski, R. S. (1983). Theory and methodology of inductive learning. In Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.), *Machine learning: An artificial intelligence approach* (pp. 323–348). Tioga Publishing Co.

Mitchell, T. M. (1980). *The need for biases in learning generalizations* (Rutgers Computer Science Tech. Rept. CBM-TR-117). Rutgers University.

Mühlenbein, H. (1992b). How genetic algorithms really work: I.Mutation and hillclim bing. In Männer, R., & Manderick, B. (Eds.), *Parallel Problem Solving from Nature, (Vol. 2)* (pp. 15–25). Amsterdam, The Netherlands: Elsevier Science.

Natarajan, B. K. (1991). *Machine learning, a theoretical approach*. San Mateo, CA: Morgan Kaufmann.

Papadimitriou, C. L., & Steiglitz, K. (1982). *Combinatorial optimization*. New Jersey: Prentice Hall.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106.

Radcliffe, N. J. (1991). Formal analysis and random respectful recombination. In Belew, R. K., & Booker, L. B. (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 222–229). San Mateo, CA: Morgan Kaufmann.

Radcliffe, N. J. (1993). Genetic set recombination. In Whitley, L. D. (Ed.), *Foundations of Genetic Algorithms* (pp. 203–219). San Mateo, CA: Morgan Kaufmann.

Rechenberg, I. (1973). Bionik, evolution und optimierung. *Naturwissenschaftliche Rundschau*, *26*, 465–472.

Rich, E., & Knight, K. (1991). *Artificial intelligence* (2 ed.). New York: McGraw Hill.

Rinnooy Kan, A. H. G., & Timmer, G. T. (1984). Stochastic methods for global optimization. *American Journal of Mathematics and Management Sciences*, *4*(1), 7–40.

Rosenberg, R. S. (1967). Simulation of genetic populations with biochemical properties. *Dissertation Abstracts International*, *28*(7), 2732B. (University Microfilms No. 67-17,836).

Rudnick, M., & Goldberg, D. E. (1991). *Signal, noise, and genetic algorithms* (IlliGAL Report No. 91005). Urbana: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Rudolph, G. (1994). Massively parallel simulated annealing and its relation to evolutionary algorithms. *Evolutionary Computation*, 361–383.

Schoen, F. (1991). Stochastic techniques for global optimization: A survey of recent advances. *Journal of Global Optimization*, *1*, 207–228.

Sirag, D. J., & Weisser, D. J. (1987). Toward a unified thermodynamic genetic operator. In Grefenstette, J. J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 116–122). Hillsdale, NJ: Lawrence Erlbaum Associates.

Smith, R. E. (1988). *An investigation of diploid genetic algorithms for adaptive search of nonstationary functions* (TCGA Report No. 88001). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

Stryer, L. (1988). *Biochemistry*. New York: W. H. Freeman Co.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 2–9).

Thierens, D., & Goldberg, D. (1993). Mixing in genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 38–45). San Mateo, CA: Morgan Kaufmann.

Törn, A., & Žilinskas, A. (1989). *Global optimization*. Berlin: Springer-Verlag.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the Association for Computing Machinery*, *27*(11), 1134–1142.

Vavasis, S. A. (1991). *Nonlinear optimization: Complexity issues*. New York: Oxford University Press.

Watanabe, S. (1969). *Knowing and guessing - A formal and quantative study*. New York: John Wiley & Sons, Inc.

Vita

HILLOL KARGUPTA
3802 Gold Street, Apt. # 4
Los Alamos, NM 87544

E-mail: hillol@gal1.ge.uiuc.edu

EDUCATION

Ph.D. in Computer Science: University of Illinois at Urbana-Champaign, IL, USA.
Period: August, 1991–September, 1995.

M.Tech. in Mechanical Engineering: Indian Institute of Technology, Kanpur, India.
Period: December, 1988–May, 1990.

B.Tech. in Mechanical Engineering: Regional Engineering College Calicut, India.
Period: August, 1984–October, 1988.

PROFESSIONAL EXPERIENCE

Research Assistant, University of Illinois at Urbana-Champaign Development of an ordinal, probabilistic, and approximate framework, SEARCH, for blackbox optimization in terms of relations, classes and ordering. Sponsored by US Air Force, AFOSR Grant F49620-94-1-0103. Duration: August, 1994–September, 1995.

Research Assistant, University of Illinois at Urbana-Champaign Theoretical and experimental analysis of population sizing for serial and parallel genetic algorithms. Sponsored by US Air Force, AFOSR Grant F49620-94-1-0103. Duration: August, 1994–September, 1995.

Research Assistant, University of Illinois at Urbana-Champaign. Analytical formulation of problem difficulty in Genetic Algorithms using detection theory and complexity theory. Sponsored by US Air Force, AFOSR Grant F49620-94-1-0103. Duration: January, 1994–August, 1994.

Research Assistant, University of Illinois at Urbana-Champaign. Virtual decision support systems for army installations. Sponsored by US Army Construction Engineering Laboratory. Duration: January, 1994–August, 1994.

Teaching & Research Assistant, University of Illinois at Urbana-Champaign. Graduate level course CS 342 – Knowledge Acquisition and Machine learning Instructor: Prof. David Wilkins, Dept. of Computer Science, UIUC. Duration: August, 1993–December, 1993.

Research Assistant, University of Illinois at Urbana-Champaign Development of Fast Messy Genetic Algorithm and its application to scan-to-scan missile tracking problem. Sponsored by United States Army, Contract DASG60-90-C-0153 Duration: January, 1992–August, 1993.

Research Assistant, University of Illinois at Urbana-Champaign. Design of difficult deceptive permutation problems and application of genetic algorithms for solving them. Sponsored by NSF Grant ECS-9022007. Duration: August, 1991–December, 1991.

Software Engineer, ADA Software Inc., Calcutta, India. Graphics application software development for drug design. Duration: May, 1991–August, 1991

Research and Teaching Assistant, University of Alabama. Research on Genetic Algorithms and Undergraduate level course on Dynamics. Duration: August, 1990–May, 1991.

Research Engineer, Computer Aided Design Center, Indian Institute of Technology, Kanpur, India. Development of a graphical front end for an expert system. Duration: June, 1990–August, 1990.

DATE OF BIRTH

25th March, 1967.

PLACE OF BIRTH

Darjeeling, India

AWARDS

National Talent Certificate (India), 1982.
Honors Distinction in B. Tech., 1988.
Conference travel grant from Sigma-Xi, UIUC for the paper titled, *Drift, diffusion, and Boltzmann distribution in simple genetic algorithm* (Kargupta, 1992).
Conference travel grant from Sigma-Xi, UIUC for the paper titled, *Temporal sequence processing based on the biological reaction-diffusion process* (Kargupta & Ray, 1994).

PROFESSIONAL ACTIVITIES

IEEE student member.

Member of International Society for Genetic Algorithms.

CONFERENCE PROGRAM COMMITTEE

International Conference on Genetic Algorithms, 1995

REVIEWER FOR

International Conference on Genetic Algorithms 1995.
International Journal of Applied Intelligence

REFEREED JOURNAL AND CONFERENCE PUBLICATIONS

1. Kargupta, H. (1995a). SEARCH, Polynomial Complexity, and The Fast Messy Genetic Algorithm. Ph.D. dissertation, Department of Computer Science, University of Illinois. *Technical report number UIUCDCS-R-95-1931.* Also available as Illinois Genetic Algorithm Laboratory technical report 95008.

2. Kargupta, H. & Goldberg, D. E. (1995b). Polynomial Complexity Blackbox Optimization: The SEARCH Perspective and Its Implications. Submitted in *Journal of the Association for Computing Machinery.*

3. Kargupta, H. & Goldberg, D. E. (1995c). SEARCH: An Alternate Perspective Toward Blackbox Optimization. To be submitted in *1996 IEEE International Conference on Evolutionary Computation.* Nagoya University, Japan.

4. Kargupta, H. (1995d). Signal-to-noise, Crosstalk and Long Range Problem Difficulty in Genetic Algorithms. In L. Eshelman (Ed.). *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 193–200). San Mateo: Morgan Kaufmann.

5. Ray S. & Kargupta, H. (1993). A Temporal Sequence Processor Based on the Biological Reaction-Diffusion Process. Submitted in *Complex Systems.*

6. Grobler, F & Subick, C. & Kargupta, H. & Govindan, K. (1994). Optimization of Uncertain Resource Allocation with Genetic Algorithm. *Presented in Second ASCE Congress for Computing in Civil Engineering, Atlanta, GA June, 1995.*

7. Kargupta, H. & Ray S. (1994). Temporal Sequence Processing Based on the Biological Reaction-Diffusion Process. *Proceedings of the IEEE World Congress on Computational Intelligence.* Volume 4, 1994 (ICNN '94). Piscataway, NJ: IEEE Service Center, 2315–2320.

8. Kargupta, H, and Ray, S. (1994). Reaction Diffusion System for Learning the Structure in Time. *Proceedings of the World Congress on Neural Networks*, San Diego, International Neural Network Society Press, Vol. 3, pp 150–155.

9. Goldberg, D. E. & Deb, K. & Kargupta, H. & George, H. (1993). Rapid Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms. In Forrest, S. (Ed.). *Proceedings of The Fifth International Conference On Genetic Algorithms.* Morgan Kaufmann Publishers. 56–64.

10. Kargupta, H. (1993a). Information Transmission in Genetic Algorithm and Shannon's Second Theorem. In Forrest, S. (Ed.). *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers. 640.

11. Kargupta, H. (1992). Drift, Diffusion and Boltzmann Distribution in Simple Genetic Algorithm, In Matzke, D. (Ed.) *Proceedings of the Workshop on Physics and Computation*. IEEE Computer Society Press. 137–145.

12. Kargupta, H. & Deb, K. & Goldberg, D. (1992a). Ordering Genetic Algorithms and Deception. In Manner, R. & Manderick, B. (Eds.), *Parallel Problem Solving from Nature, 2*, Elsevier Science Publishers. 47–56.

13. Kargupta, H. & Smith, R. (1991). System Identification Using Evolving Polynomial Network. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Eds: Belew, R. & Booker, L. Morgan Kaufmann Publishers. 370–376.

14. Sahay, B. & Kargupta, H. (1990). An Expert System for Solving Partial Differential Equations. *Presented in the International Conference for Computer Aided Science and Technology* Barcelona, Spain.

TECHNICAL REPORTS

1. Goldberg, D. E. & Kargupta, H. & Horn, J. & Cantupaz, E. (1995). Optimal Deme Size for Serial and Parallel Genetic Algorithms. *Illinois Genetic Algorithm Laboratory Technical Report Number 95002.* Dept. of General Engg., Univ. of Illinois at Urbana-Champaign, Urbana, Illinois, USA.

2. Kargupta, H. & Goldberg, D. (1994). Decision Making in Genetic Algorithms: a signal-to-noise perspective. *Illinois Genetic Algorithm Laboratory Technical Report No. 94004.* Dept. of General Engg., Univ. of Illinois at Urbana-Champaign, Urbana, Illinois, USA.

3. Ray S. & Kargupta, H. (1993). A Temporal Sequence Processor based on the Biological Reaction-Diffusion Process. *Technical Report Number. UIUCDCS-R-93-1842.* Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Urbana, Illinois, USA.

4. Kargupta, H. (1993). Information Transmission in Genetic Algorithm and Shannon's Second Theorem, *Illinois Genetic Algorithm Laboratory Technical Report No. 93003.* Dept. of General Engg., Univ. of Illinois at Urbana-Champaign, Urbana, Illinois, USA.

5. Kargupta, H., Deb K. & Goldberg D. E. (1993). Simultaneous Target Tracking using Fast Messy Genetic Algorithms. *Illinois Genetic Algorithm Laboratory Technical Report No. 94008.* Dept. of General Engg., Univ. of Illinois at Urbana-Champaign, Urbana, Illinois, USA.

WORKING PAPERS

1. Sarkar, K. and Kargupta, H. (1995). Design of Undoped Polysilicon Manufacturing Process Recipes Using Simple Genetic Algorithm.

2. Kargupta, H. (1995b). An Alternate Model of Evolution: The SEARCH Perspective.

3. Kargupta, H. (1994). Reversible Computation and Genetic Crossover.