Bayesian Optimization Algorithm: From Single Level to Hierarchy

Martin Pelikan

IlliGAL Report No. 2002023
July 2002

Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
117 Transportation Building
104 S. Mathews Avenue Urbana, IL 61801
Office: (217) 333-2346
Fax: (217) 244-5705

BAYESIAN OPTIMIZATION ALGORITHM:
FROM SINGLE LEVEL TO HIERARCHY

BY

MARTIN PELIKAN

DIPL., Comenius University, 1998

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, 2002

Urbana, Illinois

Abstract

There are four primary goals of this dissertation. First, design a competent optimization algorithm capable of learning and exploiting appropriate problem decomposition by sampling and evaluating candidate solutions. Second, extend the proposed algorithm to enable the use of hierarchical decomposition as opposed to decomposition on only a single level. Third, design a class of difficult hierarchical problems that can be used to test the algorithms that attempt to exploit hierarchical decomposition. Fourth, test the developed algorithms on the designed class of problems and several real-world applications.

The dissertation proposes the Bayesian optimization algorithm (BOA), which uses Bayesian networks to model the promising solutions found so far and sample new candidate solutions. BOA is theoretically and empirically shown to be capable of both learning a proper decomposition of the problem and exploiting the learned decomposition to ensure robust and scalable search for the optimum across a wide range of problems. The dissertation then identifies important features that must be incorporated into the basic BOA to solve problems that are not decomposable on a single level, but that can still be solved by decomposition over multiple levels of difficulty. Hierarchical BOA extends BOA by incorporating those features for robust and scalable optimization of hierarchically decomposable problems. A class of problems called hierarchical traps is then proposed to test the ability of optimizers to learn and exploit hierarchical decomposition. Hierarchical BOA passes the test and is shown to solve hierarchical traps and other hierarchical problems in a scalable manner. Finally, the dissertation applies hierarchical BOA to two important classes of problems of statistical physics and artificial intelligence—Ising spin-glass systems and maximum satisfiability. Experiments show that even without requiring any prior problem-specific knowledge about the structure of the problem at hand or its properties, hierarchical BOA is capable of achieving comparable or better performance than other state-of-the-art methods specializing in solving the examined classes of problems.

To my parents, Jaroslava and Peter.

Acknowledgments

There are many people to whom I am grateful for helping to get all the way to finishing up my dissertation. Foremost, I would like to thank my parents and the rest of my closest family, my brother Juraj, my sister Daniela and her husband Vlasto, and my godparents Dušanko and Elenka; I am grateful for having the family I have, it's perfect. I would also like to thank Angelica L. Farnum, who I take as a part of my family, for standing beside me along the way.

I wouldn't be here if it wasn't for my thesis advisor, David E. Goldberg, who gave me this great opportunity of working with him and other members of the Illinois Genetic Algorithms Laboratory. Dave Goldberg is a great teacher and a bottomless source of inspiration, and the Illinois Genetic Algorithms Laboratory is a wonderful place. I don't know how Dave Goldberg does it (and I don't know if he knows), but the laboratory is always full of great people. I would like to thank all the members and visitors of the lab that I had the opportunity of working with: Laura Albert, Leyla Babayeva, Jacob Borgerson, Martin Butz, Erick Cantú-Paz, Jian-Hung Chen, Ying-Ping Chen, Ross Gadient, Kathryn Hawley, Nazan Khan, Dimitri Knjazew, Alex Kosorukoff, Ruth Kwon, Jeffrey Leesman, Michelle Lipinski, Fernando Lobo, Mike Magin, Masaharu Munetomo, Felipe D. Padilla, Prasanna V. Parthasarathy, Franz Rothlauf, Kumara Sastry, Abhishek Sinha, Nathan Sis, Ravi Srivastava, Brad Sutton, Kurian Tharakunnel, Shigeyoshi Tsutsui, Clarissa Van Hoyweghen, Andy Vaughn, and Tian-Li Yu. I am particularly thankful to Martin Butz, Fernando Lobo, Erick Cantú-Paz, Franz Rothlauf, and Kumara Sastry.

I met several great friends in Illinois, many of whom were already listed above. My friends helped me a lot in getting this done and having fun at the same time, and I hope that we will continue to be friends for the times to come. In addition to the friends from the lab, I would like to thank Amanda Hinkle, Samarth Swarup, Nicole Swiss, and Dav Zimak.

I would also like to thank my dissertation committee, Dave Goldberg, Mehdi Harandi, Sylvian

Ray, and Dan Roth, for agreeing to serve on my committee and giving me valuable comments "from the other side". I am grateful for their help, discussions, and comments.

I am grateful to Vladimír Kvasnička and Jiří Pospíchal, who I worked with at the Department of Mathematics of the Slovak Technical University. Vladimír Kvasnička and Jiří Pospíchal awakened my interest in genetic algorithms and helped me take my first steps in the field. I would also like to thank Heinz Mühlenbein who taught and influenced me a lot during my stay in the group Adaptive Systems at the German National Center for Information Technology (GMD), two years prior to my arrival in Illinois.

A number of researchers contributed to this dissertation by providing valuable comments, answering my silly questions, and sharing their views. To name a few, I acknowledge Peter Bosman, David Chickering, Georges Harik, David Heckerman, Hillol Kargupta, Pedro Larrañaga, Thilo Mahnig, Bart Naudts, Jiří Očenášek, Josef Schwarz, Dirk Thierens, and Richard Watson.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants F49620-97-1-0050 and F49620-00-0163. Research funding for this project was also provided by a grant from the U.S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003, and a grant from the National Science Foundation under grant DMI-9908252. Some of the experiments were done using the computational facilities of the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

BB Building block.

BD Bayesian-Dirichlet (metric).

BIC Bayesian information criterion.

BMDA Bivariate marginal distribution algorithm.

BOA Bayesian optimization algorithm.

cGA Compact genetic algorithm.

CNF Conjunctive normal form.

DHC Discrete hill climber based on one-bit flips.

EBNA Estimation of Bayesian networks algorithm.

ECGA Extended compact genetic algorithm.

EDA Estimation of distribution algorithm.

EGNA Estimation of Gaussian networks algorithm.

FDA Factorized distribution algorithm.

GA Genetic algorithm.

GSAT Discrete hill climber for MAXSAT (one-bit flip).

hBOA Hierarchical Bayesian optimization algorithm.

HC Hill climber.

HIFF Hierarchical if-and-only-if.

H-PIPE Hierarchical PIPE.

IDEA Iterated density estimation algorithm.

LFDA Learning factorized distribution algorithm.

MAXSAT Maximum satisfiability.

MIMIC Mutual-information-maximizing input clustering.

PBIL Population-based incremental learning.

PIPE Probabilistic incremental program evolution.

PMBGA Probabilistic model-building genetic algorithm.

SHCLVND Stochastic hill climbing with learning by vectors of normal distributions.

UMDA Univariate marginal distribution algorithm.

Introduction

A general optimization problem may be defined by specifying (1) the set of all potential solutions to the problem and (2) a measure for evaluating the quality of each candidate solution. The task is to find the solution of highest quality. Despite the existence of specialized techniques for solving highly restricted classes of problems efficiently—such as linear and quadratic programming—it might be impossible (or impractical) to apply these techniques for two reasons. First, the problem that we are trying to solve might not fit any of the restricted classes of problems for which solutions are known. Second, even if the problem belongs to one of these restricted classes, it might be intractable or impractical to acquire sufficient knowledge to classify the problem properly. This gives rise to the growing interest in the design of adaptive optimization techniques capable of automatic discovery and exploitation of problem regularities to achieve efficient and scalable optimization of a broad class of important real-world problems (Kargupta, 1995).

Many adaptive optimization techniques use problem decomposition as the basic mechanism for reducing problem complexity. The basic idea of using problem decomposition for simplifying a problem is rather simple; if the problem can be decomposed into tractable subproblems, its difficulty can be significantly reduced, and instead of solving one big problem it is sufficient to approach several smaller, individually tractable subproblems. But how can we exploit problem decomposition in optimization?

Genetic and evolutionary computation (GEC) offers one class of methods capable of doing that by evolving a population of candidate solutions to the given problem using mechanics inspired by natural evolution and genetics. Specifically, the search in genetic and evolutionary algorithms is guided by two basic mechanisms: (1) selection and (2) variation. Selection biases the search toward high-quality solutions by making multiple copies of better solutions at the expense of their inferior competitors. Variation operators ensure exploration by creating new solutions based on the promising solutions found so far. Two variation operators are common in current GEC: (1) recombination or crossover and (2) mutation. Recombination creates new solutions by combining parts of promising solutions, whereas mutation perturbs promising solutions slightly. In this thesis, the focus is on genetic algorithms (Holland, 1975; Goldberg, 1989a), which use both crossover and mutation.

Genetic algorithms can solve problems decomposable into subproblems of bounded order in subquadratic or quadratic time measured by the number of candidate solutions that must be evaluated until reliable convergence to the optimum (Goldberg, Deb, & Clark, 1992; Harik, Cantú-Paz, Goldberg, & Miller, 1997; Mühlenbein & Schlierkamp-Voosen, 1993). However, traditional variation operators often fail at solving the given problem efficiently because they assume a *fixed* decomposition regardless of the problem. In the design of competent genetic algorithms (Goldberg, 1999b)—genetic algorithms that can solve hard problems quickly, accurately, and reliably—it is necessary that the algorithm be capable of both learning and exploiting a proper, nonmisleading problem decomposition.

Thesis Objectives

There are four primary objectives of this thesis:

(1) Design a competent optimization algorithm capable of learning and exploiting a proper problem decomposition by sampling and evaluating candidate solutions.

(2) Extend the proposed method to allow for the discovery and utilization of *hierarchical* decomposition as opposed to decomposition on only a single level.

(3) Design a class of difficult hierarchical problems that can be used to test the algorithms that attempt to exploit hierarchical decomposition.

(4) Test the developed algorithms on the designed class of problems and several real-world applications.

Specifically, the thesis proposes the Bayesian optimization algorithm (BOA), which uses Bayesian networks to model promising solutions and sample new candidate solutions. Theoretical and empirical evidence is provided to show that BOA is capable of solving hard problems decomposable into subproblems of bounded order in a scalable manner. Additionally, hierarchical BOA is proposed that extends BOA by incorporating important mechanisms necessary for scalable optimization of problems that cannot be decomposed into tractable subproblems on a single level but can be solved by a repeated decomposition down a number of levels. Hierarchical processing improves the algorithm and represents another useful strategy that can further reduce complexity of difficult problems.

A class of problems called hierarchical traps is then proposed to test the ability of optimizers to learn and exploit hierarchical decomposition. Hierarchical BOA passes the test and is shown to solve hierarchical traps and other hierarchical problems in a scalable manner. Finally, the thesis applies hierarchical BOA to two important classes of real-world problems from statistical physics and artificial intelligence. The experiments show that even without requiring any prior problem-specific knowledge about the structure of the problem or its properties, hierarchical BOA is capable of achieving competitive or better performance than other state-of-the-art methods specializing in solving the examined classes of problems.

Road Map

The thesis is divided into nine chapters. The first two chapters introduce genetic algorithms, provide motivation for the development of competent genetic algorithms, and introduce the basic concepts of probabilistic model-building genetic algorithms (PMBGAs). Chapters 3 and 4 present and analyze the Bayesian optimization algorithm (BOA), which is theoretically and empirically shown to solve problems of bounded difficulty in a scalable manner. Chapters 5 and 6 motivate the use of hierarchical decomposition for complexity reduction, propose a class of challenging problems for hierarchical optimizers, and extend the original BOA to solve difficult hierarchical problems. The thesis closes by presenting experimental results on two classes of real-world problems, discussing interesting topics for future work, and providing the conclusions. The following subsections present the content of each chapter in greater detail.

Chapter 1: From Genetic Algorithms to Probabilistic Models

Chapter 1 describes the basic genetic algorithm (GA) procedure and argues that in the design of competent black-box optimization techniques based on problem decomposition, there are several important lessons to learn from GA research. The chapter presents a simple GA simulation by hand to build some intuition regarding the dynamics of GAs. Additionally, the simulation provides motivation for the use of probabilistic models in guiding exploration more generally. The limitations of standard genetic algorithm approaches are then identified and discussed in the context of probabilistic and nonprobabilistic variation operators.

Chapter 2: Probabilistic Model-Building Genetic Algorithms

Chapter 2 surveys probabilistic model-building genetic algorithms, which guide exploration by building a probabilistic model of promising solutions and sampling the built model to generate new candidate solutions. Basic definitions are provided to present the algorithms from this class within a unified framework. The chapter focuses on methods working in a discrete domain; other representations are discussed briefly.

Chapter 3: Bayesian Optimization Algorithm

Chapter 3 proposes the Bayesian optimization algorithm (BOA), which uses Bayesian networks to learn a proper decomposition of the problem and sample new candidate solutions.

Next, the chapter describes the methods for learning and sampling multiply connected Bayesian networks. Both structural and parametrical learning are considered. The chapter finishes by presenting initial experimental results indicating good scalability of BOA.

Chapter 4: Scalability Analysis

To support the promising empirical results, Chapter 4 develops a scalability theory for BOA. As a measure of BOA's computational complexity, the chapter considers the total number of candidate solutions that must be evaluated until reliable convergence to the optimum. The total number of evaluations is computed by estimating an upper bound on the sufficient population size and an expected number of generations until convergence.

It turns out that if a problem can be decomposed into subproblems of bounded order, the number of evaluations until BOA converges to the optimum with high confidence grows subquadratically or quadratically with the size of the problem. The theory is verified by a number of experiments.

Chapter 5: Challenge of Hierarchical Difficulty

Chapter 5 discusses the use of hierarchical decomposition as opposed to decomposition on only a single level and argues that hierarchical decomposition is a powerful tool for reducing problem complexity. The chapter identifies important features that must be incorporated into the Bayesian optimization algorithm to allow the algorithm to exploit hierarchical decomposition effectively: (1) proper decomposition, (2) chunking, and (3) preservation of alternative solutions.

Next, the chapter proposes a class of challenging hierarchical problems called hierarchical traps. Hierarchical traps bound the class of problems solvable by optimization algorithms capable of automatic discovery and exploitation of hierarchical decomposition and can be used as a test bed for those algorithms. Hierarchical traps are extremely difficult if they are approached on a single level or with local operators; however, with the proper use of hierarchical decomposition, hierarchical traps can be solved in low-order polynomial time by evaluating only a subquadratic number of candidate solutions.

Chapter 6: Hierarchical Bayesian Optimization Algorithm

Chapter 6 first describes how to incorporate the important features of a successful hierarchical optimizer into BOA. Decomposition and chunking are incorporated by using local structures that ensure compact representation of Bayesian networks for problems with high-order and complex interactions. Preservation of alternative solutions is guaranteed by using a procedure called restricted tournament replacement for updating the population of candidate solutions using newly generated solutions.

Bayesian networks with local structures and restricted tournament replacement form the basis of the *hierarchical Bayesian optimization algorithm* (hBOA). Hierarchical BOA is shown to solve the proposed class of challenging hierarchical problems in a scalable manner.

The scalability theory of Chapter 4 is then extended to provide a bound on the number of evaluations required for solving hierarchical problems, which is again verified with empirical results.

Chapter 7: Hierarchical BOA in the Real World

Chapter 7 applies hierarchical BOA to two important classes of real-world problems: (1) two- and three-dimensional Ising spin-glass systems, and (2) maximum satisfiability (MAXSAT).

First, the chapter considers two-dimensional Ising spin-glass systems with cyclic boundary conditions. The scalability of hierarchical BOA is analyzed by testing the algorithm on a number of randomly generated spin-glass instances of varying problem sizes. The performance of hierarchical BOA is compared to that of state-of-the-art methods for spin-glass optimization. Next, local search is incorporated into hierarchical BOA to further improve its performance and the results with and without local search are compared. Finally, hierarchical BOA with local search is applied to an array of three-dimensional spin-glass systems. The performance of hierarchical BOA with local search is shown to be competitive or better than that of the state-of-the-art methods. Furthermore, hierarchical BOA with local search is shown to provide a scalable solution to those spin-glass instances that cannot be solved by other methods included in the comparison.

Second, the chapter considers instances of maximum satisfiability (MAXSAT). A hybrid combining hierarchical BOA with local search is tested on a number of difficult MAXSAT instances. Again, the scalability of the algorithm is analyzed by varying the problem size and the performance of hierarchical BOA is compared to that of state-of-the-art methods for solving MAXSAT. Since MAXSAT is NP-complete, it can be expected that the time complexity of hierarchical BOA on MAXSAT grows exponentially. Nonetheless, hierarchical BOA is shown to provide a robust solution to a wide range of MAXSAT instances and outperform other algorithms for solving MAXSAT in many cases.

Chapter 8: Future Work

Chapter 8 discusses important topics of future research. There are three important lines of future research:

(1) **Enhance the efficiency.** Make BOA and hierarchical BOA more efficient by parallelization, evaluation relaxation, hybridization, time continuation, and utilization of prior problem-specific

knowledge.

(2) **Extend the applicability.** Extend the applicability of hierarchical BOA by combining it with advanced techniques for multiobjective optimization and applying hierarchical BOA to problems where the candidate solutions are not represented by fixed-length strings over a finite alphabet.

(3) **Extend the theory.** Extend the developed scalability and convergence theory of BOA and hierarchical BOA.

Chapter 9: Conclusions

Chapter 9 concludes the thesis and lists its most important consequences.

Chapter 1

From Genetic Algorithms to Probabilistic Models

Genetic algorithms (GAs) (Holland, 1975; Goldberg, 1989a) are stochastic optimization methods inspired by natural evolution and genetics. Over the last few decades, genetic algorithms have been successfully applied to many problems of business, engineering, and science (Goldberg, 1999a). Because of their operational simplicity and wide applicability, genetic algorithms are now becoming an increasingly important area of computational optimization.

There are several concepts of genetic algorithms that we will use in the design of a competent black-box optimization method capable of exploiting problem decomposition: population-based search, exploration by combining bits and pieces of promising solutions, and facetwise genetic algorithm theory. The purpose of this chapter is to review the basic GA procedure and build some intuition regarding the dynamics of GAs. Additionally, the chapter shows that traditional variation operators of genetic algorithms can be approximated by learning and sampling a probabilistic model of promising solutions. Finally, the chapter identifies important conditions of GA success and discusses limitations of traditional GAs.

The chapter starts by discussing the general form of a black-box optimization problem and the need for a bias to solve such problems efficiently. Section 1.2 describes the basic genetic algorithm procedure as one of the representatives of black-box optimization techniques. Section 1.3 presents a sample GA run to build some intuition regarding the mechanics and limitations of genetic algorithms. The simulation motivates the use of probabilistic recombination, which replaces traditional recombination by building a probabilistic model of promising solutions and sampling the learned model to generate new candidate solutions. Section 1.4 presents one approach to probabilistic recombination and relates the presented approach to the traditional one. Section 1.5 discusses selectorecombinative search and its limitations. Additionally, the section defines a class of problems of bounded difficulty that challenge traditional GAs. Finally, Section 1.6 summarizes the chapter.

1.1 Black-box Optimization

An optimization problem may be defined by specifying (1) the set of all potential solutions to the problem and (2) a measure to evaluate performance of each candidate solution with respect to the objective. The goal is to find a solution or a set of solutions that perform best according to the specified measure. For example, in the maximum satisfiability of logic formulas in conjunctive normal form, each candidate solution can be represented by a list of truth values for all variables and the quality of a solution can be defined as the number of satisfied clauses. In the design of an aircraft wing, a solution can be represented by a set of parameters that specify the shape of the wing and the performance of each parameter set can be determined by an experiment in a wind tunnel. In the design of an algorithm for playing chess, a solution can be represented by a set of condition-action rules, and the performance of each such set can be defined as the portion of games won against other candidate strategies.

In black-box optimization, there is no information about the relation between the performance measure and the semantics of the solutions. The only way of learning something about the problem at hand is to *sample new candidate solutions and evaluate them*. The task of finding the best solution to a problem that we know nothing about is extremely difficult. To illustrate the difficulty of black-box optimization, imagine you are asked to implement a program in an unknown programming language given only the syntax of the language and a procedure that evaluates how good each valid program is.

Black-box optimization procedures iteratively sample candidate solutions and use the results of the evaluation of those candidate solutions to sample new solutions. There are many ways to sample new candidate solutions. A hill climber, for instance, explores the neighborhood of the current best candidate solution by perturbing the solution in some way. If the perturbations reveal a candidate solution that is better than the current one, the better solution serves as the starting point for further exploration. The way in which a particular optimization method samples new candidate solutions and exploits the result of the evaluation of those new solutions limits the class of problems that the method can solve in an efficient manner. For instance, using local operators in the hill climber limits the applicability of the algorithm to those problems that contain only a few basins of attraction, or those problems where an approximate location of the global optimum is known in advance.

1.2 Genetic Algorithms

Genetic algorithms (GAs) (Holland, 1975; Goldberg, 1989a; Mitchell, 1996) approach black-box optimization by evolving a population of candidate solutions with the operators inspired by natural evolution and genetics. Maintaining a population of solutions, as opposed to a single solution, has several advantages. Using a population allows a simultaneous exploration of multiple basins of attraction. Additionally, a population allows for statistical decision making based on the entire sample of promising solutions even when the evaluation procedure is affected by external noise.

Genetic algorithms pose no significant prior restrictions on the representation of candidate solutions or the performance measure. Representations can vary from binary strings to vectors of real numbers, to permutations, to production rules, to schedules, and to program codes. Performance measures can be based on a computer procedure, a simulation, an interaction with a human, or some combination of the above. Additionally, evaluation of each candidate solution can be affected by external noise. Nonetheless, for the sake of simplicity, the rest of this chapter assumes that solutions are represented by binary strings of fixed length and that the performance of each candidate solution is represented by a real number called *fitness*. The task is to find the binary string with the highest fitness.

The first population of candidate solutions is usually generated randomly with a uniform distribution over all possible solutions. Each iteration starts by selecting a set of promising solutions from the current population based on the performance of each solution. Various selection operators can be used, but the basic idea of all selection methods is the same—*make more copies of solutions that perform better at the expense of solutions that perform worse.* Two selection methods will be

(a) One-point crossover. (b) Uniform crossover.

Figure 1.1: An illustrative example of two common two-parent crossover operators. In one-point crossover, the tails are exchanged after a randomly chosen position. In uniform crossover, the bits on each position are exchanged with probability 50%.

used in this thesis: tournament selection and truncation selection. Tournament selection iteratively selects one solution at a time by first choosing a random subset of candidate solutions from the current population and then selecting the best solution out of this subset. Random tournaments are repeated until there are sufficiently many solutions in the selected population. The size of the tournaments determines selection pressure—the larger the tournaments, the higher the pressure on the quality of each solution. Truncation selection selects the best $1/s$ of the current population, where s determines selection pressure. To select a population of the same size as the population before selection is, s copies of selected solutions can be created.

Once the set of promising solutions has been selected, new candidate solutions are sampled by applying recombination (crossover) and mutation to the promising solutions. Recombination combines subsets of promising solutions by exchanging some of their parts; mutation perturbs the recombined solutions slightly to explore their immediate neighborhood. Most of the commonly used crossover operators combine partial solutions between pairs of promising solutions with a specified probability. For example, one-point crossover first randomly selects a single position in the two strings and exchanges the bits on all the positions after the selected one (see Figure 1.1(a)). On the other hand, uniform crossover exchanges bits on each position with probability 50% (see Figure 1.1(b)). For binary strings the so-called bit-flip mutation is usually used. Bit-flip mutation proceeds by flipping each bit with a fixed probability. The probability of flipping each bit is quite small, so only small changes are expected to occur. Recombination is the primary source of variation in most GAs; these GAs are often referred to as *selectorecombinative GAs*.

After applying crossover and mutation to the set of promising solutions, the population of new candidate solutions replaces the original one and the next iteration is executed, unless the

Term	Alternative name(s)
candidate solution	individual, chromosome, string
decision variable	variable, locus
value of decision variable	bit, allele
performance function	fitness function
performance value	fitness, fitness value
population of promising solutions, $S(t)$	parent population, parents, selected solutions
population of new solutions, $O(t)$	offspring population, offspring
iteration	generation

Table 1.1: Common GA terminology.

termination criteria are met. For example, the run can be terminated when the population converges to a singleton, the population contains a good enough solution, or a bound on the number of iterations has been reached.

In explaining and discussing dynamics and limitations of GAs, several additional terms will be used. A *partial solution* denotes specific bits on a subset of string positions. Competitors of a partial solution are all partial solutions that are specified in the same positions as the partial solution but that differ from the partial solution in at least one of the specified bits. A *partition* denotes a subset of string positions. The partition corresponding to a particular partial solution is defined as the subset of string positions that are specified in the partial solution.

Common GA terminology is listed in Table 1.1. In the rest of the thesis, we will use terminology consistent with that listed in the above table.

1.3 Simulation: Onemax and Population-wise Uniform Crossover

By applying selection and crossover to the population of candidate solutions, genetic algorithms can process a large number of partial solutions in parallel. The processing of partial solutions consists of (1) deciding between competing partial solutions in the same partition, (2) making more copies of those partial solutions that perform better than their competitors in the same partition, and (3) combining partial solutions to ensure effective exploration of the search space. Selection ensures that superior partial solutions will be given more copies in the selected population. Crossover ensures that partial solutions propagated by selection are combined to explore new regions in the search space.

This section presents a simple GA simulation by hand on a onemax problem. The purpose of presenting the simulation is twofold. First, the simulation attempts to build some intuition regarding the processing of partial solutions in GAs. Second, the simulation motivates the use of probabilistic models to guide the sampling of new candidate solutions in GAs.

1.3.1 Simulation

Onemax is defined as the sum of the bits in the input binary string:

$$f_{onemax}(X) = \sum_{i=0}^{n-1} X_i, \tag{1.1}$$

where $X = (X_0, \ldots, X_{n-1})$ is the input string of length n. Onemax is a simple linear function with the optimum in the string of all ones. The simulation considers a 5-bit onemax and a population of size $N = 6$. Binary tournament selection is used to select promising solutions. Population-wise uniform crossover is used to create new candidate solutions by shuffling the bits on each position in the selected set of promising solutions. No mutation is used. The simulation is based on an actual GA run.

Figure 1.2 shows the first two generations of the GA simulation. The initial population of candidate solutions is generated at random. Next, tournament selection is applied to create a set of promising solutions. Six tournaments take place, and the winners of these six tournaments form the population of promising solutions. Population-wise uniform crossover is then applied to the population of selected solutions by shuffling the bits on each position among the selected solutions. The resulting (shuffled) set of solutions then replaces the original population.

In both generations of the simulation, the average fitness of the new population of candidate solutions (the offspring) is greater than the average fitness of the population before selection. The reason for the increased fitness lies in the way the GA processes the population. Since the solutions with more ones have higher fitness than those with fewer ones, selection should increase the number of ones in the population. Crossover neither creates nor destroys any bits in the population. Therefore, the population after applying selection and crossover will contain more ones than the original population. Since fitness increases with the number of ones in a solution, overall

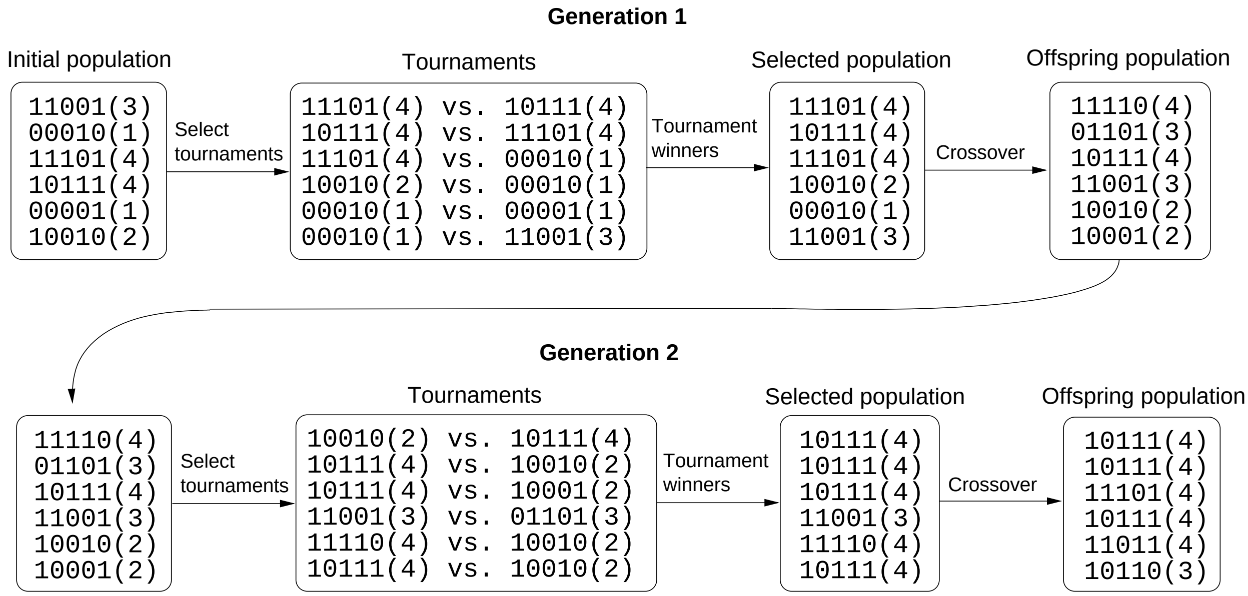


Figure 1.2: The simulation of the GA with a population of size $N = 6$, population-wise uniform crossover, and binary tournament selection on onemax of size 5. The fitness of each solution is shown in parentheses.

fitness is expected to increase over time.

Ideally, every generation should increase average fitness (unless no improvement is possible). However, an increase in average fitness tells only half the story. The same increase in the fitness of the population in the first generation would be achieved by using no recombination at all. However, by applying selection without recombination, the best solution in the population would soon overtake the entire population and there would be no chance of finding the real optimum. Even more importantly, only initial solutions would be considered. Since the initial population in GAs is generated at random, the GA with selection only would perform no better than random search or total enumeration with a limited sample. For an efficient and reliable search for the optimum, there must be an additional mechanism besides selection that ensures effective exploration of new solutions. This can be achieved by combining superior partial solutions propagated by selection.

From a mixing point of view, population-wise uniform crossover is the most powerful recombination one can imagine. It completely shuffles the bits on each position. As the proportion of ones on each position increases, the chances of finding the optimum by shuffling increase over time. For instance, if the proportion of ones on each position reaches 99%, there is almost no way of avoiding the optimum if crossover with a sufficient exploratory power is used.

In subsequent generations, the selectorecombinative search further increases the quality of candidate solutions. In the simulation, the optimum is found in the third generation. Once the optimum has been found, it quickly (in two more generations) takes over the entire population.

1.4 Population-wise Uniform Crossover as a Probabilistic Model

Recall that population-wise uniform crossover considers each position independently and creates a new population of candidate solutions by shuffling bits on each position across the population of promising solutions. Consider using the following *probabilistic* recombination instead:

1. Compute the probability of a 1 at each position in the selected set of promising solutions. The probability of a 1 at the ith position is denoted by $p(X_i = 1)$ or simply p_i.

2. Generate new candidate solutions by setting the bit at the ith position of each solution to be 1 with the probability p_i; otherwise, set the bit to 0. The order in which the bits in each solution are generated does not matter.

3. Repeat step 2 until enough candidate solutions have been generated.

We will refer to the above crossover operator as *probabilistic uniform crossover*. There is an important difference between population-wise uniform crossover and probabilistic uniform crossover. Population-wise uniform crossover manipulates the selected population of promising solutions to create a new population of candidate solutions. On the other hand, probabilistic uniform crossover first computes relevant statistics from the selected population, then *discards the population*, and uses only the acquired statistics to generate new solutions. Nonetheless, the expected outcome of both operators is the same.

Figure 1.3 shows the first two generations of the simulation using probabilistic uniform crossover. As discussed above, "recombination" proceeds by computing the probability of 1 for each position of the selected population of promising solutions, and then using the computed probabilities to sample new candidate solutions. Although the proportion of 1s in each position is expected to remain the same after sampling the probability vector, a certain amount of noise is introduced with sampling; however, as the population grows, these effects become negligible. In any case, the proportion

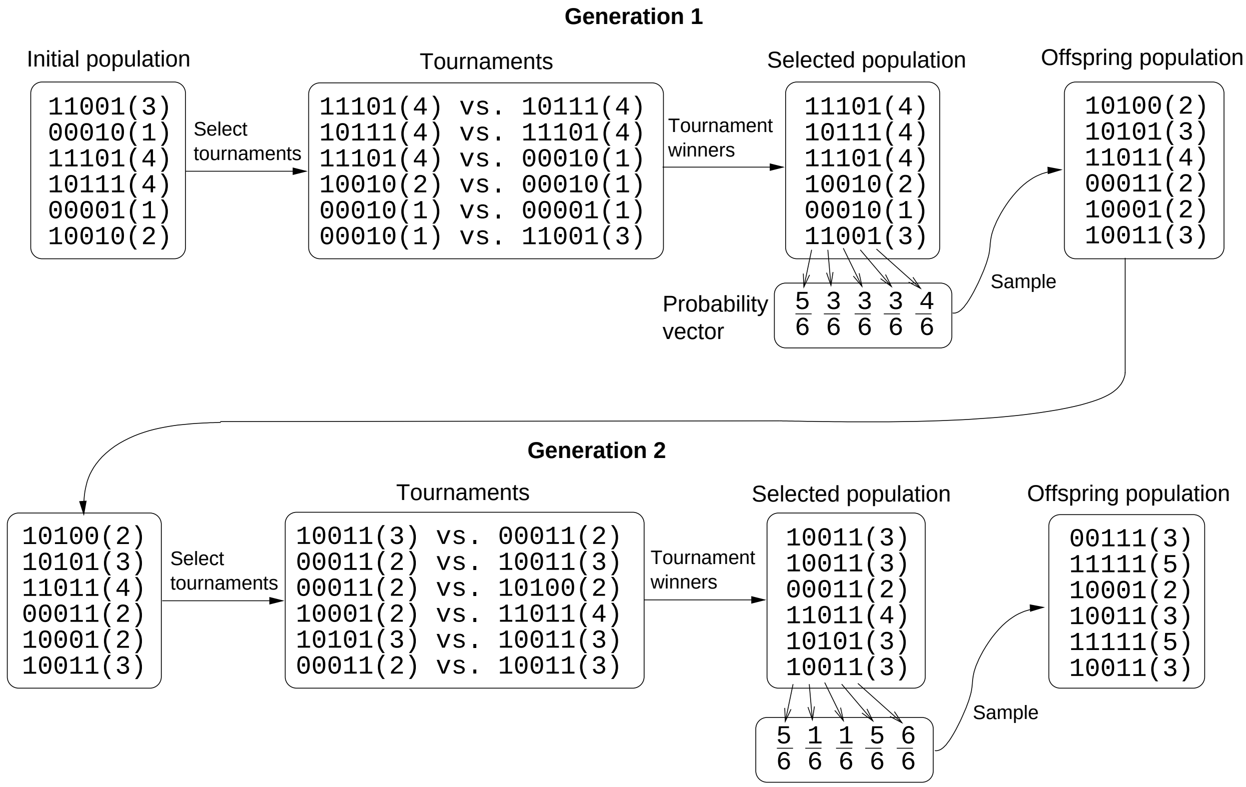


Figure 1.3: The simulation of the GA with probabilistic uniform crossover. The simulation uses a population of size $N = 6$, and binary tournament selection on onemax of size 5. The fitness of each solution is shown in parentheses.

of 1s continues to increase over time along with the average fitness. Using probabilistic uniform crossover, the simulation finds the optimum already in the second generation. After additional two generations, the entire population converges to the optimum.

Probabilistic uniform crossover suggests an alternative approach to processing promising solutions. Rather than combining the bits and pieces of solutions by directly manipulating the set of promising solutions, one can extract important statistical information about the population of promising solutions, and use the resulting probabilistic model to sample new solutions. In probabilistic uniform crossover, only bitwise probabilities are extracted; however, it is straightforward to modify the method to look at pairs of bits, for instance. This possibility raises an interesting question: When does the GA need more complex probabilistic models? The following section attempts to answer that question.

1.5 What if Single Bits are Misleading?

Some problems—such as onemax—can be solved efficiently by calculating and exploiting probabilities independently bit by bit. However, what happens if the value of the ith bit of the global optimum is 1, but the average fitness of solutions with 1 on that position is lower than the average fitness of solutions with 0 on that position? More generally, what happens if the average fitness of partial solutions of order one *misleads* selection away from the optimum? Unfortunately, in that case both population-wise uniform crossover and probabilistic uniform crossover will fail in finding the optimum.

The reason for the failure of GAs with population-wise uniform crossover is that due to misleading statistics over single bits, the number of partial solutions contained in the optimum will *decrease* after selection and recombination. As an example, consider the trap function of order 5 (Ackley, 1987; Deb & Goldberg, 1991), defined as

$$trap_5(u) = \begin{cases} 5 & \text{if } u = 5 \\ 4 - u & \text{otherwise} \end{cases}, \tag{1.2}$$

where u is the number of ones in the input string. See Figure 1.4 to visualize the trap function. In trap, the average fitness of all solutions containing 1 on the first position is 1.375, while the average fitness of all solutions that contain 0 on the first position is 2. Therefore, if we focus on one bit only, a 0 seems to be a better choice than a 1 is, even if the population was infinite. However, the optimum of trap is in 11111. Therefore, using statistics over solutions of order 1 leads *away from the optimum* and after selection and recombination, the proportion of ones in the population decreases. The same situation occurs in subsequent generations (although the average fitnesses change a little), and the GA converges to the local optimum in 00000.

What if we use statistics over pairs of bits? The average fitness of block 00 is 2.5, the average fitness of 01 and 10 is 1.5, and the average fitness of 11 is 1.25. Again, the statistics lead to 00000, while the actual optimum is in 11111. The situation remains the same even if we consider blocks of size 3 or 4 bits. In all those cases, single-bit statistics leads to the local optimum 00000. For statistics to lead in the right direction, blocks of size 5 bits must be considered; in this case, the

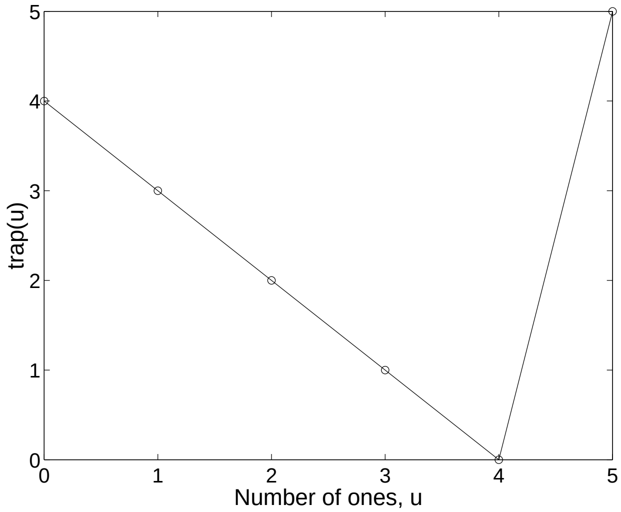


Figure 1.4: The trap function of order 5. The value of the trap function depends on the number of ones in the input string. There are two optima of the trap; the global optimum in 11111 and the local optimum in 00000. The average fitness over any subset of bits in the trap leads to the local attractor, unless all the bits are considered together.

fitness of 11111 is 5 and is therefore greater than the fitness of any alternative instantiation of 5 bits.

The remainder of this chapter looks at several important conditions that must be satisfied for effective exploitation of selectorecombinative bias. First, the notions of building blocks and problem decomposition are discussed. Next, the remaining conditions are examined briefly. Finally, the need for learning an appropriate decomposition of the problem motivates linkage learning, which focuses on the design of adaptive recombination techniques capable of adapting to the problem at hand to enable exploitation of a proper problem decomposition.

1.5.1 Building Blocks

Although the trap function is an artificial example, it suggests that there exist functions for which processing bits independently does not work. For success of selectorecombinative search, it is necessary that recombination effectively processes partial solutions that are contained in the optimum, and that are superior to their competitors over the course of the GA run. We call those superior partial solutions of the optimum *building blocks* (Holland, 1975; Goldberg, 1989a; Goldberg, 2002). Since building blocks are superior to their competitors, selection will increase the proportion of building blocks in a population. By combining promising solutions without disrupting building blocks (disruption can eliminate a building block) the number of copies of partial solutions contained in the optimum increases, and the chances of hitting the optimum (with strong enough recombination) grow over time. Of course, it is necessary that building blocks cover the entire optimum.

Note that the fitness of a solution containing a particular partial solution does not depend only on the partial solution itself but also on the values on the remaining positions in the solution. The values on the positions outside a partial solution define the *context* of the partial solution (Goldberg, 2002). In the initial population, contexts of a partial solution are distributed uniformly. As the population evolves, the distribution of contexts of each partial solution changes as well. Building blocks must be superior to their competitors in all distributions of contexts that can be expected to occur over the GA run.

The above definition of building blocks is based on the concept of *minimal sequentially superior building blocks of the optimum* (Goldberg, 2002). Kargupta (1995) describes a different view on problem decomposition and its exploitation by defining a class of problems that are order-k delineable within the SEARCH (search envisioned as relation and class hierarchizing) framework. SEARCH sees GAs as inductive procedures that try to sample solutions near the optimum given a sample of promising solutions. Another view on problem decomposition can be found in Mühlenbein and Mahnig (1999), who define sufficient conditions for decomposing a problem based on distribution factorization.

1.5.2 Building Blocks and Nonlinearities in Probabilistic Models

Since the primary focus of this thesis is on probabilistic recombination operators that extend probabilistic uniform crossover to deal with more complex problems, it is important to relate the notion of building blocks with the probabilistic approach. In this case, the concept of building blocks is important to identify partial solutions that must be treated by a probabilistic model as an intact block. Probabilities for each partition corresponding to a proper building-block decomposition of the problem must be extracted from the population of promising solutions and the new solutions must be generated using those probabilities.

However, most probabilistic approaches depart from the notion of building blocks and focus on nonlinearities in the problem. Clearly, not every nonlinearity among a group of bits means that the group must be considered as an intact block. However, it is difficult to distinguish between the nonlinearities that must be considered and those that need not be considered. For instance, if the optimum in onemax was given an additional reinforcement, a nonlinearity would be introduced but the problem could still be solved by considering each bit independently. On the other hand, the trap function introduces nonlinearities that must be considered to avoid deception. If any subset of positions in a trap is considered independently of other positions in the trap, selectorecombinative search will simply go in the wrong direction. Probabilistic approaches take the safe route by assuming that every nonlinearity must be taken into account. Population size and other factors might not permit all nonlinearities to be considered, so sometimes only the strongest nonlinearities can be identified and processed.

Identification of important statistics for processing populations of promising solutions is closely related to feature extraction in machine learning. While in GAs the task is to find important statistics to converge to the optimum, in feature extraction the task is to process a given set of features to ensure that the target concept (function) can be learned using a specified class of concepts. Although this connection is interesting, it is beyond the scope of this thesis to discuss it in more detail.

1.5.3 Some Additional Requirements

There are additional requirements for the above scheme to work. For effective processing of building blocks, the population must be large enough to ensure that there is a sufficient supply of raw building blocks in the initial population. Additionally, recall that the fitness of solutions containing a particular building block depends on building block's context. The population must be large enough for the effects of context to average out so that selection creates more copies of each building block at the expense of its competitors. For a more thorough overview of the important issues to consider for the successful application of GAs, see Goldberg's seven conditions of GA success (Goldberg, Deb, & Clark, 1992; Goldberg, 1994; Goldberg, 2002). We will return to many of these issues in chapter 4.

The necessity of an adequate initial supply of building blocks suggests that the population size must grow at least exponentially with the order of the largest building block in the problem. Therefore, it is necessary to consider as short building blocks as possible, while maintaining a cover for the entire solution. The class of problems that can be solved by considering building blocks of order bounded by a constant is often referred to as *problems of bounded difficulty* or *boundedly difficult problems*. It can be shown that selectorecombinative GAs can solve problems of bounded difficulty (no matter what the difficulty is) in subquadratic or quadratic time with respect to the number of fitness evaluations (Goldberg, Deb, & Clark, 1992; Mühlenbein & Schlierkamp-Voosen, 1993; Harik, Cantú-Paz, Goldberg, & Miller, 1997). Of course, effective processing of building blocks is necessary to ensure such efficient performance.

1.5.4 Decompositional Bias and Linkage Learning

Problem decomposition according to building blocks defines the optimization bias exploited by a proper combination of selection and recombination. In other words, decomposition defines what regularities are exploited by GAs. These regularities can be represented by a set of dependencies and independencies between groups of variables according to a proper, nonmisleading, problem decomposition. Recombination must respect these dependencies and independencies and juxtapose the building blocks without their disruption. At the same time, selection must ensure their growth.

The line of research that focuses on identification and effective processing of building blocks is sometimes called *linkage learning*. The primary goal of linkage learning is to design methods that are capable of (1) an automatic identification of building blocks and (2) effective processing of the identified blocks. There are a number of approaches that attempt to modify GA operators or representation to ensure that the correct building blocks are processed. For an overview, see Harik (1997).

More recently, a number of probabilistic crossover operators have been proposed to deal with the same problem. While linkage-learning approaches based on traditional genetic algorithms attempt to directly identify building blocks in some way, probabilistic linkage-learning algorithms attempt to learn the nonlinearities in a problem and to sample new solutions based on sufficient statistics that cover all, or at least most, of the identified nonlinearities in the problem. The next chapter reviews the most influential approaches that use probabilistic models to model promising solutions found so far and sample new candidate solutions.

1.6 Summary

This chapter introduced the basic procedure of genetic algorithms and argued that probabilistic-model building and sampling can be used as an alternative to traditional recombination in genetic algorithms. A summary of the key points of this chapter follows:

- An optimization problem may be defined by specifying (1) the set of all potential solutions to the problem and (2) a measure to evaluate performance of each candidate solution with respect to the objective. The goal is to find a solution or a set of solutions that perform best with respect to the specified measure.

- Black-box optimization methods approach an optimization problem without requiring any problem-specific knowledge in advance. The only way of learning something about the problem at hand is to sample and evaluate new candidate solutions.

- Genetic algorithms (GAs) approach black-box optimization by evolving a population of candidate solutions to a given problem. The sampling in GAs is typically guided by (1) selection, (2) recombination, and (3) mutation. Selection biases the search to solutions that look promising based on their performance. Crossover combines bits and pieces of promising solutions found so far, whereas mutation perturbs promising solutions slightly.

- The performance of each candidate solution in a GA population can be determined by a computer procedure, a simulation, an interaction with a human, or a combination of the above. The evaluation can be affected by external noise. Often, the performance of each candidate solution is expressed by one real number, called fitness. The better the solution, the higher its fitness.

- An alternative way of combining promising solutions is to build and sample a probabilistic model of these solutions. For example, population-wise uniform crossover can be simulated by computing the proportion of ones on each position in a selected set of promising solutions, and generating new candidate solutions according to these proportions.

- For a successful application of the selectorecombinative bias of GAs, it is important that GAs process partial solutions that are both superior to their competitors and contained in the optimum. These partial solutions are called building blocks. The building-block decomposition can be expressed in the form of dependencies and independencies, or by a set of building blocks that cover the entire problem. In any case, it is important that statistics over the subproblems in the decomposition lead to the optimum.

- The class of problems that can be decomposed into subproblems of bounded order is usually referred to as *problems of bounded difficulty* or *boundedly difficult problems*.

- Linkage learning focuses on the design of methods that can identify, propagate, and combine building blocks effectively. In terms of probabilistic recombination, linkage learning methods attempt to find relevant statistics leading to the global optimum. The purpose of linkage learning is to design robust and scalable methods for solving problems of bounded difficulty.

Chapter 2

Probabilistic Model-Building Genetic Algorithms

Probabilistic model-building genetic algorithms (PMBGAs) (Pelikan, Goldberg, & Lobo, 2002) use probabilistic modeling of promising solutions to guide the exploration of the search space instead of using the traditional recombination and mutation operators of simple GAs. There are many ways of estimating the probability distribution of promising solutions and each PMBGA deals with the problem of estimating distributions in its own way. This chapter reviews most influential PMBGAs and discusses their strengths and weaknesses.

The chapter starts by describing the general procedure of PMBGAs, which can be seen as a template where one simply fills in methods of choice for learning and sampling probabilistic models. Section 2.2 motivates the use of probability distributions in PMBGAs by presenting an example of how a simple univariate distribution can be used to solve a linear problem and how the simple approach can be generalized to more complex problems. The example leads to an overview of the approaches that have shaped the field of using probabilistic models in genetic and evolutionary computation. Section 2.3 introduces the family of PMBGAs for optimizing problems in the domain of fixed-length strings over a finite alphabet. Binary strings are used throughout most of the section, but most approaches can be generalized to any finite alphabet in a straightforward manner. Section 2.4 presents some of the approaches for optimizing problems in other than discrete domains. Section 2.5 concludes the chapter with a brief summary.

Probabilistic Model-Building Genetic Algorithm (PMBGA)

(1) set $t \leftarrow 0$
randomly generate initial population $P(0)$

(2) select a set of promising strings $S(t)$ from $P(t)$

(3) estimate the probability distribution of the selected set $S(t)$

(4) generate a set of new strings $O(t)$ according to the estimate

(5) create a new population $P(t+1)$ by replacing some strings from $P(t)$ with $O(t)$
set $t \leftarrow t+1$

(6) if the termination criteria are not met, go to (2)

Figure 2.1: Pseudo-code of the general PMBGA procedure.

2.1 General PMBGA Procedure

The previous chapter presented a simple probabilistic recombination operator that builds and samples a probabilistic model of promising solutions as opposed to using traditional recombination and mutation of GAs. Probabilistic model-building genetic algorithms (PMBGAs) build on this idea and use the estimation of a true distribution of promising solutions to guide the search.

The general procedure of PMBGAs is similar to that of GAs (see Section 1.2). The initial population of PMBGA is generated at random. In each iteration, promising solutions are first selected from the current population of candidate solutions. The true probability distribution of the selected solutions is then estimated. New candidate solutions are then generated by sampling the estimated probability distribution. The new solutions are then incorporated into the original population, replacing some of the old ones or all of them. The process is repeated until the termination criteria are met. The pseudo-code of the PMBGA procedure is shown in Figure 2.1.

The difference between PMBGAs and traditional GAs lies in the way PMBGAs process a population of promising solutions to generate new candidate solutions. Instead of performing crossover on pairs of selected solutions with a certain probability and then applying mutation to each of the resulting solutions, the following two steps are performed:

1. **Model building.** A probabilistic model of promising solutions is constructed.

2. **Model sampling.** The constructed model is sampled to generate new solutions.

PMBGAs differ in how they cope with the above two steps and in whether they incorporate special selection or replacement mechanisms for processing the populations of solutions. In the literature, PMBGAs are also called *estimation of distribution algorithms* (EDAs) (Mühlenbein & Paaß, 1996), and *iterated density estimation algorithms* (IDEAs) (Bosman & Thierens, 2000a).

The following section motivates the use of probability distributions by presenting two example problems and the PMBGA operators that can solve the presented problems efficiently and reliably. The section briefly discusses fundamental questions of a successful application of PMBGAs and motivates a decomposition of PMBGAs according to the complexity of models they employ.

2.2 Motivation

The previous chapter indicated that the purpose and mechanics of GAs and PMBGAs are very similar. This section supports this claim by providing two simple examples that motivate the use of probability distributions to guide the search in GAs. The first example focuses on onemax and probabilistic uniform crossover from the previous chapter. The second example shows how to extend the methodology to cope with more complex problems, such as composed traps of order k.

2.2.1 Onemax and Probabilistic Uniform Crossover

Recall that onemax is defined as the sum of bits in the input binary string (see Equation 1.1). Since in onemax the contribution of each bit is independent of the values at the remaining positions, one way of decomposing the problem properly is to assume mutual independence of all the variables in the problem. Consequently, any crossover can be used without introducing disruption of important building blocks. As discussed above, to maximize the mixing, recombination can create new solutions by shuffling the values on each position by population-wise uniform crossover introduced in the previous chapter.

The previous chapter showed that population-wise uniform crossover can be simulated by probabilistic uniform crossover. In the language of probability distributions, probabilistic uniform crossover approximates the distribution of promising solution by a *univariate marginal distribution*:

$$p(X) = \prod_{i=0}^{n-1} p(X_i),$$

where $X = (X_0, \ldots, X_{n-1})$ is the vector of random variables corresponding to string positions; $p(X)$ is the probability of the vector X; and $p(X_i)$ is the probability of the variable X_i corresponding to the ith string position. To learn a univariate probability distribution, the set of promising solutions is parsed to determine the probability of 1 at each position. If a variable can obtain more than two values, one probability must be computed for each value the variable can obtain; of course, one of the probabilities can be omitted, because the sum of the probabilities for each variable must be equal to 1. To sample a univariate distribution, the value of each variable is generated according to the learned probabilities; in the binary case, ith position is set to 1 with probability $p(X_i = 1)$, otherwise the position is set to 0.

Using the above distribution estimate, it can be shown (Thierens & Goldberg, 1994; Harik, Cantú-Paz, Goldberg, & Miller, 1997; Mühlenbein & Schlierkamp-Voosen, 1993) that the PMBGA converges to a solution with a fixed probability of getting each bit right in $O(n)$ fitness evaluations (assuming an adequate population size). If the accuracy increases with the problem size (requiring a constant number of mistaken bits overall), the overall time complexity is $O(n \ln n)$.

2.2.2 Composed Traps and Probabilistic Building-Block Crossover

Let us now define the class of composed trap functions of order k. The basic structure of composed traps is simple. The entire string is first partitioned into nonoverlapping partitions of k bits each. The partitioning is fixed during the entire optimization, but it is important that we do not assume anything about the positions of the variables corresponding to different partitions. We denote the positions of the bits corresponding to the ith partition by indices from $b_{i,0}$ to $b_{i,k-1}$. A composed trap function of order k is then defined as

$$f_{trap,k}(X) = \sum_{i=0}^{\frac{n}{k}-1} trap_k(X_{b_{i,0}} + \ldots + X_{b_{i,k-1}}). \tag{2.1}$$

Each block $(X_{b_{i,0}}, \ldots, X_{b_{i,k-1}})$ contributes to the fitness through a general trap function of order

k (Ackley, 1987; Deb & Goldberg, 1991) defined as

$$trap_k(u) = \begin{cases} f_{high} & \text{if } u = k \\ f_{low} - u\frac{f_{low}}{k-1} & \text{otherwise} \end{cases} \tag{2.2}$$

where u denotes the number of ones in the input block of k bits. The trap function of order 5 presented in the previous chapter is a special case of the general trap function with $k = 5$, $f_{high} = 5$, and $f_{low} = 4$ (see Figure 1.4 in the previous chapter to visualize the special case). The trap function has one global optimum in the string of all ones and a local optimum in the string of all zeroes. For a large enough ratio f_{low}/f_{high} (see Deb and Goldberg (1991) for sufficient conditions), an important feature of traps is that the average fitness of any block of bits of order lower than k misleads the algorithm to the local optimum (recall the discussion in the previous chapter).

Composed trap functions contain a number of traps, which are mutually independent. Traps cannot be further decomposed; a proper decomposition should therefore include dependencies among the positions in each partition and independencies between the positions in different trap partitions. Let's assume that the algorithm is capable of discovering where the trap partitions are; in other words, let's assume that the algorithm is capable of learning linkage. To prevent the disruption of trap partitions, recombination should never exchange only part of a partial solution in any trap partition, it should exchange either the entire partial solution or nothing. To maximize the mixing, recombination should exchange as many partial solutions as possible. This results in *population-wise building-block crossover*, which shuffles entire blocks (defined by indices $b_{i,j}$ as described above) similarly as population-wise uniform crossover shuffles single bits.

Analogously to the single-bit case, population-wise building-block crossover can be approximated by selecting partial solutions for each trap partition with the probability proportional to the partial solution's frequency in the selected set of promising solutions. It the language of probability distributions, new solutions can be generated by sampling the following distribution:

$$p(X) = \prod_{i=0}^{\frac{n}{k}-1} p(X_{b_{i,0}}, \ldots, X_{b_{i,k-1}}),$$

where $p(X_{b_{i,0}}, \ldots, X_{b_{i,k-1}})$ denotes the probability of a partial solution $(X_{b_{i,0}}, \ldots, X_{b_{i,k-1}})$ corresponding to the ith trap partition. We refer to the operator based on the above distribution estimate as *probabilistic building-block crossover*.

As in the linear case, it can be shown that PMBGA with probabilistic building-block crossover will converge in $O(n)$ evaluations for a fixed probability of getting each trap partition right and $O(n \ln n)$ evaluations for a constant number of wrong partitions. This reasoning leads to an interesting conclusion: If the order of subproblems in a proper problem decomposition is bounded and a probabilistic model encodes a correct problem decomposition, the complexity of PMBGAs is $O(n)$ or $O(n \ln n)$ independently of the order of subproblems. That poses challenge of how we learn the linkage. If we can learn the linkage without substantial increase in cost, PMBGAs can solve problems of bounded difficulty quickly, accurately, and reliably.

But let's consider other alternative approaches to solving problems of bounded difficulty. GAs with fixed crossover—such as one-point and uniform crossover—will require exponentially sized populations to find the optimum of composed traps of any order $k \geq 3$ (Thierens & Goldberg, 1993). Therefore, the complexity of GAs with fixed recombination is at least exponential. Consequently, fixed recombination operators are not capable of solving arbitrary problems of bounded difficulty efficiently.

Another alternative is to use a stochastic hill climber based on bit-flip mutation. The stochastic hill climber starts with a random solution. In each iteration, the hill climber applies bit-flip mutation to the current solution, and replaces the original solution by the new one if the new solution performs better. The number of evaluations until the stochastic hill climber converges to the optimum for problems decomposable into subproblems of order k can be bounded by $O(n^k \ln n)$ as shown by Mühlenbein (1992). Although n^k is polynomial for a fixed k, for moderate values of k, n^k grows extremely fast and the search becomes inefficient compared to PMBGAs. The performance of GAs with selection and mutation only follows a similar trend, because it can be approximated by a number of parallel hill climbers.

Therefore, both mutation and fixed crossover don't scale up well with the problem size for boundedly difficult problems of moderate difficulty. Nonetheless, if we were capable of designing operators that would *learn* appropriate decomposition without introducing significant computation overhead, problems of bounded difficulty (no matter what the difficulty is) can be solved in a subquadratic number of fitness evaluations.

Much research in PMBGAs focuses on learning appropriate decomposition using populations of promising solutions as a source of information about the structure of the problem. To learn a proper decomposition (identify the building blocks), PMBGAs use techniques for learning probabilistic models. However, learning probabilistic models is difficult and there are several important tradeoffs that must be considered for the design of competent PMBGAs.

The first tradeoff is related to the *number of independence assumptions* a model makes. With too few independence assumptions, the model does not allow for effective exploration and requires large populations. Imagine we would assume that each variable depends on every other variable and therefore we would not recombine promising solutions at all. That would lead to no exploration and, consequently, we would fail unless we guessed the solution initially. Moreover, to estimate the parameters of such a complex model, the population size would have to grow exponentially with the problem size. On the other hand, with too many independence assumptions, a model may not prevent important partial solutions from disruption. Recall that if the model would not consider important dependencies in composed traps, PMBGAs would require exponentially large populations and the time complexity would therefore be at least exponential with respect to the problem size.

The second tradeoff is related to the *complexity of learning algorithms*. Simple models are easy to learn, but PMBGAs with simple models often fail at solving complex problems. Complex models allow PMBGAs to solve complex problems, but it costs more to learn such models.

The remainder of this chapter reviews PMBGAs proposed in the past. The methods are classified according to the underlying representation of candidate solutions and the complexity of the class of models they consider. The first order of business is to describe PMBGAs that assume that candidate solutions are represented by fixed-length strings over a finite alphabet; we call these algorithms *discrete PMBGAs*. Next, PMBGAs that use models defined over other domains are summarized.

2.3 Discrete Variables

One natural classification of discrete PMBGAs considers the order of interactions that the underlying models employ. PMBGAs have evolved from modeling low-order interactions to modeling higher order ones over time. This section starts by reviewing PMBGAs that assume that all variables in a problem are independent; these approaches are based on probabilistic uniform crossover discussed earlier. Next, the section discusses those PMBGAs capable of covering some pairwise interactions by using probabilistic models in the form of a chain, tree, and forest. The section ends with a short overview of PMBGAs that can cope with interactions of any order.

2.3.1 No Interactions

The population-based incremental learning (PBIL) (Baluja, 1994) replaces the population of solutions by a *probability vector* $(p_0, p_1, \ldots, p_{n-1})$, where p_i denotes the probability of 1 at the ith position of solution strings. Each p_i is initially set to 0.5, which corresponds to a uniform distribution over the set of all solutions. In each iteration, PBIL generates a specified number of solutions according to the current probability vector. Each value is generated independently of its context (remaining bits) and thus no interactions are considered (see Figure 2.2). The best solution out of the generated set of solutions is then used to update the probability-vector entries using

$$p_i = (1 - \lambda)p_i + \lambda x_i,$$

where $\lambda \in (0, 1)$ is the learning rate (say, 0.02), and x_i is the ith bit of the best solution. Using the above rule, the probability p_i of each bit being 1 increases if the best solution contains 1 in that position and decreases otherwise. In other words, probability-vector entries move *toward* the best solution. The process of generating new solutions and updating the probability vector is repeated until some termination criteria are met; for instance, the run can be terminated if all the probabilities are close to either 0 or 1.

For example, for a 5-bit onemax, 3 strings generated in each iteration, and $\lambda = 0.2$, the first iteration of PBIL could proceed as follows. Assume that the solutions generated from the initial probability vector $(0.5, 0.5, 0.5, 0.5, 0.5)$ are 10010, 11010, and 10111. The last solution has the

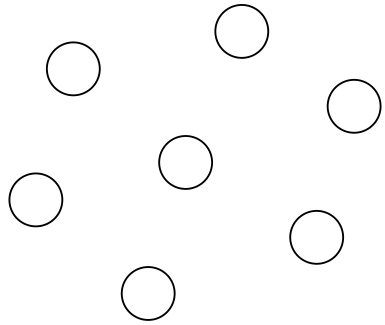


Figure 2.2: A graphical model with no interactions covered displayed as a Bayesian network. The model with no interactions is equivalent to probabilistic uniform crossover.

highest fitness and is used to update the probability vector. With $\lambda = 0.2$, the new probability vector is $(0.6, 0.4, 0.6, 0.6, 0.6)$.

In the GA literature, PBIL has been also referred to as hill climbing with learning (HCwL) (Kvasnicka, Pelikan, & Pospichal, 1996) and the incremental univariate marginal distribution algorithm (IUMDA) (Mühlenbein, 1997). Theoretical analyses of PBIL can be found in Kvasnicka, Pelikan, and Pospichal (1996), Höhfeld and Rudolph (1997), and Gonzalez, Lozano, and Larranaga (2001).

The basic difference between PBIL and most GAs (including PMBGAs) is that PBIL does not actually use a population, but it replaces the population by a probability vector. Consequently, the connection between PBIL and GAs becomes somewhat fuzzy. The compact genetic algorithm (cGA) (Harik, Lobo, & Goldberg, 1997) eliminates the gap between PBIL and traditional GAs. As PBIL, cGA replaces the population by a probability vector. All entries in the probability vector are initialized to 0.5. However, a different update rule is used to simulate one round of a variant of binary tournament selection that replaces the worst of the two solutions by the best one using a population of size N. In particular, denoting the bit in the ith position of the best and worst of the two solutions by x_i and y_i, respectively, the probability-vector entries are updated as follows:

$$p_i = \begin{cases} p_i + \frac{1}{N} & \text{if } x_i = 1 \text{ and } y_i = 0 \\ p_i - \frac{1}{N} & \text{if } x_i = 0 \text{ and } y_i = 1 \\ p_i & \text{otherwise} \end{cases},$$

where N denotes the "population size". Although cGA uses a probability vector in place of a population, updates of the probability vector correspond to replacing one candidate solution by another one using a population of size N and shuffling the resulting population using population-wise uniform crossover.

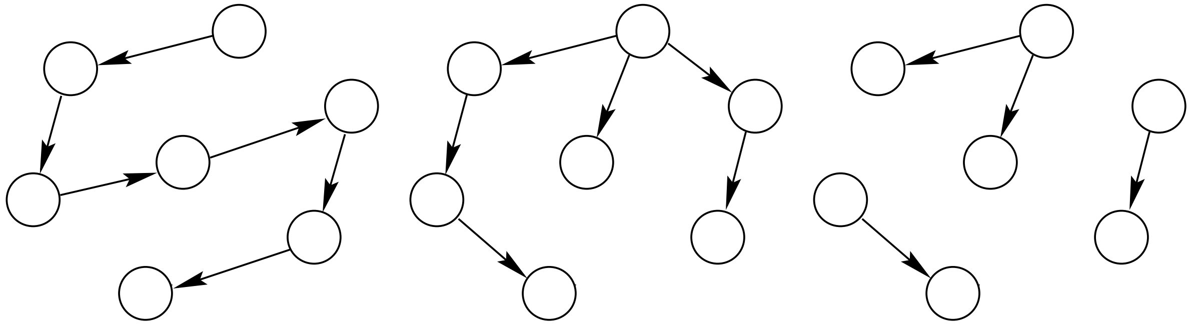
Unlike PBIL and cGA, the univariate marginal distribution algorithm (UMDA) (Mühlenbein & Paaß, 1996) maintains a population of solutions. Each iteration of UMDA starts by selecting a population of promising solutions as in standard GAs. A probability vector is then computed using the selected population of promising solutions and new solutions are generated by sampling the probability vector. The new solutions replace the old ones and the process is repeated until termination criteria are met. UMDA is therefore equivalent to a GA with probabilistic uniform crossover (see sections 1.4 and 2.2.1). Since UMDA uses a probabilistic model as an intermediate step between the original and new populations, UMDA fits the PMBGA framework better than PBIL and cGA do. Nonetheless, the performance and dynamics of PBIL, cGA, and UMDA are similar.

All the algorithms described in this section can solve problems decomposable into subproblems of order one in a linear or quadratic number of fitness evaluations, depending on the type of the problem at hand. However, if decomposition into single-bit subproblems misleads the decision making away from the optimum (some building blocks cannot be smaller than 2 bits long), these algorithms scale up poorly with increased problem size. The next section considers PMBGAs that use models with pairwise dependencies; consequently, these PMBGAs extend the class of problems that can be solved in a scalable manner to problems decomposable into subproblems of order at most two.

2.3.2 Bivariate Interactions

This section focuses on PMBGAs using pairwise probabilistic models that can encode dependencies in the form of a chain, a tree, or a forest (a set of isolated trees) among the variables. Because these models move beyond the assumption of variable independence, they represent a first step toward competent PMBGAs capable of linkage learning.

The mutual-information-maximizing input clustering (MIMIC) algorithm (De Bonet, Isbell, & Viola, 1997) uses a chain distribution (see Figure 2.3(a)) specified by (1) an ordering of string



(a) MIMIC (b) Baluja & Davies (c) BMDA

Figure 2.3: The graphical models with pairwise interactions covered displayed as a Bayesian network. The models that can cover some pairwise interactions form the (a) chain, (b) tree, or (c) forest.

positions (variables), (2) a probability of 1 in the first position of the chain, and (3) conditional probabilities of each other position given the value in the previous position of the chain. A chain probabilistic model encodes the probability distribution where the first variable is independent of the remaining variables (as in probabilistic uniform crossover) and each other variable is conditioned on the previous variable in the chain.

After selecting promising solutions and computing marginal and conditional probabilities, a greedy algorithm is used to maximize the mutual information of adjacent variables in the chain. In this fashion the Kullback-Liebler divergence (Kullback & Leibler, 1951) between the chain and actual distributions is minimized; however, the greedy algorithm does not guarantee global optimality of the constructed model (with respect to Kullback-Liebler divergence). The greedy algorithm starts in a variable with the lowest unconditional entropy. The chain is expanded by adding a new variable that minimizes the conditional entropy of the new variable given the last variable in the chain. Once the full chain is constructed for the selected population of promising solutions, new solutions are generated by sampling the distribution encoded by the chain and the probabilities associated with the chain.

There are two important drawbacks of using chain distributions. The first drawback is that chain distributions allow only a very limited representation of dependencies in a problem. Despite that, chain distributions can encode dependencies between pairs of positions that are distant in solution strings; these dependencies are preserved by neither uniform nor one-point crossover. The second drawback is that there is no known algorithm for learning the best chain distribution in polynomial time. Despite the disadvantages of using only chain distributions, the use of pairwise interactions was one of the most important steps in the development of competent PMBGAs. It was the first time probability distributions and their estimation were used to learn linkage.

Baluja and Davies (1997) use dependency trees (see Figure 2.3(b)) to model promising solutions. As in PBIL, the population is replaced by a probability vector containing all *pairwise* probabilities. The probabilities are initialized to 0.25 and repeatedly adjusted according to new promising solutions acquired on the fly. A dependency tree encodes the probability distribution where the variable in the root of the tree is independent, and every other variable is conditioned on its parent in the tree.

A variant of Prim's algorithm for finding the minimum spanning tree of a given graph can be used (Prim, 1957) to construct an optimal tree distribution. Here the task is to find a tree that maximizes mutual information between the parents and their children. This can be done by first randomly choosing a variable to form the root of the tree, and "hanging" new variables to the existing tree so that the mutual information between the parent of the new variable and the variable itself is maximized. In this way, the Kullback-Liebler divergence between the tree and actual distributions is minimized as shown by Chow and Liu (1968). Once a full tree is constructed, new solutions are generated according to the distribution encoded by the constructed dependency tree and the conditional probabilities computed from the probability vector.

The bivariate marginal distribution algorithm (BMDA) (Pelikan & Mühlenbein, 1999) uses a forest distribution (a set of mutually independent dependency trees, see Figure 2.3(c)). This class of models is even more general than the class of dependency trees, because any forest that contains two or more disjoint trees cannot be represented by a tree. As a measure used to determine whether to connect two variables, BMDA uses a Pearson's chi-square test (Marascuilo & McSweeney, 1977). This measure is also used to discriminate the remaining dependencies in order to construct the final model. To learn a model, BMDA uses a variant of Prim's algorithm (Prim, 1957).

Pairwise models capture some interactions in a problem with reasonable computational overhead. The algorithms presented in this section can identify, propagate, and juxtapose building blocks of order two, and therefore they work well on problems decomposable into subproblems of order at most two (De Bonet, Isbell, & Viola, 1997; Baluja & Davies, 1997; Mühlenbein, 1997; Pelikan & Mühlenbein, 1999; Bosman & Thierens, 1999). Nonetheless, capturing only some pairwise interactions has still shown to be insufficient for solving problems with multivariate or highly overlapping building blocks (Pelikan & Mühlenbein, 1999; Bosman & Thierens, 1999). That is why PMBGA research has pursued more complex models discussed in the next section.

2.3.3 Multivariate Interactions

This section overviews PMBGAs using models that can encode multivariate interactions. Using general multivariate models has brought powerful algorithms capable of solving problems of bounded difficulty quickly, accurately, and reliably. On the other hand, learning distributions with multivariate interactions necessitates more complex model-learning algorithms that require significant computational time and still do not guarantee global optimality of the resulting model. Nonetheless, many difficult problems are intractable using simple models and the use of complex models and algorithms is warranted.

The factorized distribution algorithm (FDA) (Mühlenbein, Mahnig, & Rodriguez, 1999) uses a fixed factorized distribution throughout the whole computation. The model is allowed to contain multivariate marginal and conditional probabilities, but FDA learns only the probabilities, not the structure (dependencies and independencies). To solve a problem using FDA, we must first decompose the problem and then factorize the decomposition. This brings us back to population-wise building-block crossover that is given building-block partitions in advance (see Section 2.2.2). While it is useful to incorporate prior information about the regularities in the search space, the basic idea of black-box optimization is to *learn* the regularities in the search space as opposed to using the regularities specified by an expert. In other words, FDA ignores the problem of learning *what statistics* are important to process within the PMBGA framework, and must be given that information in advance.

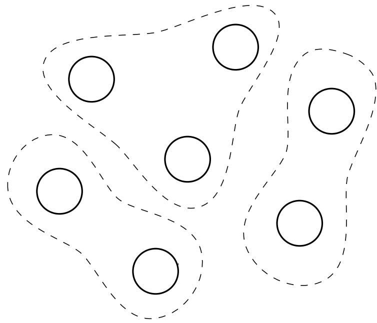
The extended compact genetic algorithm (ECGA) (Harik, 1999) uses a marginal product model (MPM) that partitions the variables into several partitions that are processed as independent variables in UMDA (see Figure 2.4(a)). Each partition is treated as a single variable and different partitions are considered to be mutually independent. The effects of learning and sampling an

MPM are thus the same as those of probabilistic building-block crossover from Section 2.2.2 with the building-block partitions specified by the MPM.

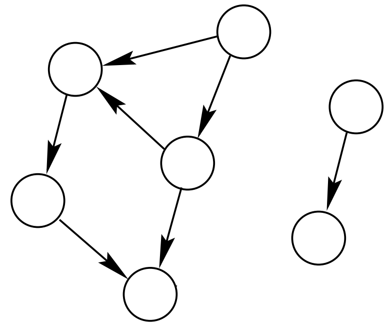
To decide between alternative MPMs, ECGA uses one of minimum description length (MDL) metrics (Rissanen, 1978; Rissanen, 1989; Rissanen, 1996), which favor models that allow higher compression of data (in this case, the selected set of promising solutions). In particular, the Bayesian information criterion (BIC) (Schwarz, 1978) is used. The advantage of using MDL metrics is that they penalize complex models when they are not supported by significant statistical evidence and, therefore, the resulting models are usually not overly complex. To find a good model, ECGA uses a simple greedy algorithm that starts with each variable forming one partition. Each iteration of the greedy algorithm merges two partitions that maximize the improvement of the model with respect to BIC. If no more improvement is possible, the current model is used.

The initial model in ECGA is equivalent to the probability vector of PBIL, cGA, and UMDA. However, bigger building blocks can be learned by merging multiple groups into a single one. If the constructed model reflects a proper decomposition of the problem, ECGA is a very powerful algorithm (Sastry & Goldberg, 2000; Sastry, 2001a). However, many real-world problems contain overlapping dependencies (e.g., two-dimensional spin-glass systems examined in chapter 7), which cannot be accurately modeled by dividing the variables into disjoint partitions. This can result in poor performance of ECGA on those problems.

The Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 1998) builds a Bayesian network for the population of promising solution (see Figure 2.4(b)). The learned network is sampled to generate new solutions. Initially, BOA used the Bayesian-Dirichlet metric subject to a maximum model-complexity constraint (Cooper & Herskovits, 1992; Heckerman, Geiger, & Chickering, 1994) to discriminate competing models, but other metrics have been analyzed in later work. In all variants of BOA, the model is constructed by a greedy algorithm that iteratively adds a new dependency in the model that maximizes the model quality. Other elementary graph operators—such as edge removals and reversals—can be incorporated, but edge additions are most important. If no more improvement is possible, the current network is used. The greedy algorithm used to learn a model in BOA is similar to the one used in ECGA. However, Bayesian networks can encode more complex dependencies and independencies than models used in ECGA can. Therefore,



(a) ECGA



(b) BOA

Figure 2.4: The graphical models with multivariate interactions covered. In ECGA, the variables are partitioned into several groups, each corresponding to one component of the problem decompositions; the partitions are considered independent. In BOA, the variables are related using directed edges, forming a directed acyclic graph called Bayesian network.

BOA is also applicable to problems with overlapping dependencies. Several such problems will be discussed in chapter 7.

BOA uses an equivalent class of models as FDA does; however, BOA learns both the structure and the probabilities of the model. Although BOA does not require problem-specific knowledge in advance, prior information about the problem can be incorporated using Bayesian statistics, and the relative influence of prior information to that contained in the population of promising solutions can be tuned by the user. An interesting study on incorporating prior problem-specific information to improve the performance of BOA in graph partitioning can be found in Schwarz and Ocenasek (2000).

A discussion of the use of Bayesian networks as an extension to tree models can also be found in Baluja and Davies (1998). An algorithm that uses Bayesian networks to model promising solutions was independently developed by Etxeberria and Larrañaga (1999), who called it the estimation of Bayesian network algorithm (EBNA). Mühlenbein and Mahnig (1999) later improved the original FDA by using Bayesian networks together with the greedy algorithm for learning the networks described above. The modification of FDA was named the learning factorized distribution algorithm (LFDA).

PMBGAs that use models capable of covering multivariate interactions can solve a wide range of problems in a scalable manner; promising results were reported on two-dimensional Ising spin-glass systems (Pelikan, Goldberg, & Cantú-Paz, 1998; Mühlenbein & Mahnig, 1998), graph partitioning (Schwarz & Ocenasek, 1999; Schwarz & Ocenasek, 2000), telecommunication network optimization (Rothlauf, Goldberg, & Heinzl, 2000), and silicon cluster optimization (Sastry, 2001a). Later (see chapters 5 and 6) we will see how hierarchical decomposition can be used to further extend the applicability of BOA beyond problems decomposable into tractable subproblems on a single level.

2.4 Other Representations

There are two basic approaches to extending PMBGAs for discrete fixed-length strings to other domains such as variable-length strings, vectors of real numbers, symbolic expressions, and program codes:

1. Map the problem to the domain of fixed-length discrete strings, solve the discrete problem, and map the solution back to the problem's original domain.

2. Extend or modify the class of probabilistic models to other domains.

The first approach requires the design of different techniques for mapping one domain to another; such techniques have been studied in the context of genetic and evolutionary algorithms for several decades. This section reviews PMBGAs based on the second approach. The section starts with an overview of PMBGAs for optimizing problems over the fixed-length vectors of real-valued variables. Subsequently, the section discusses PMBGAs for optimizing computer programs and symbolic expressions.

2.4.1 Real-valued Variables

There are many approaches to modeling and sampling real-valued distributions. One way of classifying these approaches considers the number of variables that are treated together; some approaches build a separate model for each variable (like PBIL), whereas other approaches model groups of variables or all variables together (like ECGA or BOA). Another way of classifying real-valued PMBGAs focuses on the type of distributions that are used to model each variable or each group of variables; real-valued models include normal distributions, joint normal distributions, histogram

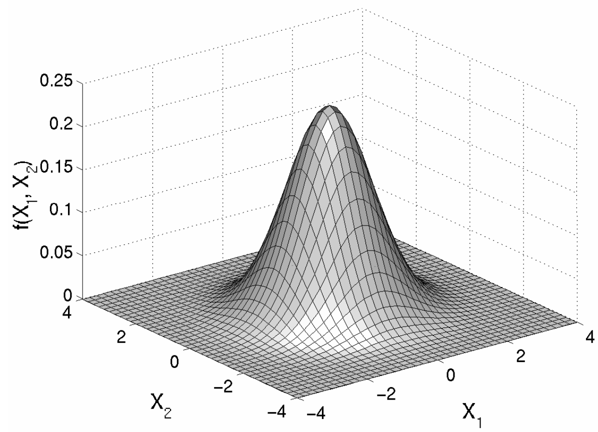


Figure 2.5: The product of one-dimensional Gaussian distributions as used in the stochastic hill-climbing with learning by vectors of normal distributions (SHCLVND) of Rudlof et al. (1996). Each variable is modeled by a normal distribution, and the overall distribution is given by the product of the individual distributions for each variable.

distributions, uniform distributions over intervals, mixture distributions, and others. This section starts by presenting PMBGAs based on normal distributions. Next, the section discusses real-valued PMBGAs based on histogram and interval distributions.

Single-Peak Normal Distributions

The stochastic hill climbing with learning by vectors of normal distributions (SHCLVND) (Rudlof & Köppen, 1996) is a straightforward extension of PBIL to vectors of real-valued variables using a normal distribution to model each variable. SHCLVND replaces the population of real-valued solutions by a vector of means $\mu = (\mu_1, \ldots, \mu_n)$, where μ_i denotes a mean of the distribution for the ith variable. The same standard deviation σ is used for all variables. See Figure 2.5 for an example model. Each generation, a random set of solutions is first generated according to μ and σ. The best solution out of this subset is then used to update the entries in μ by shifting each μ_i toward the value of ith variable in the best solution using a similar update rule as in PBIL. Additionally, each generation reduces the standard deviation to make the future exploration of the search space narrower. A similar algorithm was independently developed by Sebag and Ducoulombier (1998), who also discussed several approaches to evolving a standard deviation for each variable.

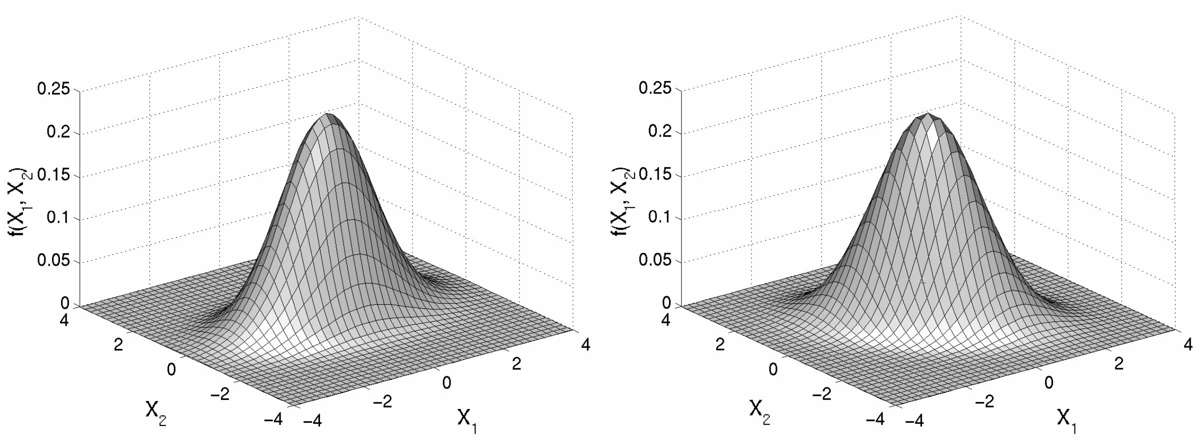
Mixtures of Normal Distributions

The probability density function of a normal distribution is centered around its mean and decreases exponentially with the distance from the mean. If there are multiple "clouds" of values, a normal distribution must either focus on only one of these clouds, or it can embrace multiple clouds at the expense of including the area between these clouds. In both cases, the resulting distribution cannot model the data accurately. One way of extending normal distributions to enable coverage of multiple groups of similar points is to use a mixture of normal distributions. Each component of a mixture of normal distributions is a normal distribution by itself. A coefficient is specified for each component of the mixture to denote the probability that a random point belongs to this component. The probability density function of a mixture is thus computed by multiplying the density function of each mixture component by the probability that a random point belongs to the component, and adding these weighted densities together.

Gallagher, Frean, and Downs (1999) extended PMBGAs using single-peak normal distributions by using an adaptive mixture of normal distributions to model each variable. The parameters of a mixture (including the number of components) evolve based on the discovered promising solutions. Using mixture distributions is a significant improvement compared to single-peak normal distributions, because mixtures allow simultaneous exploration of multiple basins of attraction for each variable.

Joint Normal Distributions and Their Mixtures

What changes when instead of fitting each variable with a separate normal distribution or a mixture of normal distributions, groups of variables are considered together? Let us first consider using a single-peak normal distribution. In multivariate domains, a joint normal distribution can be defined by a vector of n means (one mean per variable) and a *covariance matrix* of size $n \times n$. Diagonal elements of a covariance matrix specify the variances for all variables, whereas nondiagonal elements specify linear dependencies between pairs of variables. Considering each variable separately corresponds to setting all nondiagonal elements in a covariance matrix to 0. Using different deviations for different variables allows for "squeezing" or "stretching" the distribution along the axes. On the other hand, using nondiagonal entries in the covariance matrix allows rotating the distribution



(a) Diagonal covariance matrix.

(b) Full covariance matrix.

Figure 2.6: A joint normal distribution. The left-hand side shows a joint normal distribution with only diagonal entries in the covariance matrix; all remaining entries in the covariance matrix are set to 0. The right-hand side shows a joint normal distribution that includes also the covariance between the two variables, which allows rotating the distribution around its mean.

around its mean. Figure 2.6 illustrates the difference between a joint normal distribution using only diagonal elements of the covariance matrix and a distribution using the entire covariance matrix. Therefore, using a covariance matrix introduces another degree of freedom and improves the expressiveness of a distribution. Again, one can use a number of joint normal distributions in a mixture, each component with its own covariance matrix and weight.

A joint normal distribution including a full or partial covariance matrix was used within the IDEA framework (Bosman & Thierens, 2000a) and in the estimation of Gaussian networks algorithm (EGNA) (Larranaga, Etxeberria, Lozano, & Pena, 2000b). Both these algorithms can be seen as an extension of PMBGAs that model each variable by a single normal distribution to use also nondiagonal elements in the covariance matrix.

Bosman and Thierens (2000b) proposed *mixed IDEAs* as an extension of PMBGAs that use a mixture of normal distributions to model each variable. Mixed IDEAs allow multiple variables to be modeled by a separate mixture of joint normal distributions. At one extreme, each variable can have a separate mixture; at another extreme, one mixture of joint distributions covering all the variables is used. Despite that learning such a general class of distributions is quite difficult and a large number of samples is necessary for reasonable accuracy, good results were reported on single-

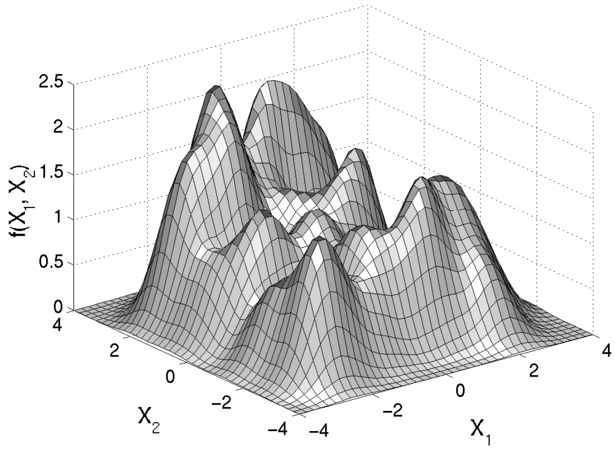


Figure 2.7: A joint normal kernels distribution is a mixture of joint normal distributions, where a special mixture component is reserved for each point in the modeled data.

objective (Bosman & Thierens, 2000b) as well as multiobjective problems (Thierens & Bosman, 2001). Using mixture models for all variables was also proposed as a technique for reducing model complexity in discrete PMBGAs (Pelikan & Goldberg, 2000a).

Within the IDEA framework, Bosman and Thierens (2000a) proposed IDEAs using joint normal *kernels* distributions, where a single normal distribution is placed around each promising solution (see Figure 2.7). A joint normal kernels distribution can be therefore seen as an extreme use of mixture distributions with one mixture component per point in the training sample. The variance of each normal distribution can be either fixed to a relatively small value, but it should be preferable to adapt variances to the current state of search. Using kernel distributions corresponds to using a fixed zero-mean normally distributed mutation for each promising solution as is often done in evolution strategies (Rechenberg, 1973). That is why it is possible to directly take up strategies for adapting the variance of each kernel from evolution strategies (Rechenberg, 1973; Rechenberg, 1994; Schwefel, 1977; Hansen, Ostermeier, & Gawelczyk, 1995).

Real-valued PMBGAs presented so far are applicable real-valued optimization problems without requiring differentiability or continuity of the underlying problem. However, if it is possible to at least partially differentiate the problem at hand, gradient information can be used to incorporate some form of gradient-based local search and the performance of real-valued PMBGAs can be significantly improved. A study on combining real-valued PMBGAs within the IDEA framework with gradient-based local search can be found in Bosman and Thierens (2001).

To summarize, the following questions are important in the design of real-valued PMBGAs using normal distributions or mixtures of normal distributions:

- **Decomposition?** Use a separate distribution estimate for each variable, or consider groups of variables together?
- **Covariances?** If considering multiple variables together, use a diagonal, partial, or full covariance matrix?
- **Mixtures?** Use one-peak normal distribution, or a mixture of normal distributions?
- **Kernels?** Use a mixture of normal distributions with one component per solution (joint normal kernels distribution), or attempt to create a global model that generalizes the data globally?

Different models might be advantageous for different types of problems. For successful application of real-valued PMBGAs based on normal distributions it is important to consider the above questions and make decisions based on the properties of the problem at hand. For example, if the problem at hand is highly multimodal, using a single-peak distribution will most likely not work; however, if there are extremely many local optima, using single-peak distributions could generalize the problem landscape and avoid getting stuck in a local optimum. Furthermore, it is important to consider the choice of a distribution to set a population size and other parameters. For example, while for joint normal kernels distributions populations of only few points might suffice, for adapting complex multivariate mixture distributions a rather large sample is required.

Other Real-valued PMBGAs

Of course, using normal distributions is not the only one way of modeling real-valued distributions. Other density functions are frequently used in practice, including histogram distributions, interval distributions, and others. The discussion of PMBGAs that use other than normal distributions follows.

In the algorithm proposed by Servet, Trave-Massuyes, and Stern (1997), an interval (a_i, b_i) and a number $z_i \in (0, 1)$ are stored for each variable (see Figure 2.8). By z_i, the probability that the ith variable is in the lower half of (a_i, b_i) is denoted. Each z_i is initialized to 0.5. To generate a new

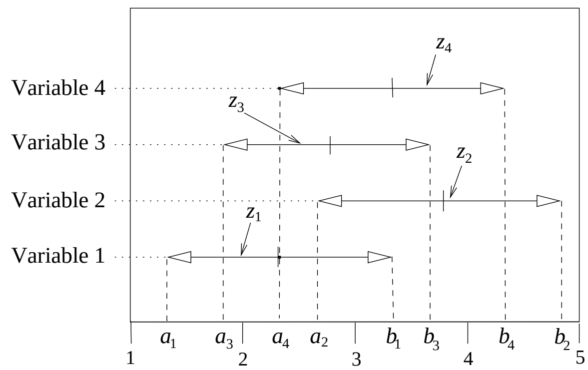
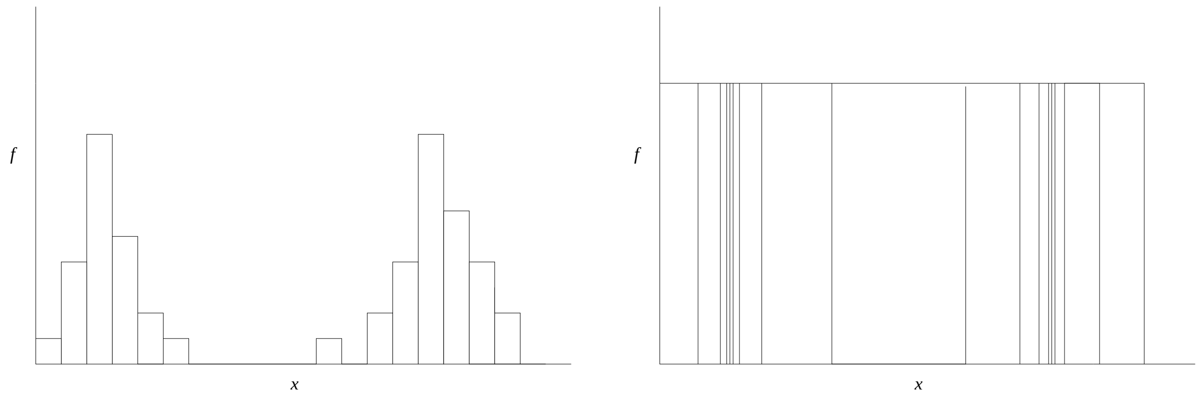


Figure 2.8: A model based on adaptive intervals (Servet et al., 1998).

candidate solution, the value of each variable is selected randomly from the corresponding interval. The best solution is then used to update the value of each z_i. If the value of the ith variable of the best solution is in a lower half of (a_i, b_i), z_i is shifted toward 0; otherwise, z_i is shifted toward 1. When z_i gets close to 0, interval (a_i, b_i) is reduced to its lower half; if z_i gets close to 1, interval (a_i, b_i) is reduced to its upper half. Figure 2.8 shows an example probabilistic model based on adaptive intervals; in the figure, each z_i is mapped to the corresponding interval (a_i, b_i).

Bosman and Thierens (2000a), Tsutsui, Pelikan, and Goldberg (2001) and Cantú-Paz (2001) use empirical histograms to model each variable as opposed to using a single normal distribution or a mixture of normal distributions. In these approaches, a histogram for each single variable is constructed. New points are then generated according to the distribution encoded by the histograms for all variables. Figure 2.9 shows examples of fixed-height and fixed-width histograms. The sampling of a histogram proceeds by first selecting a particular bin based on its relative frequency, and then generating a random point from the interval corresponding to the bin. It is straightforward to replace the histograms in the above methods by various classification and discretization methods of statistics and machine learning (such as k-means clustering).

Pelikan, Goldberg, and Tsutsui (2001) use an adaptive mapping from the continuous domain to the discrete one in combination with linkage learning in the discrete domain. The population of promising solutions is first discretized by using equal-width histograms, equal-height histograms, k-means clustering, or other classification techniques. A population of promising discrete solutions is then selected. New points are created by applying a discrete recombination operator capable of linkage learning (such as BOA) to the selected population of promising discrete solutions. The



(a) Fixed-width histogram.

(b) Fixed-height histogram.

Figure 2.9: Fixed-width and fixed-height histograms. In a fixed-width histogram, each bin has equal width, and the height of each bin determines the number of points in that bin. In a fixed-height histogram, the width of each bin is set so that each bin contains the same number of points.

resulting discrete solutions are then mapped back into the continuous domain by sampling each class (a bin or a cluster) using the original values of the variables in the selected population of continuous solutions (before discretization). The resulting solutions are mutated using one of the adaptive mutation operators of evolution strategies (Rechenberg, 1973; Rechenberg, 1994; Schwefel, 1977; Hansen, Ostermeier, & Gawelczyk, 1995). In this way, competent discrete PMBGAs can be combined with advanced methods based on adaptive local search in the continuous domain.

2.4.2 Computer Programs

Genetic programming (Koza, 1992; Koza, 1994) evolves a population of computer programs using variants of crossover and mutation appropriate to a program code. Programs are represented by labeled trees. Internal nodes of a program tree correspond to functions with at least one argument, the successors (children) of an internal node represent arguments of the function encoded in the node, and leaves of a program tree corresponds to terminal symbols (functions with no arguments, variables, or constants).

Since the context of different parts of a program usually matters a great deal, it should be advantageous to use competent PMBGAs capable of linkage learning in the domain of genetic programming. However, using PMBGAs for genetic programming poses two challenges: (1) computer programs are not linear structures, and (2) the length of computer programs can vary. The remain-

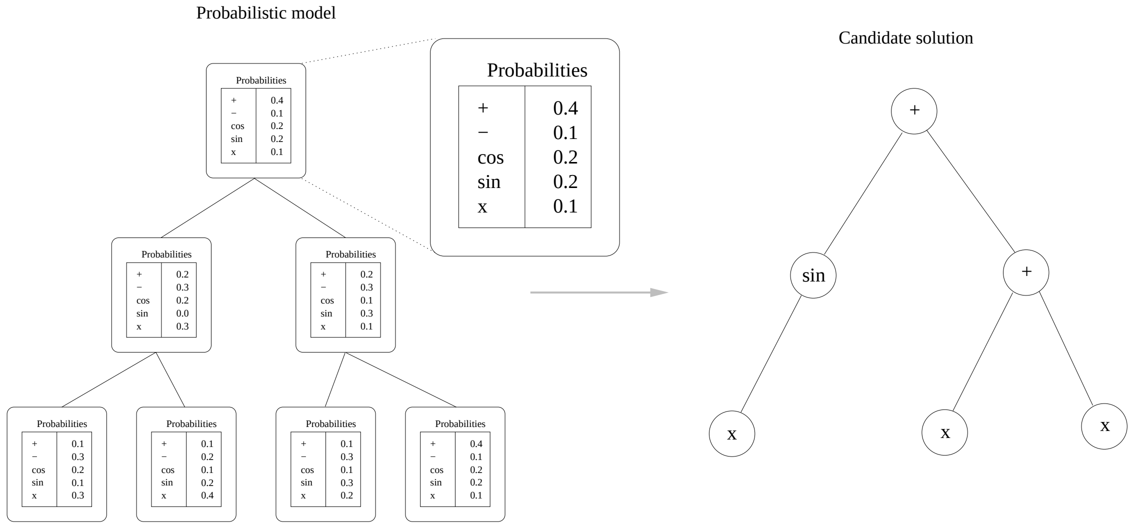


Figure 2.10: This figure shows an example probabilistic model of a program with no interactions covered used in PIPE and a candidate solution (mathematic expression) generated by the model. Each node in the model stores a probability of each instruction and terminal symbol. The simulation proceeds by deciding whether to terminate in each node or not. If the node is not terminated, the probabilities of different operators and terminals are used to generate a value or a function in the node.

der of this section presents several approaches to applying PMBGAs to genetic programming.

Probabilistic incremental program evolution (PIPE) (Salustowicz & Schmidhuber, 1997a; Salustowicz & Schmidhuber, 1997b) uses a probabilistic model in the form of a tree of a specified maximum allowable size. Nodes in a model specify the probabilities for different functions and terminals. PIPE does not employ any interactions among the nodes in a model. To visualize the model used in PIPE, see Figure 2.10. The model is updated by adapting the probabilities based on the new promising solutions. To sample new program trees, the generation starts in the root and continues down as necessary. In particular, if the model generates a function in some node and that function requires additional arguments, the successors (children) of the node are generated to form the arguments of the function. If a terminal is generated, the generation along the considered branch terminates.

A hierarchical extension of PIPE—the so-called H-PIPE—has been later proposed by Salustowicz and Schmidhuber (1998). In H-PIPE, nodes of a model are allowed to contain subroutines. Both the subroutines and the overall program are evolved.

Handley (1994) uses directed acyclic graphs to represent the population of programs (trees) in genetic programming. Although the goal of this work was to compress the population of computer programs in genetic programming, Handley's approach can be used within the PMBGA framework to model and sample candidate solutions represented by computer programs or symbolic expressions.

Using advanced probabilistic models in genetic programming is a challenging area, because optimization of program codes and symbolic expressions is difficult and the need for effective exploration necessitates learning semantics of the used representation and its effects on performance of candidate programs. Furthermore, the task of building and sampling probabilistic models becomes much more difficult when using the complex representation of genetic programming.

2.5 Summary

This chapter motivated the use of probability distributions to combine bits and pieces of promising solutions. Additionally, the chapter provided a survey of probabilistic model-building genetic algorithms, which use probabilistic models to bias the sampling in optimization. A summary of the key points of this chapter follows:

- Probabilistic model-building genetic algorithms (PMBGAs) replace crossover and mutation of traditional GAs by (1) building a probabilistic model for the promising solutions and (2) sampling the built model to generate new solutions. PMBGAs are sometimes called estimation of distribution algorithms (EDAs) and iterated density estimation algorithms (IDEAs).

- Probabilistic models are a general tool for encoding a problem decomposition. The effects of a model that assumes no dependencies at all are similar to those of uniform crossover. On the other hand, if a probabilistic model encodes all possible dependencies, no recombination takes place. It is necessary that PMBGAs be capable of learning a model that encodes a proper, non-misleading, decomposition of the problem.

- Learning a probabilistic model corresponds to learning a proper problem decomposition. Sampling the constructed model corresponds to combining promising solutions according to a decomposition encoded by the model.

- PMBGAs for discrete domains can be classified according to the complexity of the models they use:

    1. **No interactions.** PMBGAs that use models with no interactions can solve linear problems very fast, but they achieve poor performance on more complex problems with strong interactions of high order.

    2. **Some pairwise interactions.** PMBGAs that use models in the form of a chain, tree, or forest can process some partial solutions of order two. That extends the domain of problems that can be solved by PMBGAs in a scalable manner, but is still insufficient to ensure that PMBGAs can solve arbitrary problems of bounded difficulty.

    3. **Multivariate interactions.** PMBGAs that use models with multivariate interactions can solve any problem decomposable into subproblems of bounded order. However, using complex models also introduces additional computational overhead.

- There are two basic approaches to extending discrete PMBGAs to other domains: (1) map solutions from the other domain to discrete fixed-length strings over a finite-sized alphabet, and (2) extend or modify models to allow for the modeling and sampling of solutions in the other domain.

- Most PMBGAs for optimizing real-valued problems are based on normal distributions and mixtures of normal distributions. Other real-valued PMBGAs use adaptive intervals, histograms, clustering, and other classification techniques. Classification techniques can also be used to combine optimization methods for real-valued and discrete optimization to combine the best of both worlds.

- Designing PMBGAs for optimization of computer programs of genetic programming is an important challenge. Advanced probabilistic models of computer programs can be expected to improve the performance and scalability of genetic programming.

Chapter 3

Bayesian Optimization Algorithm

The previous chapter showed that using probabilistic models with multivariate interactions is a powerful approach to solving problems of bounded difficulty. For efficient and scalable optimization, a model must be learned that encodes a proper decomposition of the problem. The Bayesian optimization algorithm (BOA) combines the idea of using probabilistic models to guide optimization and the methods for learning Bayesian networks. To learn a proper decomposition of the problem, BOA builds a Bayesian network for the set of promising solutions. New candidate solutions are generated by sampling the built network.

The purpose of this chapter is threefold. First, the chapter describes the Bayesian optimization algorithm (BOA), which uses Bayesian networks to model promising solutions and bias the sampling of new candidate solutions. Second, the chapter describes how to learn and sample Bayesian networks. Third, the chapter tests the proposed algorithm on a number of challenging decomposable problems.

The chapter starts by describing the basic procedure of BOA. Section 3.2 provides background of Bayesian networks and their semantics in BOA. Section 3.3 discusses the problem of learning the structure and parameters of Bayesian networks given a sample from an unknown target distribution. The section provides two approaches to measuring the quality of each candidate model: (1) Bayesian metrics and (2) minimum description length metrics. Additionally, the section describes a greedy algorithm for learning the structure of Bayesian networks by repeated application of primitive graph operations. Section 3.4 describes a method for sampling the learned Bayesian network to generate new candidate solutions. Section 3.5 presents initial experimental results indicating good scalability of BOA on problems of bounded difficulty. Finally, Section 3.6 summarizes the chapter.

Bayesian Optimization Algorithm (BOA)

(1) set $t \leftarrow 0$
randomly generate initial population $P(0)$

(2) select a set of promising strings $S(t)$ from $P(t)$

(3) construct the network B using a chosen metric and constraints

(4) generate a set of new strings $O(t)$ according to the joint distribution encoded by B

(5) create a new population $P(t+1)$ by replacing some strings from $P(t)$ with $O(t)$
set $t \leftarrow t+1$

(6) if the termination criteria are not met, go to (2)

Figure 3.1: The pseudo-code of the Bayesian optimization algorithm (BOA).

3.1 Description of BOA

The Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, and Cantú-Paz, 1998, 1999, 2000b) evolves a population of candidate solutions to a given problem by building and sampling Bayesian networks. BOA can be applied to the problems where candidate solutions are represented by fixed-length strings over a finite alphabet, but for the sake of simplicity, only binary strings will be considered in most of this chapter.

BOA generates the initial population of strings at random with a uniform distribution over all possible strings. The population is updated for a number of iterations (generations), each consisting of four steps. First, promising solutions are selected from the current population using a GA selection method, such as tournament and truncation selection. Second, a Bayesian network that fits the population of promising solutions is constructed. Third, new candidate solutions are generated by sampling the built Bayesian network. Fourth, the new candidate solutions are incorporated into the original population, replacing some of the old ones or all of them.

The above four steps are repeated until some termination criteria are met. For instance, the run can be terminated when the population converges to a singleton, the population contains a good enough solution, or a bound on the number of iterations has been reached. The pseudo-code of BOA is shown in Figure 3.1.

There are a number of alternative ways to perform each step. The initial population can be biased according to a prior knowledge about the problem (Schwarz & Ocenasek, 2000; Sastry, 2001a). Selection can be performed using any popular selection method. Various algorithms can be used to construct a model and a method for measuring the quality of each candidate model can also be chosen arbitrarily. Additionally, the measure of model quality can incorporate prior information about the problem to enhance the estimation and, as a consequence, to improve the efficiency.

The next section describes Bayesian networks and their semantics. The section then discusses techniques for learning and utilization of Bayesian networks.

3.2 Bayesian Networks

A Bayesian network (Howard & Matheson, 1981; Pearl, 1988) is defined by two components:

(1) **Structure.** The structure is encoded by a directed acyclic graph with the nodes corresponding to the variables in the modeled data set (in this case, to the positions in solution strings) and the edges corresponding to conditional dependencies.

(2) **Parameters.** The parameters are represented by a set of conditional probability tables specifying a conditional probability for each variable given any instance of the variables that the variable depends on.

Mathematically, a Bayesian network encodes a joint probability distribution given by

$$p(X) = \prod_{i=0}^{n-1} p(X_i|\Pi_i), \tag{3.1}$$

where $X = (X_0, \ldots, X_{n-1})$ is a vector of all the variables in the problem; Π_i is the set of parents of X_i in the network (the set of nodes from which there exists an edge to X_i); and $p(X_i|\Pi_i)$ is the conditional probability of X_i given its parents Π_i.

A directed edge relates the variables so that in the encoded distribution, the variable corresponding to the terminal node is conditioned on the variable corresponding to the initial node. More incoming edges into a node result in a conditional probability of the variable with a conjunc-

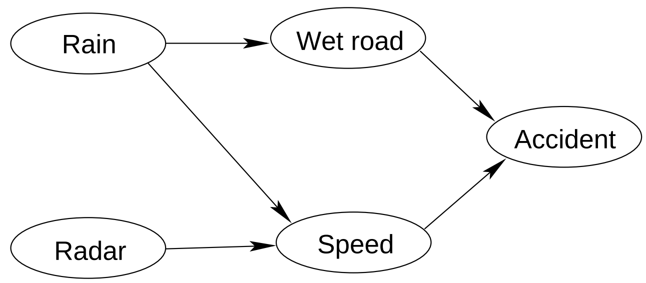


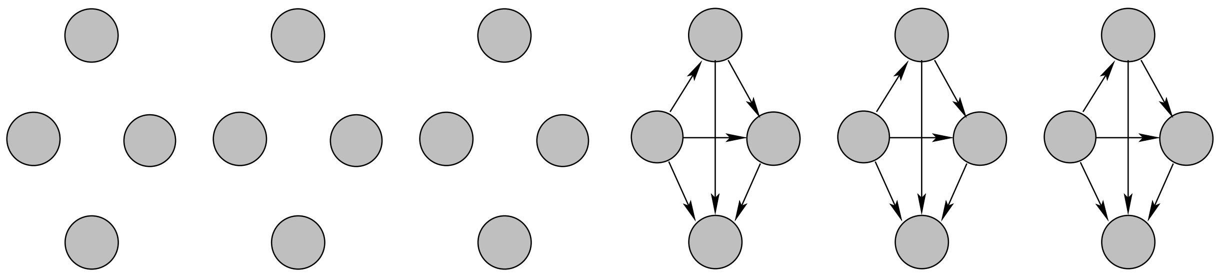
Figure 3.2: An example Bayesian network structure.

Accident	Wet Road	Speed	$p(\text{Accident}\|\text{Wet Road}, \text{Speed})$
Yes	Yes	High	0.18
Yes	Yes	Low	0.04
Yes	No	High	0.06
Yes	No	Low	0.01
No	Yes	High	0.82
No	Yes	Low	0.96
No	No	High	0.94
No	No	Low	0.99

Table 3.1: If we encode the speed of a car using two values (high and low), the conditional probability table for the probability of an accident could look as shown in this table. Note that the last four entries in the table are unnecessary, because they can be computed using the remaining ones (all conditional probabilities with a fixed conditions must sum to 1).

tional condition containing all its parents. In addition to encoding dependencies, each Bayesian network encodes a set of independence assumptions. Independence assumptions state that each variable is independent of any of its antecedents in the ancestral ordering, given the values of the variable's parents.

A simple example Bayesian network structure is shown in Figure 3.2. The example network encodes a number of conditional dependencies. For instance, the speed of the car depends on whether it is raining and/or radar is enforced. The road is most likely wet if it is raining. Additionally, the network encodes a number of simple and conditional independence assumptions. For instance, the radar enforcement is independent of whether it is raining or not. A more complex conditional independence assumption is that the probability of an accident is independent of whether the radar is enforced, given a particular speed and whether the road is wet. To fully specify the Bayesian network with the structure shown in the figure, it would be necessary to add a table of conditional probabilities for each variable. An example conditional probability table is shown in Table 3.1.



(a) No interactions. (b) Separable blocks of order 4.

Figure 3.3: A good model for the case of a problem with no interactions and a problem with separable subproblems of order 4.

It is important to understand the semantics of Bayesian networks in the framework of PMBGAs. Conditional *dependencies* will cause the involved variables to remain in the configurations seen in the selected population of promising solutions. On the other hand, conditional *independencies* lead to the mixing of bits and pieces of promising solutions in some contexts (the contexts are determined by the variables in the condition of the independency). The complexity of a proper model is directly related to a proper problem decomposition discussed in Chapter 1. If the problem was linear, a good network would be the one with no edges (see Figure 3.3(a)); the effects of using an empty network are the same as those of using population-wise uniform crossover. On the other hand, if the problem was composed of traps of order k, the network should be composed of fully connected sets of k nodes, each corresponding to one trap, with no edges between the different groups (see Figure 3.3(b)); the effects of using such a model would be the same as those of population-wise building-block crossover. More complex problems lead to more complex models, although many problems can be solved with quite simple networks despite the presence of nonlinear interactions of high order.

The following section discusses how Bayesian networks can be learned given a data set. Subsequently, we provide a simple method called forward simulation, which can be used for sampling the distribution encoded by a given Bayesian network and its parameters.

3.3 Learning Bayesian Networks

For a successful application of BOA, it is necessary that BOA is capable of *learning* a network that reflects the dependencies and independencies that decompose the problem at hand properly. There are two subtasks of learning a Bayesian network:

(1) **Learning the structure.** First, the structure of a network must be determined. The structure defines conditional dependencies and independencies encoded by the network.

(2) **Learning the conditional probabilities.** The structure also identifies conditional probabilities that must be specified for a complete model. After learning the structure, the values of the conditional probabilities with respect to the final structure must be learned.

In BOA, learning the parameters for a given structure is simple, because the value of each variable in the population of promising solutions is specified; in other words, the data is complete. To maximize the likelihood of the model with a fixed structure and complete data, the probabilities should be set according to the relative frequencies observed in the modeled data (in our case, the selected set of promising solutions). Thus, the parameters can be learned by iterating through all selected solutions and computing relative frequencies of different partial solutions.

As an example of learning the parameters of a Bayesian network, recall probabilistic uniform crossover, which can be represented by a Bayesian network with no edges. The parameters of an empty network consist of the probabilities of different values for each variable. Therefore, to determine the parameters of an empty network, it is sufficient to parse the population of promising solutions and compute the observed probabilities.

Learning the structure is a much more difficult problem. The algorithms for learning the structure of Bayesian networks have two components:

1. **A scoring metric.** A scoring metric measures quality of Bayesian network structures. In GA terminology, the scoring metric specifies the fitness of a structure. Usually, a scoring metric is proportional to the likelihood of the structure or it is equal to a combination of the likelihood and some penalty for complex models. However, other measures can be used, such as statistical tests on independence.

2. **A search procedure.** A search procedure searches the space of all possible network structures to find the best network with respect to a given scoring metric. The space of network structures can be restricted according to a bound on the complexity of networks or some other prior problem-specific knowledge.

Next, the section discusses several approaches to evaluating competing network structures. Subsequently, the section describes a greedy algorithm, which can be used to learn the structure of a Bayesian network given a scoring metric.

3.3.1 Scoring Metric

There are two approaches to measuring quality of competing network structures: (1) Bayesian metrics, and (2) minimum description length metrics. Bayesian metrics (Cooper & Herskovits, 1992; Heckerman, Geiger, & Chickering, 1994) measure the quality of each structure by computing a marginal likelihood of the structure with respect to given data and inherent uncertainties. Minimum description length metrics (Rissanen, 1978; Rissanen, 1989; Rissanen, 1996) are based on the assumption that the number of regularities in the data encoded by a model is somehow proportional to the amount of compression of the data allowed by the model.

Bayesian Metrics

Bayesian metrics (Cooper & Herskovits, 1992; Heckerman, Geiger, & Chickering, 1994) account for the uncertainty of network structures and their parameters by using Bayes rule and assigning prior distributions to both network structures as well as the parameters of each structure. The quality of a structure is measured by the marginal likelihood of the structure with respect to the given data. The marginal likelihood is computed by averaging the likelihood of the models conditioned on the observed data according to a prior distribution over all possible conditional probabilities in the model:

$$p(B|D) = \frac{p(B)}{p(D)} \int_{\theta} p(\theta|B) p(D|B,\theta)\, d\theta, \tag{3.2}$$

where B is the evaluated Bayesian network structure (without parameters); D is the data set; and each value of θ represents one possible way of assigning conditional probabilities in the network B. Furthermore, $p(B)$ is the prior probability of the network structure B, $p(\theta|B)$ is the prior probability of parameters θ (conditional probabilities) given B, and $p(D|B,\theta)$ denotes the probability of D given the network structure B and its parameters θ. Since the probability of data denoted in the last equation by $p(D)$ is the same for all network structures, $p(D)$ is usually omitted when evaluating the structures.

To compute the marginal likelihood, a prior probability distribution over the parameters of each structure must be given. The Bayesian-Dirichlet metric (BD) (Cooper & Herskovits, 1992; Heckerman, Geiger, & Chickering, 1994) assumes that the conditional probabilities follow Dirichlet distribution and makes a number of additional assumptions, yielding the following score:

$$BD(B) = p(B)\prod_{i=1}^{n}\prod_{\pi_i}\frac{\Gamma(m'(\pi_i))}{\Gamma\left(m'(\pi_i)+m(\pi_i)\right)}\prod_{x_i}\frac{\Gamma\left(m'(x_i,\pi_i)+m(x_i,\pi_i)\right)}{\Gamma(m'(x_i,\pi_i))}, \tag{3.3}$$

where $p(B)$ is the prior probability of the network structure B; the product over x_i runs over all instances of x_i (in the binary case these are 0 and 1); the product over π_i runs over all instances of the parents Π_i of X_i (all possible combinations of values of Π_i); $m(\pi_i)$ is the number of instances with the parents Π_i set to the particular values given by π_i; and $m(x_i,\pi_i)$ is the number of instances with $X_i = x_i$ and $\Pi_i = \pi_i$. Terms $m'(\pi_i)$ and $m'(x_i,\pi_i)$ denote prior information about the statistics $m(\pi_i)$ and $m(x_i,\pi_i)$, respectively. Here we consider K2 metric, which uses an uninformative prior that assigns $m'(x_i,\pi_i) = 1$ and $m'(\pi_i) = \sum_{x_i} m'(x_i,\pi_i)$.

A prior distribution over network structures specified by term $p(B)$ can bias the construction toward particular structures by assigning higher prior probabilities to those preferred structures. Prior knowledge about the structure permits the assignment of higher prior probabilities to those networks similar to the structure believed to be close to the correct one (Heckerman, Geiger, & Chickering, 1994). The search can also be biased toward simpler models by assigning higher prior probabilities to models with fewer edges or parameters (Chickering, Heckerman, & Meek, 1997; Friedman & Goldszmidt, 1999; Pelikan, Goldberg, & Sastry, 2001). If there is no prior information about the network structure, the probabilities $p(B)$ are set to a constant and omitted in the construction (uniform prior).

For further details on Bayesian metrics, please refer to Cooper and Herskovits (1992) and Heckerman, Geiger, and Chickering (1994).

Minimum Description Length Metrics

Minimum description length metrics (Rissanen, 1978; Rissanen, 1989; Rissanen, 1996) are based on the assumption that the number of regularities in the data encoded by a model is somehow proportional to the amount of compression of the data allowed by the model. A model that results in the highest compression should therefore encode the most regularities. There are two major approaches to the design of MDL metrics. The first is based on a two-part coding where the score is negatively proportional to the sum of the number of bits required to store the (1) model, and (2) data compressed according to the model. The second approach is based on universal code, which normalizes the conditional probability of data given a model by the sum of the probabilities of all data sequences given the model. The normalized probability of the data is used as the basis for computing the number of bits required to compress the data.

In this work we consider one of the two-part MDL metrics called the Bayesian information criterion (BIC) (Schwarz, 1978) used previously in the extended compact genetic algorithm (ECGA) (Harik, 1999) and the estimation of Bayesian networks algorithm (EBNA) (Larranaga, Etxeberria, Lozano, & Pena, 2000a). In the binary case, BIC assigns the network structure a score

$$BIC(B) = \sum_{i=1}^{n} \left(-H(X_i|\Pi_i)N - 2^{|\Pi_i|}\frac{\log_2(N)}{2} \right), \tag{3.4}$$

where $H(X_i|\Pi_i)$ is the conditional entropy of X_i given its parents Π_i; n is the number of variables; and N is the population size (the size of the training data set). The conditional entropy $H(X_i|\Pi_i)$ is given by

$$H(X_i|\Pi_i) = -\sum_{x_i,\pi_i} p(x_i,\pi_i)\log_2 p(x_i|\pi_i), \tag{3.5}$$

where $p(x_i,\pi_i)$ is the observed probability of instances with $X_i = x_i$ and $\Pi_i = \pi_i$; and $p(x_i|\pi_i)$ is the conditional probability of instances with $X_i = x_i$ given that $\Pi_i = \pi_i$.

$H(X_i|\Pi_i)$ denotes the average number of bits required to store a value of X_i given a value of Π_i. BIC multiplies the entropy $H(X_i|\Pi_i)$ by the population size N to reflect the number of bits required to store the entire population. The term $\log_2(N)$ denotes the number of bits required to store one parameter of the model (one probability or frequency). The number of bits required to store each parameter is divided by two because only half of the bits really matter in practice (Friedman &

Yakhini, 1996). The term with the conditional entropy ensures that the more the information about the parents of a variable enables to compress the values of the variable, the higher the value of the BIC metric. The term with the $\log_2(N)$ introduces the pressure toward simpler models by decreasing the metric in proportion to the number of parameters required to fully specify the network.

For further details on the minimum description length metrics, please see Rissanen (1996) and Grünwald (1998).

According to our experience, Bayesian metrics tend to be too sensitive to noise in the data and often capture unnecessary dependencies. To avoid overly complex models, the space of network structures must usually be restricted by specifying a maximum order of interactions (Heckerman, Geiger, & Chickering, 1994; Pelikan, Goldberg, & Cantú-Paz, 2000b). On the other hand, MDL metrics favor simple models so that no unnecessary dependencies need to be considered. In fact, MDL metrics often result in overly simple models and require large populations to learn a model that captures all necessary dependencies.

3.3.2 Search Procedure

Learning the structure of a network given a scoring metric is a difficult combinatorial problem. In fact, it has been shown that finding the best network is NP-complete for most Bayesian and non-Bayesian metrics (Chickering, Geiger, & Heckerman, 1994); therefore, there is no known polynomial-time algorithm for finding the best network structure with respect to most scoring metrics. However, a simple greedy algorithm (Heckerman, Geiger, & Chickering, 1994) often performs well and has been successfully used in a number of difficult machine learning tasks.

The greedy algorithm performs an elementary graph operation that improves the quality of the current network the most until no more improvement is possible. The network structure can be initialized to the graph with no edges or the best tree graph computed using the polynomial-time maximum branching algorithm (Edmonds, 1967). In BOA, the initial structure can be also set to the structure learned in the previous generation. In all experiments presented in this thesis, the network is constructed from an empty network in every generation. There are three elementary operations that can be used:

The greedy algorithm for network construction

(1) initialize the network B (e.g., to an empty network)

(2) choose all simple graph operations that can be performed on the network without violating the constraints

(3) pick the operation that increases the score of the network the most

(4) perform the operation picked in the previous step

(5) if the network can no longer be improved under given constraints on its complexity or a maximal number of interactions has been reached, finish

(6) go to 2

Figure 3.4: The pseudo-code of the greedy algorithm for learning the structure of Bayesian networks.

1. **Edge addition.** An edge is added into the network to add a new dependency.

2. **Edge removal.** An existing edge is removed from the current network to remove an existing dependency and introduce a new independence assumption or make an existing independence assumption stronger.

3. **Edge reversal.** An existing edge is reversed to change the character of the corresponding dependency. Each edge reversal can be replaced by first removing the edge and then adding the reversed edge in its place.

The search is terminated when there is no operation that can improve the score of the current metric. It is necessary to ensure that after each performed operation on the network structure, the resulting graph represents a valid Bayesian network structure. Consequently, the operations that introduce cycles in the structure must be eliminated. Additionally, it is useful to limit the number of edges that end in any node to upper-bound the complexity of the final structure. Figure 3.4 shows the pseudo-code of the greedy algorithm described above. Figure 3.5 shows an example sequence of operators to learn the Bayesian network structure from Figure 3.2.

At the beginning of this section we said that searching for the best Bayesian network is NP-complete. This leads to an interesting observation: BOA is a search algorithm that uses another search algorithm, BOA searches within a search. Why should the problem of learning the structure

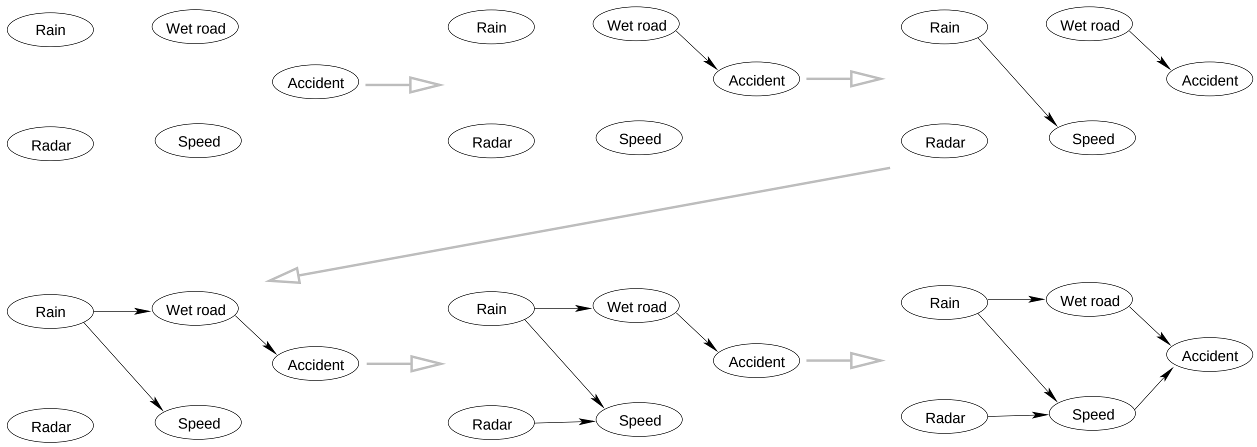


Figure 3.5: An example sequence of steps leading to the network structure shown earlier in Figure 3.2. For the sake of simplicity, only edge additions are used in this example.

of Bayesian networks be simpler than the original optimization problem BOA attempts to solve? It is premature to discuss this topic in great detail, but theoretical results of chapter 4 will show that even if the subproblems in a proper problem decomposition deceive GAs away from the optimum, the signal for constructing a Bayesian network will still lead in the right direction. The reason for this is that the search for a good model does not attempt to solve the optimization problem, but instead it learns interactions between the variables in a performance measure for the problem. Furthermore, BOA does not require the best network, it only needs a network that encodes all important interactions in a problem (or most of them). In addition to the these two arguments, an array of results presented throughout this thesis will support the above claim empirically.

3.4 Sampling a Bayesian Network

Once the structure and parameters of a Bayesian network have been learned, new candidate solutions are generated according to the distribution encoded by the learned network (see Equation 3.1).

The sampling can be done by forward simulation of Bayesian networks (Henrion, 1988), which proceeds in two steps. The first step computes an ancestral ordering of the nodes, where each node is preceded by its parents. The basic idea is to generate the variables in a certain sequence so that the values of the parents of each variable are generated prior to the generation of the value of the variable itself. The pseudo-code of the algorithm for computing an ancestral ordering of the variables is shown in Figure 3.6.

The algorithm for creating the ancestral ordering of the variables

(1) Mark all the variables as unprocessed.

(2) Empty the list of ordered variables.

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

(4) If any unmarked variables remain, go to 3

Figure 3.6: The pseudo-code of the algorithm for computing the ancestral ordering of the variables in the Bayesian network.

The algorithm for sampling the Bayesian network based on forward simulation

(1) Create ancestral ordering of the variables (see Figure 3.6).

(2) Generate the values of all the variables according to the ancestral ordering using the conditional probabilities encoded by the network.

(3) If more instances needed, go to 2.

Figure 3.7: The algorithm for sampling a given Bayesian network starts by ordering the variables according to the dependencies, yielding an ancestral ordering. The variables of each new solution are generated according to the ancestral ordering using conditional probabilities encoded by the network.

In the second step, the values of all variables of a new candidate solution are generated according to the computed ordering. Since the algorithm generates the variables according to the ancestral ordering, when the algorithm attempts to generate the value of each variable, the parents of the variable must have already been generated. Given the values of the parents of a variable, the distribution of the values of the variable is given by the corresponding conditional probabilities. The second step is executed to generate each new candidate solution. A more detailed description of the algorithm for generating new candidate solutions is shown in Figure 3.7.

3.5 Initial Experiments

This section presents the results of initial experiments using BOA on the aforementioned linear and trap problems. Additionally, BOA is tested on a composed deceptive function of order 3. All tested problems are of bounded difficulty—they can be decomposed into subproblems of bounded order. The performance of BOA is compared to that of the simple GA with uniform crossover and the stochastic hill climber using bit-flip mutation.

First, test problems are reviewed and discussed briefly. Next, experimental methodology is provided. Finally, the performance of BOA on the test problems is presented and compared to that of the simple GA with uniform crossover and the mutation-based hill climber.

3.5.1 Test Functions

Recall that onemax is defined as the sum of bits in the input binary string (see Equation 1.1). The optimum of onemax is in the string of all 1s. Onemax is a monotone unimodal function and therefore it is easy to optimize. The purpose of testing BOA on onemax is to show that BOA can solve not only decomposable problems of bounded difficulty but also those problems that are simple and can be efficiently solved by the mutation-based hill climber.

A composed trap function of order 5 is defined as the sum of single trap functions of order 5 over non-overlapping 5-bit partitions of solution strings (see Section 2.2.2 on page 27 for a detailed definition). The partitioning in a composed trap is fixed, but there is no information about the positions in each partition revealed to BOA. The two optima in each trap are defined as $f_{low} = 4$ and $f_{high} = 5$. Composed traps cannot be decomposed into subproblems of order lower than 5 and therefore they can be used to analyze the scalability of BOA on nontrivial problems of bounded difficulty.

A composed deceptive function of order 3 (Pelikan et al., 1998) is defined as the sum of single deceptive functions of order 3 over non-overlapping 3-bit partitions of solution strings. As in composed traps, the partitioning in composed deceptive functions is fixed, but there is no information about the positions in each deceptive block revealed to BOA. The fitness contribution of each block of 3 bits is given by

$$f^3_{deceptive}(u) = \begin{cases} 0.9 & \text{if } u = 0 \\ 0.8 & \text{if } u = 1 \\ 0 & \text{if } u = 2 \\ 1 & \text{otherwise} \end{cases}, \tag{3.6}$$

where u is the number of ones in the input block of 3 bits. The optimum of the composed deceptive function is in the string of all 1s. Composed deceptive functions mislead selectorecombinative bias into a local optimum in the string of all 0s if the deceptive partitions are decomposed. The composed deceptive function represents yet another example of a nontrivial problem of bounded difficulty; however, since the order of subproblems in composed deceptive functions is different compared to that in composed traps, a comparison of the performance of BOA on these two functions should reveal whether BOA's performance depends on the order of subproblems in a proper problem decomposition or not.

3.5.2 Experimental Methodology

For all tested problems, 30 independent runs are performed and BOA is required to find the optimum in all the 30 runs. The performance of BOA is measured by the average number of fitness evaluations until the optimum is found. The population size is determined empirically by a bisection method so that the resulting population size is the minimal population size required to ensure that the algorithm converges in all the 30 runs. There might be a 10% difference between the optimal population size and that used in the experiments, since the bisection method is only run until an interval with the width of at most 10% of its lower bound is found. BIC is used to construct a Bayesian network in each generation, and the construction always starts with an empty network.

Binary tournament selection without replacement is used in all the experiments. In most cases, better performance could be achieved by increasing selection pressure; however, the purpose of these experiments is not to show the best BOA performance, but to look at its scalability instead. The number of candidate solutions generated in each generation is equal to half the population size, and an elitist replacement scheme is used that replaces the worst half of the original population by

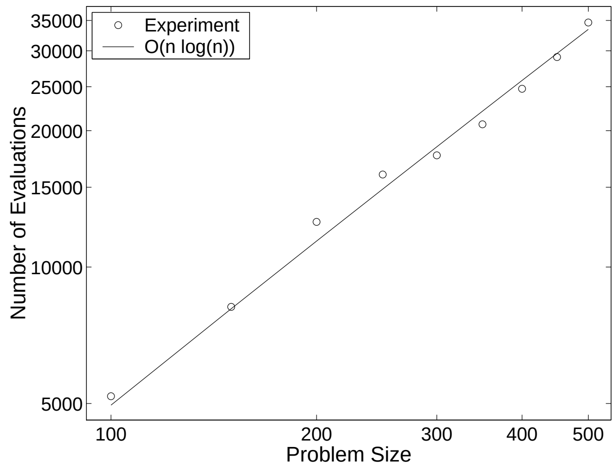


Figure 3.8: The number of evaluations until BOA finds the optimum on onemax of varying problem size averaged over 30 independent runs. The number of evaluations can be approximated by $O(n \log n)$.

offspring (newly generated solutions).

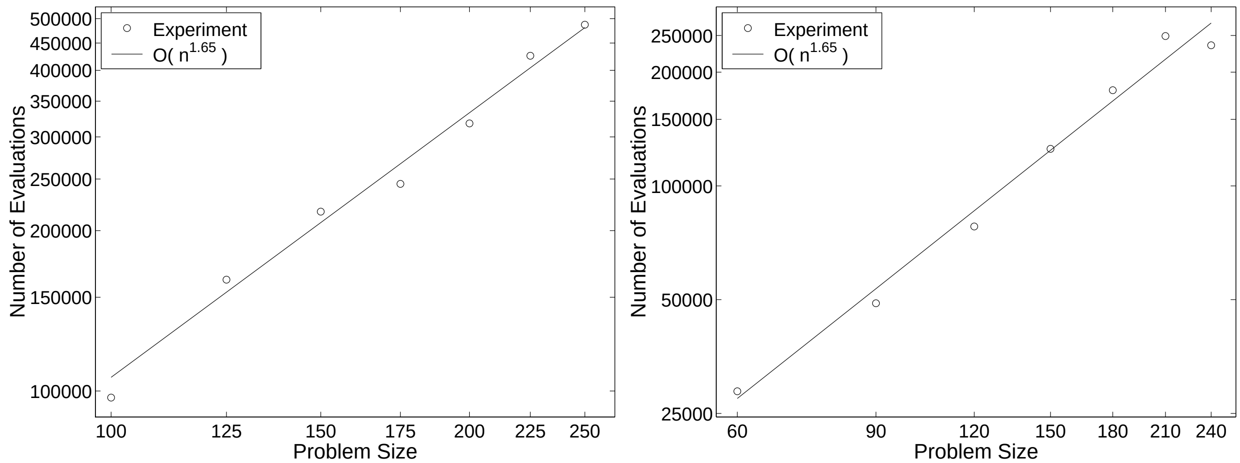
The GA with uniform crossover is also included in some of the results. All the parameters that overlap with BOA—except for population sizes—are set in the same fashion. Population sizes are also determined empirically by a bisection method. To maximize the mixing on onemax, the probability of applying crossover to each pair of parents is 1. On other problems, the crossover probability is 0.6. No mutation is used to focus on the effects of selectorecombinative search.

The performance of the mutation-based hill climber (see Section 2.2.1) is also compared to that of BOA and the GA with uniform crossover. The performance of the hill climber is computed according to a Markov-chain model of Mühlenbein (1992), which provides the theory for computing both the optimal mutation rate as well as the expected performance.

3.5.3 BOA Performance

Figure 3.8 shows the number of fitness evaluations until BOA finds the optimum of onemax. The size of the problem ranges from $n = 100$ to $n = 500$ bits. The number of fitness evaluations can be approximated by $O(n \log n)$. Therefore, the results indicate that BOA can solve onemax in a near-linear number of evaluations.

Figure 3.9 shows the number of fitness evaluations until BOA finds the optimum of the composed trap of order 5 and the composed deceptive function of order 3. The number of bits in composed



(a) Composed trap of order 5.

(b) Composed deceptive function of order 3.

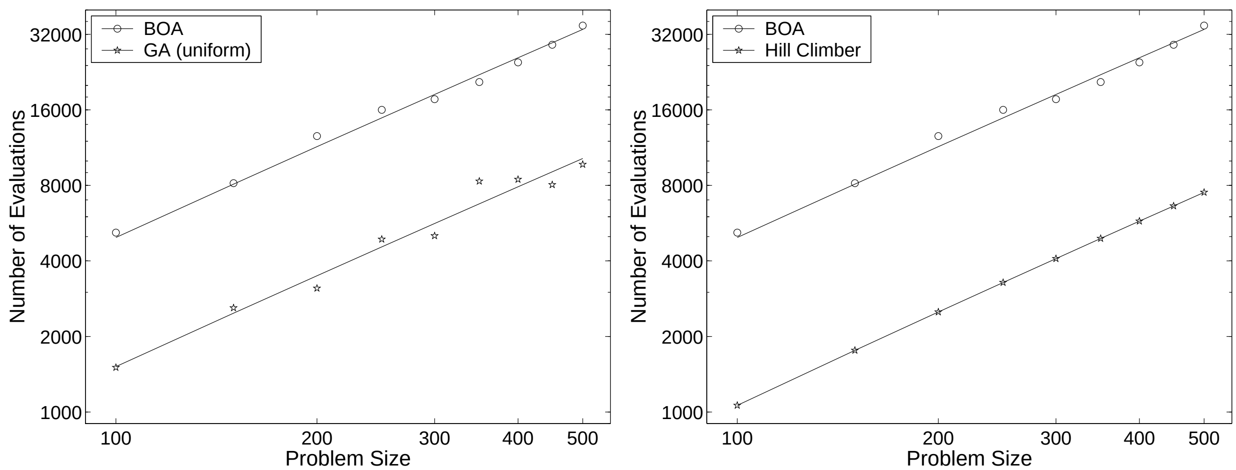
Figure 3.9: The number of evaluations until BOA finds the optimum on the composed trap of order 5 and the composed deceptive function of order 3. The number of evaluations can be approximated by $O(n^{1.65})$ in both cases.

traps ranges from $n = 100$ to $n = 250$ and the number of bits in composed deceptive functions ranges from $n = 60$ to $n = 240$. In both cases, the number of fitness evaluations can be approximated by $O(n^{1.65})$. Therefore, the results indicate that BOA can solve composed trap and deceptive functions in a subquadratic number of evaluations. Furthermore, the order of subproblems in a proper problem decomposition does not seem to affect the order of growth of the number of fitness evaluations.

Experiments on other decomposable problems indicate similar performance. In the case of onemax, the performance of BOA is close to the expected performance with population-wise uniform crossover, which is a "perfect" crossover to use in this case. The performance gets slightly worse for other decomposable problems where the discovery of a proper decomposition is necessary. However, in all the cases, the number of evaluations until reaching the optimum appears to grow subquadratically with the number of variables in the problem (problem size).

3.5.4 BOA vs. GA and Hill Climber

For onemax, both the hill climber and the simple GA with uniform crossover can be expected to converge in $O(n \log n)$ evaluations. The performance of the simple GA can be estimated by the



(a) BOA vs. the simple GA.

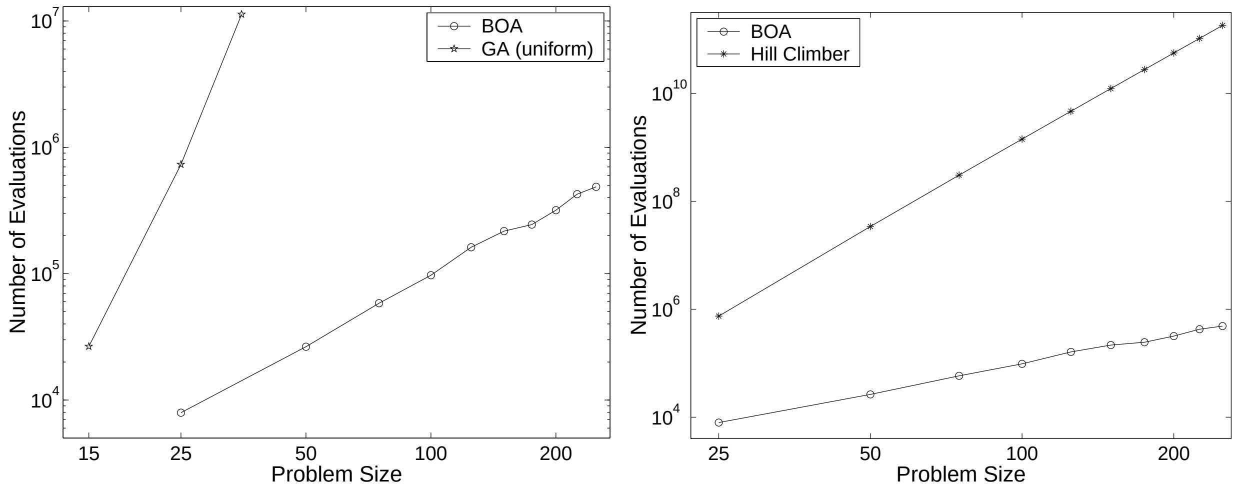
(b) BOA vs. the hill climber.

Figure 3.10: The comparison of BOA, the simple GA with uniform crossover, and the mutation-based hill climber on onemax.

gambler's ruin population-sizing model (Harik, 1999) and onemax convergence model (Mühlenbein & Schlierkamp-Voosen, 1993). For the simple GA with one-point or n-point crossover, the performance would get slightly worse because of slower mixing, which results in an increased number of generations. The performance of the hill climber can be estimated using the theory of Mühlenbein (1992).

Figure 3.10 compares the performance of the simple GA and the mutation-based hill climber with that of BOA for onemax. In all cases, the total number of evaluations is bounded by $O(n \log n)$; however, the performance of BOA is about 3.57-times worse than the performance of the simple GA, and the performance of the simple GA is about 1.3-times worse than the performance of the hill climber.

The reason for the worse performance of BOA is that onemax can be solved by processing each bit independently, but BOA introduces unnecessary dependencies, which lead to increased population-sizing requirements according to the gambler's ruin model (Harik, 1999). The comparison of the performances of the simple GA and the hill climber is inconclusive, because choosing a different selection pressure would change the results of the simple GA. However, it is important to note that the number of evaluations for all the algorithms grows as $O(n \ln n)$. That means that even for those simple problems that are ideal for mutation and uniform crossover, the use of



(a) BOA vs. the simple GA.

(b) BOA vs. the hill climber.

Figure 3.11: The comparison of BOA, the simple GA with uniform crossover, and the mutation-based hill climber on the composed trap function of order 5.

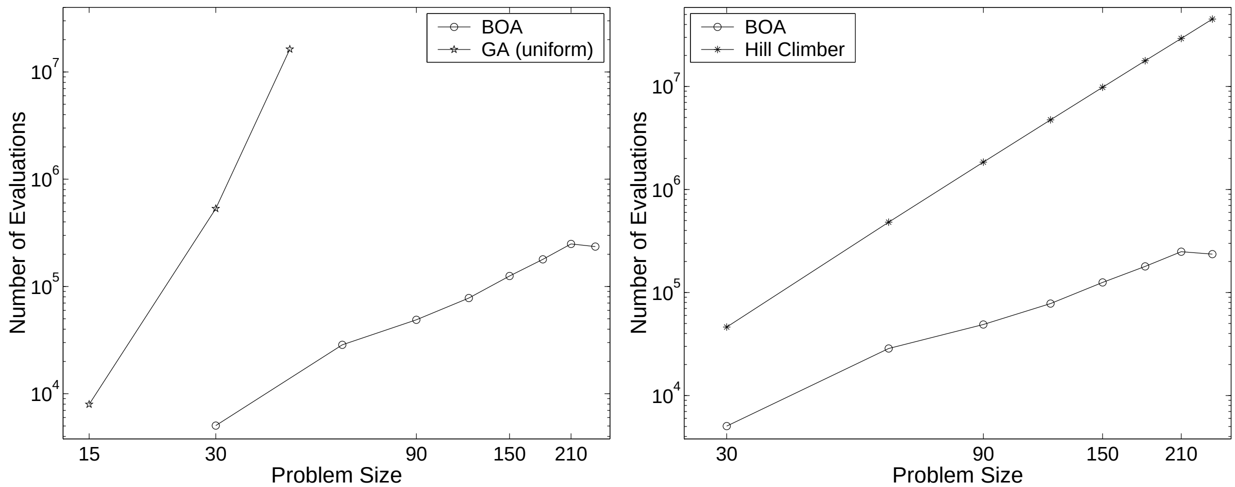
sophisticated search operators of BOA does not lead to a qualitative decrease in the performance.

Figure 3.11 compares the performance of the simple GA and the mutation-based hill climber with that of BOA for the composed trap of order 5. The performance of the GA and the hill climber dramatically changes compared to onemax.

To solve the composed trap of order 5, the simple GA requires exponentially large populations, because mixing is ineffective and requires exponentially large populations for innovative success (Thierens, 1995). Similar performance can be expected with other crossover methods, because the partitions of a proper problem decomposition are not located tightly in the strings. Already for the problem of size 50, even population sizes of a half million do not result in reliable convergence; the figure only shows the results on problems of sizes 15, 20, and 25 bits. Therefore, the performance of the simple GA changes from $O(n \log n)$ for onemax to $O(2^n)$ for the composed trap.

The performance of the hill climber changes from $O(n \log n)$ for onemax to $O(n^5 \ln n)$ for the composed trap of order $k = 5$ (Mühlenbein, 1992). In the general case, the complexity of the hill climber grows with the largest minimal order of building blocks k as $O(n^k \ln n)$.

Similar results can be observed for the composed deceptive function of order 3 (see Figure 3.12).



(a) BOA vs. the simple GA.

(b) BOA vs. the hill climber.

Figure 3.12: The comparison of BOA, the simple GA with uniform crossover, and the mutation-based hill climber on the composed deceptive function of order 3.

The number of evaluations for the simple GA grows exponentially, and the number of evaluations for the hill climber grows as $O(n^3 \log n)$.

To summarize, there are three important observations:

(1) **All on onemax.** BOA, GA, and the hill climber find the optimum of onemax in approximately $O(n \log n)$. However, BOA is outperformed with respect to the magnitude of the number of evaluations.

(2) **GA and the hill climber on trap and deceptive functions.** The performance of the simple GA and the mutation-based hill climber significantly suffers from an increased order of the problem decomposition. Both algorithms become intractable for problems of moderate difficulty.

(3) **BOA.** BOA requires a subquadratic number of evaluations until it finds the optimum of all tested problems of bounded difficulty; these results are supported by theory in the next chapter. Furthermore, the performance of BOA does not depend on the location of positions corresponding to each subproblem in a proper decomposition. BOA is capable of both finding a proper decomposition of a problem and exploiting that decomposition to solve the problem quickly, accurately, and reliably.

3.6 Summary

This chapter described the Bayesian optimization algorithm (BOA), which uses Bayesian networks to model promising solutions and sample new candidate solutions. Additionally, the chapter provided basic background of learning Bayesian networks, which is one of the core components of BOA. Finally, the chapter tested BOA on several problems of bounded difficulty. A summary of the key points of this chapter follows:

- Bayesian optimization algorithm (BOA) replaces crossover and mutation of GAs by the following two steps:

  1. **Learn a Bayesian network.** Given the set of promising solutions after selection, learn a Bayesian network that best fits the selected solutions.
  2. **Sample the learned network.** Generate new points according to the distribution encoded by the network learned in the previous step.

- BOA implementations can vary in the choice of a selection operator, methods for learning Bayesian networks, methods for evaluating each candidate network, methods for sampling the learned model, and methods for incorporating new candidate solutions into the parent population.

- Learning the structure of a network can be split into (1) the problem of measuring the quality of each candidate network and (2) the problem of searching for the best network with respect to a given measure.

- There are two approaches to measuring the quality of each candidate network:

  1. **Bayesian metrics.** Bayesian metrics account for the uncertainty of network structures and their parameters by specifying prior distributions for both network structures and parameters of each structure. The quality of a model grows with the marginal likelihood of the model given the population of promising solutions and prior distributions encoding inherent uncertainties.
  2. **Minimum description length metrics.** Minimum description length metrics are based on the assumption that the more regularities in the data a model encodes, the

higher lossless compression can be achieved when using the model to compress the data. The quality of a model decreases with the number of bits required to compress the population of promising solutions using the model.

- Unfortunately, the problem of finding the best network with respect to most Bayesian and non-Bayesian metrics is NP-complete. The good news is that a simple greedy algorithm performs well on most practical problems. The greedy algorithm iteratively performs elementary operations on the current network that improve the quality of the network the most until no more improvement is possible. Elementary operations usually consist of (1) adding a new edge into the current network, (2) removing an existing edge from the current network, and (3) reversing an edge in the current network.

- The greedy algorithm for network construction can start with an empty network, the network from the previous generation, or the network generated by the polynomial maximum branching algorithm for constructing tree models.

- To sample the learned Bayesian network using forward simulation, the variables are first ordered so that each variable is preceded by its parents. The values of the variables in each new solution are then generated according to the computed ordering using conditional probabilities encoded by the network.

- The results of initial experiments indicate that BOA is capable of solving problems of bounded difficulty in a subquadratic number of fitness evaluations. For onemax the number of evaluations grows as $O(n \ln n)$, and for composed trap and deceptive functions the number of evaluations grows as $O(n^{1.65})$.

- On the other hand, the performance of the simple GA and the mutation-based hill climber dramatically deteriorates as the problem becomes more difficult. For composed traps of order 5, for instance, the number of fitness evaluations grows as $O(2^n)$ for the GA with uniform crossover, and $O(n^5 \ln n)$ for the hill climber.

Chapter 4

Scalability Analysis

The empirical results of the last chapter were tantalizing. Easy and hard problems were automatically solved without user intervention in polynomial time. This raises an important question: How is BOA going to perform on other problems of bounded difficulty?

The purpose of this chapter is to analyze the scalability of BOA on problems of bounded difficulty. In particular, the chapter considers the number of evaluations of the objective function until the optimum is found with high confidence. The total number of evaluations is computed by (1) estimating an adequate population size that ensures reliable convergence to the optimum, (2) approximating the number of generations until convergence, and (3) making a product of these two quantities. Empirical results are then compared to the developed theory.

The chapter starts by arguing that the number of evaluations is a reasonable measure of computational complexity of BOA. Section 4.2 provides background of the GA population-sizing theory. Section 4.3 focuses on the BOA population-sizing theory. Section 4.4 discusses background of GA time-to-convergence theory, which estimates the number of generations until convergence assuming a sufficiently large population. Section 4.5 presents the convergence model for population-wise uniform crossover (univariate marginal distribution) on onemax and discusses how the model can be extended to the more general case of BOA. Two bounding types of scaling the subproblems in the problem decomposition are considered: (1) uniform scaling, and (2) exponential scaling. The scaling in any decomposable problem of bounded difficulty is expected to lie between the two cases. Section 4.6 puts all the pieces of the scalability puzzle together and validates the developed theory with experiments. Finally, Section 4.8 summarizes the chapter.

4.1 Time Complexity and the Number of Evaluations

One of the important characteristics of black-box optimization algorithms is the number of evaluations until reliable converge to the optimum. There are two primary reasons why considering the number of evaluations makes sense. The first reason is that in complex real-world applications it becomes expensive to evaluate each new solution; even a second for evaluating one solution usually overshadows the remaining overhead of the optimization method. The second reason is that per-evaluation computational overhead grows usually as a low-order polynomial of the problem size. Good scalability with respect to the number of evaluations thus implies good scalability with respect to the more traditional measures of computation complexity.

Let us assume that in each generation the number of new candidate solutions that must be evaluated is equal to a certain proportion of the size of the original population before selection,

$$E_g = cN, \tag{4.1}$$

where $c \in (0, 1)$ is a constant and N is the size of the population before selection. For example, if the size of the offspring population is equal to the size of the original population before selection, then $c = 1$. On the other hand, if only half of the original population is replaced by offspring, then $c = 0.5$.

The number of evaluations E until convergence is then given by

$$E = N + E_g G = N + c(N \times G), \tag{4.2}$$

where G is the number of generations until convergence. Note that N, G, and E are in fact functions of the problem size and the type of the problem, and that the focus is on the growth of E with respect to the problem size. Since the number of generations does not decrease with the problem size,

$$E = O(N \times G). \tag{4.3}$$

Therefore, to compute the growth of E with respect to the problem size, it is sufficient to compute the required population size and the number of generations until convergence, both expressed in terms of the problem size.

The following section provides background of the GA population-sizing theory. Section 4.3 presents the BOA population-sizing model. Section 4.4 provides background of GA theory for estimating the number of generations until convergence. Section 4.5 focuses on the number of generations until reliable convergence in BOA. Finally, Section 4.6 combines the pieces of theory to compute the overall number of evaluations required by BOA on decomposable problems of bounded difficulty.

4.2 Background of GA Population-Sizing Theory

GA population-sizing theory attempts to estimate an adequate population size for ensuring reliable convergence to the optimum. There are three important factors influencing the population sizing in GAs (assuming that crossover combines and preserves the building blocks effectively):

1. **Initial supply.** The population must be large enough to ensure that there is a sufficient supply of alternative solutions to each subproblem.

2. **Decision making.** The population must be large enough to ensure that decision making between alternative solutions to each subproblem is not misled by the noise from the remaining subproblems and that the best partial solution (the building block or BB) indeed wins.

3. **Genetic drift.** The population must be large enough to ensure that if the subproblems converge in several phases, there is an adequate initial supply of alternative solutions to each subproblem once the algorithm gets to the beginning of the subproblem's phase.

The next section discusses population-sizing models that focus on the initial supply of building blocks (BBs). Subsequently, population-sizing models concerned with the decision making are reviewed briefly. Finally, models that deal with genetic drift are examined.

4.2.1 Having an Adequate Initial Supply of BBs

Recall the GA simulation for onemax presented in Section 1.3. Without making sure that there are enough 1s in each position of the initial population, the GA could not find the optimum (regardless of how well or how badly it would recombine). To ensure that the solution can be found by combining bits and pieces of promising solutions, it is necessary that there is enough raw material to start with. Initial-supply population-sizing models (Holland, 1975; Goldberg, Sastry, & Latoza, 2001; Goldberg, 2002) focus on the initial supply of this raw material and bound the population size so that the initial population contains enough BBs to enable the selectorecombinative search of GAs to juxtapose these BBs and "build" the optimum.

Let us start with some simple mathematics to motivate initial-supply models, followed by a short review of past work on this topic. Assuming that the initial population of N binary strings is generated at random, the expected number of copies of any partial solution of order k is

$$m(BB_k) = \frac{N}{2^k}. \tag{4.4}$$

This suggests that to ensure a fixed number of copies of each BB, the population size should grow at least exponentially with the BB size. Of course, the actual number of copies can fluctuate from its expected value, because the initial population is generated at random (and almost anything can happen). Nonetheless, for the purposes of this thesis, this level of understanding of the inital supply of BBs is sufficient.

The importance of the initial supply of BBs was first recognized by Holland (1975), who computed the number of BBs that receive a specified number of copies in the initial population using Poisson distribution. Goldberg (1989b) refined Holland's model by using binomial distribution and applied the resulting model to population sizing. Reeves (1993) proposed an initial-supply population-sizing model for BBs of unit size and fixed cardinality. Poli, Langdon, and O'Reilly (1998) looked at the required number of copies of each BB to prevent its loss. Most recently, Goldberg, Sastry, and Latoza (2001) computed the necessary population size to ensure that every BB gets at least one copy in the initial population using alphabets of arbitrary cardinality.

4.2.2 Deciding Well Between BBs and their Competitors

Besides having an adequate initial supply of BBs, another important factor that determines the success of GAs is the one of *deciding well* between BBs and their competitors. Naturally, each BB should get more copies in the offspring population than its competitors do. However, as it was recognized by Holland (1973), the decision making in GAs is a statistical one, and the population size must be set accordingly to ensure that good decisions are made with high probability. Holland illustrated the statistical nature of decision making using a 2^k-armed bandit model. Later, De Jong (1975) developed equations for the 2-armed bandit and noted the importance of *noise* in the GA decision making. In short, the decision-making for a particular BB is affected by *noise* from the fitness contributions of the remaining partitions of the problem decomposition (the context); this noise is often referred to as *collateral noise*.

The effects of collateral noise can be illustrated by looking at the GA simulation on onemax shown in Figure 1.2. Although the last bit of 00001 is contained in the optimum and it increases the fitness of each solution independently of its context, the fitness of 00001 is worse than that of 10010. If 00001 and 10010 compete in a tournament, the proportion of 1s in the last position will *decrease*. However, on average, the performance of solutions with 1 in the last position should increase. Decision-making population-sizing models estimate the required size of a population to ensure that good decisions are made; in the onemax case, good decision making should increase the proportion of 1s in each position of the population over time.

Goldberg and Rudnick (1991) computed the variance of collateral noise using Walsh analysis and considered the ramifications of the result for the decision making. The variance of collateral noise formed the basis of the first practical decision-making population-sizing model (Goldberg, Deb, & Clark, 1992). The proposed model reduced decision making to the two best partial solutions of a subproblem—the BB and its toughest competitor. It estimated the required population size so that each BB wins over its best competitor in the same partition; once that is ensured, it is natural to expect the BB to win over its remaining competitors in the same partition as well. This model was a bit pessimistic, it required each BB to win in the first generation. Assuming that the problem is decomposable into similar subproblems of bounded order, the resulting population-sizing estimate was computed as

$$N = 2c(\alpha)2^k m' \frac{\sigma_{bb}^2}{d^2}, \tag{4.5}$$

where $c(\alpha)$ is the square of the ordinate of a unit normal distribution where the probability equals to α; α is the probability of failure; k is the order of the considered BB; m' is one less than the number m of BBs (i.e. $m' = m - 1$); σ_{bb}^2 is the root mean square (RMS) fitness variance of the fitness contributions in the BB's partition; and d is the difference between the fitness contributions of the BB and its toughest competitor.

Harik, Cantú-Paz, Goldberg, and Miller (1997, 1999) refined the above population-sizing estimate by eliminating the requirement for successful decision making in the first generation, and modeling subsequent generations using gambler's ruin model in one dimension (Feller, 1970). Assuming perfect mixing, the bound on the population size sufficient to find a solution containing each BB with the probability $(1-\alpha)$ was reduced to

$$N = -2^{k-1}\ln(\alpha)\frac{\sigma_{bb}\sqrt{\pi m'}}{d}. \tag{4.6}$$

Empirical results with tightly-encoded deceptive BBs and the two-point crossover matched the theory very well (Harik, Cantú-Paz, Goldberg, & Miller, 1997). Thus, with perfect mixing, the required population size in GAs grows proportionally to the square root of the number of BBs in a problem. The gambler's ruin population-sizing model was later extended to accomodate noise in the fitness function (Harik, Cantú-Paz, Goldberg, & Miller, 1999).

An empirical population-sizing model for onemax, truncation selection, and uniform crossover was discussed by Mühlenbein and Schlierkamp-Voosen (1993); this model agrees with the Gambler's ruin model for this special case and estimates the population size as $O(\sqrt{n}\log n)$.

4.2.3 Genetic Drift

Genetic drift causes some partial solutions to be lost even if their fitness values are the same or better than those of their competitors. This could become a problem if the contributions of the different subproblems are scaled so that the subproblems converge in several phases, where in each phase only some of the subproblems matter. Consequently, when the subproblem's phase starts, the alternative solutions corresponding to this subproblem may have already been eliminated. Much of the following discussion on the drift population-sizing models is motivated by the works of Rudnick (1992), Thierens, Goldberg, and Pereira (1998), Lobo, Goldberg, and Pelikan (2000), Rothlauf (2001), and Albert (2001).

To illustrate the importance of genetic drift in sizing the populations for some problems, it is

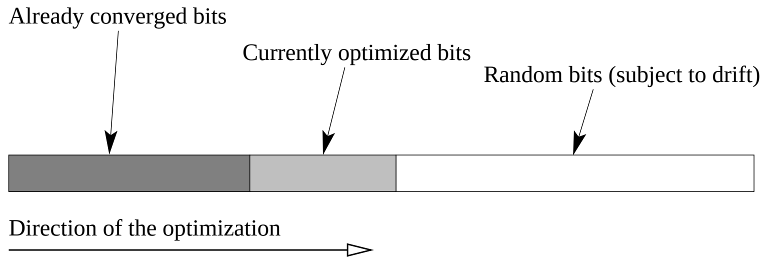


Figure 4.1: Convergence on a problem with exponentially scaled subproblems. The variables converge sequentially. At any point in time, only a small subset of variables matters; other variables have either converged already or their values do not influence the fitness enough to be affected by selection.

helpful to consider the *binary integer* fitness function (Rudnick, 1992) defined as

$$f_{bin}(X) = \sum_{i=0}^{n-1} 2^{n-i-1} X_i, \tag{4.7}$$

where $X = (X_0, \ldots, X_{n-1})$ is the input binary string of size n. The binary integer fitness decodes the binary number and uses the decoded value to determine its fitness. For example, $f_{bin}(001) = 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1$ and $f_{bin}(110) = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 6$.

Similarly as in onemax (see Equation 1.1 on page 13) the optimum of the binary integer is in the string of all ones and the function is monotonous. However, in the binary integer, the ith bit matters more than all the remaining bits to its right. As the result, selection will always put pressure only on one or a very small subset of bits and the GA will converge sequentially, one bit after another, taking one or a couple of generations for each single bit. At any point in time, some bits are already fixed, one or a couple of bits are just in the process of converging, and the remaining bits fluctuate randomly due to the stochasticity of selection (Rudnick, 1992). The random fluctuations of the partial solutions whose fitness contributions are too low are often called *genetic drift*. See Figure 4.1 to visualize of this process.

Due to genetic drift, once the GA starts optimizing the last bit (which did not matter in the selection process so far) its optimal value (in this case, 1) might be already gone. The drift population-sizing models ensure that when a particular subproblem starts to matter, there are still enough partial solutions to this subproblem to find the global optimum. There are two extreme ways of scaling the subproblems. At one extreme, all the subproblems are scaled the same and converge at the same time. At another extreme, there is one phase per variable, and the variables converge sequentially. The problems from the first extreme are taken care of by the initial-supply and decision-making models. The problems between the two extremes are discussed next.

It can be shown (Goldberg & Segrest, 1987; Asoh & Mühlenbein, 1994; Lobo, Goldberg, & Pelikan, 2000) that, assuming that a proper crossover is used and that the considered subproblem does not play a role in the selection at the time, the expected time for losing the partial solution due to genetic drift (the drift time) is given by

$$t_{drift} = cN, \tag{4.8}$$

where N is the population size, and the constant c depends on the initial proportion of the considered partial solution. Note that by enlarging the population size, any drift time can be achieved. In other words, the time until any partial solution is eliminated due to genetic drift can be increased arbitrarily by enlarging the population.

To find out how long the waiting must be, it is necessary to compute the number of generations until the start of each phase. The number of phases can be upper-bounded by the size of the problem (since in each phase at least one variable must fully converge):

$$n_{phase} \leq n. \tag{4.9}$$

Let us denote the number of subproblems of bounded order that play role in the ith phase by n_i, where $i \in \{1, \ldots, n_{phase}\}$. Each variable must be contained in at most one phase:

$$\sum_i n_i \leq n. \tag{4.10}$$

The number of generations required to converge in the ith phase can be bounded by (Mühlenbein & Schlierkamp-Voosen, 1993; Thierens & Goldberg, 1994)

$$t_i = O(\sqrt{n_i}). \tag{4.11}$$

Therefore, the number of generations until the ith phase starts is given by

$$T_i = \sum_{k=1}^{i-1} t_i = \sum_{k=1}^{i-1} a_i \sqrt{n_i}, \tag{4.12}$$

where a_i are constants depending on the type of the problem and the selection method used. We must now ensure that none of the partial solutions in the ith problem drifts earlier than T_i for every phase i:

$$t_{drift} > T_i. \tag{4.13}$$

Therefore, we require that for all phases i, the following equation must be satisfied

$$cN \geq \sum_{k=1}^{i-1} a_i \sqrt{n_i}. \tag{4.14}$$

The right-hand side of the last equation can grow at most linearly with the size of the problem, because

$$\sum_{k=1}^{i-1} a_i \sqrt{n_i} \leq \sum_{k=1}^{i-1} a_i n_i = O(n). \tag{4.15}$$

Therefore, $cN \geq dn$, where c and d are constants, yielding

$$N = O(n). \tag{4.16}$$

To summarize, the bound on the population size from the above equations can range from $O(\sqrt{n})$ (when all the subproblems are uniformly scaled as in onemax) to $O(n)$ (when each subproblem corresponds to a special phase as in the binary integer). The population size should thus grow at most linearly with the problem size.

The following section focuses on the population sizing in BOA.

4.3 Population Sizing in BOA

There are four factors that influence the population sizing in BOA (Pelikan, Goldberg, & Cantú-Paz, 2000a). The first three factors were discussed in the background part of this chapter; these factors can be adopted from the GA population-sizing theory (Holland, 1975; Goldberg, 1989b;

Goldberg, Deb, & Clark, 1992; Harik, Cantú-Paz, Goldberg, & Miller, 1997; Goldberg, Sastry, & Latoza, 2001; Goldberg, 2002). However, the GA theory assumes that crossover combines and preserves the building blocks properly; in the terminology of BOA, the probabilistic model was assumed to encode a proper problem decomposition. In addition to the first three factors, BOA must be capable of learning a proper, non-misleading, problem decomposition. Once this is ensured, the results of the GA population-sizing theory considering the first three factors can be applied.

This section presents the theory for sizing the populations in BOA so that a good model is found (Pelikan, Goldberg, & Cantú-Paz, 2000a; Pelikan, Sastry, & Goldberg, 2001). To simplify the analysis, we require that BOA finds a good enough model in the first generation. Although the population-sizing model derived under such an assumption is somewhat pessimistic because BOA can still recover after missing some important dependencies in the first generation, the resulting bound on the population size estimate fits the actual empirical results well.

4.3.1 Road Map to BOA Population-Sizing Model

The section starts by discussing the sources of statistical dependencies in the population of promising solutions. Section 4.3.3 relates collateral noise to the problem size, states the important assumptions of the BOA population-sizing model, and provides basic notational conventions followed in the remainder of the section. Section 4.3.4 discusses the problem of deciding between adding and not adding an edge into the Bayesian network and defines the *critical population size*, which is the minimum population size for finding the dependency under consideration. Section 4.3.5 computes the probabilities of one partition of the problem decomposition after applying binary tournament selection. The section shows that any two positions can be treated separately even if they are contained in a bigger partition.

Section 4.3.6 computes the critical population size for the general two-bit case. Section 4.3.8 discusses the effect of using finite populations on the difference between the expected frequencies after applying binary tournament selection and the actual ones. Section 4.3.9 compares the developed theory to the empirical results. Section 4.3.10 summarizes and discusses the results of the BOA population-sizing theory. The chapter then continues by discussing the number of generations until BOA convergence.

X	$p(X)$
00	0.25
01	0.25
10	0.25
11	0.25

(a) Initial (random) population

X	$p(X)$
00	0.0625
01	0.25
10	0.25
11	0.4375

(b) Population after binary tournament

Table 4.1: The proportions of the solutions on the two-bit onemax before and after binary tournament selection. The population is assumed to be infinite.

If the reader is not interested in the detailed theory, he or she may skip over most of this section without losing track and continue with the section summary on page 106.

4.3.2 Finding a Proper Model: The Good, the Bad, and the Ugly

Consider two bit positions, X_1 and X_2. If the fitness contributions of X_1 and X_2 are independent as in onemax, the model should not contain an edge between the two variables. On the other hand, if the contributions of X_1 and X_2 do depend on each other, it might be necessary to consider the dependency between the two variables. Not all nonlinearities must be covered; however, some nonlinearities could lead to deception as in the trap example in chapter 1. To avoid deception, BOA should consider as many nonlinearities as possible.

However, the construction of a model is guided by statistical dependencies and independencies over subsets of variables, while nonlinearities that must be detected are in the fitness. To find a model that encodes nonlinearities in the problem, it is necessary to ensure that selection "transforms" nonlinearities in the fitness into statistical dependencies in the population of promising solutions.

If the fitness contributions of X_1 and X_2 depend on each other, it can be expected that selection will lead to a detectable statistical dependency between the two variables. However, contrary to intuition, all commonly used selection methods also introduce the statistical dependencies between the variables whose fitness contributions are independent. This happens even if the infinite population is used. Let us illustrate this on an example. Consider a 2-bit onemax and an infinite population. As shown in Table 4.1(a), all the solutions 00, 01, 10, and 11 will occupy 25% of the initial population. After performing binary tournament selection, the frequencies will change as shown in Table 4.1(b). If the two positions were statistically independent, the following equation would have to be satisfied:

$$p(11) = p(1*)p(*1),$$

where $p(1*)$ and $p(*1)$ denote the total probability of 1 on the first and second positions, respectively. Substituting the probabilities from Table 4.1 yields $p(11) = 0.4375$ on the left-hand side of the above equation, and $p(1*)p(*1) = 0.6875^2 \approx 0.4727$ on the right-hand side. Clearly, $0.4375 \neq 0.4727$ and therefore the two variables are *not* statistically independent, even though their fitness contributions *are* independent.

Therefore, there are two kinds of statistical dependencies after selection; some dependencies are introduced by the nonlinearities in the fitness, while some dependencies are introduced by selection only. It is important that BOA discovers those dependencies that correspond to the nonlinearities (good dependencies), and that it ignores those dependencies that are introduced by selection only (bad dependencies). In particular, it is desirable that the following two conditions are satisfied:

1. The statistical dependencies corresponding to the fitness nonlinearities are significantly stronger than the statistical dependencies introduced by selection only.

2. The stronger the nonlinearity, the stronger the statistical dependency.

This section shows that both the conditions are satisfied. To make the computation tractable, the section first assumes that the frequencies behave as if an infinite population were used, although the focus is on the sufficient population size to discover the good dependencies and avoid the bad ones. Next, the assumption about the accuracy of the frequencies is justified by computing the minimal bound on the population size that ensures that with high confidence, the frequencies will be sufficiently close to the expected ones with an infinite population.

4.3.3 Assumptions and Notation

To make the theoretical analysis tractable, we make several assumptions about the problem. First, we assume that the fitness function is defined as the sum of the subfunctions applied to disjoint subsets of the variables of order k and that all the subfunctions are the same. In some cases, it is possible to apply the results of the theory also to the case where the subfunctions overlap; however, in most cases the theory would have to be extended to incorporate the effects of the overlap.

A particular partition of the problem decomposition is considered. Without loss of generality, we denote the variables in the considered partition by $X = (X_1, \ldots, X_k)$ or $Y = (Y_1, \ldots, Y_k)$ and their instantiations (blocks of k bits or partial solutions) by $x = (x_1, \ldots, x_k)$ and $y = (y_1, \ldots, y_k)$. The fitness contribution of X and Y is denoted by $g(X_1, \ldots, X_k) = g(X)$ and $g(Y_1, \ldots, Y_k) = g(Y)$, respectively. We denote the total fitness of the solutions containing the block $x = (x_1, \ldots, x_k)$ by $F(x)$ (note that $F(x)$ is a random variable) and we assume that the contributions of the remaining variables can be modeled by a normal distribution with the variance proportional to the size of the problem:

$$F(x) \sim N(\mu_x, \sigma_N^2), \tag{4.17}$$

where μ_x is the average fitness of the solutions containing $x = (x_1, \ldots, x_k)$, and σ_N^2 is the variance of the fitness contributions of the remaining variables (collateral noise). Additionally,

$$\sigma_N^2 \propto n, \tag{4.18}$$

where n is the size of the problem. The above assumption can be justified by the central limit theorem for all decomposable problems of bounded difficulty where the fitness contribution of each subproblem is of the same magnitude and the order of the subproblems is bounded by a constant. If the magnitude of the contributions varies from one subproblem to another, the population sizes required for building a good model decrease, because in each generation only a subset of the subproblems will matter.

By $p(x) = p(x_1, \ldots, x_k)$, we denote the probability of the partial solution x in the selected population of promising solutions. The probability is computed as the relative frequency of the partial solution x in the selected population. The probability of the other partial solutions is denoted in a similar fashion. For example, $p(x_1, x_2)$ denotes the probability of the solutions with $X_1 = x_1$ and $X_2 = x_2$, $p(x_2)$ denotes the probability of the solutions with $X_2 = x_2$, and so forth. The probability distribution of $p(x)$ is denoted by $p(X)$. The probabilities *before* selection are denoted by $p_{init}(x)$.

Additionally, as mentioned above, in many derivations we assume that the probabilities follow their expected behavior in the case of an infinite population, although in practice we can expect additional noise due to the finite size of the population. The assumption is later justified by bounding the population size so that the frequencies are close enough to their expected values with high confidence.

We consider only binary tournament selection (see page 11). Although the results seem to hold with other selection methods, the theoretical analysis becomes intractable. The empirical results for tournament selection with bigger tournaments are presented to justify this claim. The selected population is assumed to be of the same size as the population before selection is.

External noise in the fitness function can be incorporated into the theory in a straightforward manner, if the noise can be approximated by a zero-mean normal distribution. If the variance of external noise does not grow faster than linearly with the size of the problem, all the above assumptions will remain satisfied. However, if external noise grows faster than linearly with the size of the problem, the theory would have to be modified slightly; however, all the modifications are straightforward and are therefore omitted for the sake of simplicity.

4.3.4 Edge Additions and the Critical Population Size

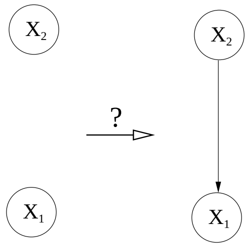
Consider the decision making between the following two cases:

(1) Add an edge from X_2 to X_1.

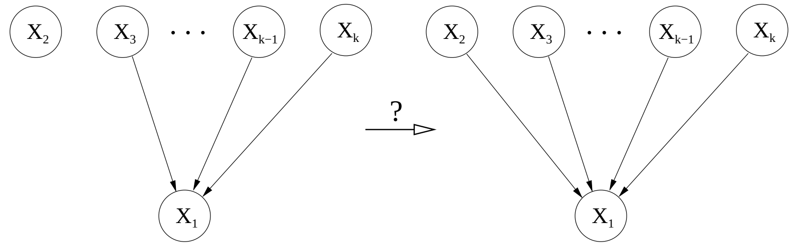
(2) Don't add the edge from X_2 to X_1.

To decide whether to add or not add the edge, we must compare the values of the used scoring metric for the current network with and without the edge, and choose the better alternative. Since both MDL and Bayesian metrics are decomposable, it is sufficient to look at the term corresponding to the node X_1.

There are two cases. Figure 4.2 illustrates both cases. In the first case (see part (a) in the figure), X_1 is isolated; in the second case (see part (b) in the figure), several edges that end in X_1 already exist in the network. The first case is first analyzed in detail. Section 4.3.7 extends the results of the analysis to the second, more general, case.



(a) Adding the first edge



(b) Adding an additional edge

Figure 4.2: There are two cases to consider when making a decision between adding and not adding an edge. The first case assumes that there are no other edges into the terminal node of the considered edge, the second assumes that there are a number of such edges.

The score assigned by BIC to X_1 without the edge from X_2 to X_1 is given by

$$BIC(X_1) = -H(X_1)N - \frac{\log_2 N}{2}, \tag{4.19}$$

where $H(X_1)$ is the entropy of X_1, and N is the number of selected solutions (the size of the selected population). After adding an edge from X_2 to X_1, the new score for X_1 is given by

$$BIC(X_1 \leftarrow X_2) = -H(X_1|X_2)N - \log_2 N, \tag{4.20}$$

where $H(X_1|X_2)$ is the conditional entropy of X_1 given X_2. For an addition of the edge $X_2 \rightarrow X_1$, the following inequality must be satisfied:

$$BIC(X_1 \leftarrow X_2) > BIC(X_1). \tag{4.21}$$

By substituting the equations 4.19 and 4.20 into the last equation, we get

$$(H(X_1) - H(X_1|X_2))\, N - \frac{\log_2 N}{2} > 0. \tag{4.22}$$

Let us denote the difference between the marginal and conditional entropies of X_1 by D:

$$D = H(X_1) - H(X_1|X_2). \tag{4.23}$$

If X_1 and X_2 are not statistically independent, D is strictly positive. The positivity of D is later supported by an exact calculation of D in the general case; in particular, D is shown to be positive if X_1 and X_2 contribute to the fitness in some way. If X_1 and X_2 do not influence the fitness, $D = 0$ and the model will not add any dependency between the two variables.

Since $D > 0$ and the linear term in Equation 4.22 grows faster than the logarithmic one, Equation 4.22 will be satisfied for a large enough N. Intuitively, when the two variables are not independent, for a big enough population size, the dependency should be discovered. We call the sufficient population size for the discovery of the dependency $X_2 \rightarrow X_1$ the *critical population size* and denote it by N_{crit}. To determine N_{crit}, the following equation must be solved for N:

$$N - \frac{\log_2 N}{2D} = 0. \tag{4.24}$$

The above equation has two solutions but there is no closed form for either of these solutions. The dependency is discovered for the population sizes lower than the first (lower) solution or greater than the second (greater) one. The first solution is approximately equal to $1 + 2D \ln 2$. However, since even for small problems the value of D is very small, this solution is of no interest in our case. N_{crit} is therefore defined as the larger of the two solutions of the last equation.

If the ratio $\frac{1}{2D}$ is large enough, the larger of the two solutions of the above equation follows a power law is thus of the form $\alpha \left(\frac{1}{2D}\right)^{\beta}$, where $\alpha \sim 8.34$ and $\beta \sim 1.05$. More specifically,

$$N_{crit} = 8.33 \left(\frac{1}{2D}\right)^{1.05} = 4.027\ D^{-1.05} \tag{4.25}$$

Since for an increasing problem size the magnitude of D decreases inversely proportionally to the number of decision variables in the problem or even faster, $\frac{1}{2D}$ is going to be large and the above approximation can be used to determine N_{crit}. Figure 4.3 shows the numerical solution and its approximation using Equation 4.25. The solutions corresponding to the two methods are practically indistinguishable.

In order to apply our results to the scale-up behavior of BOA with BIC metric, we are interested in the *growth* of N_{crit} with respect to the size of the problem (the number of decision variables). Using the approximation given in Equation 4.25, it can be shown that the growth of N_{crit} is

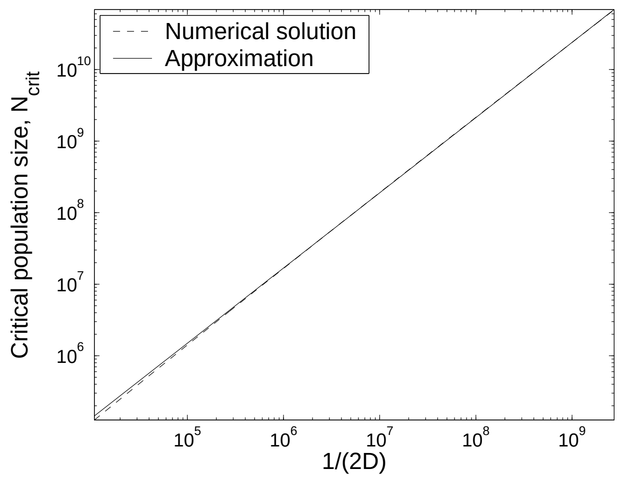


Figure 4.3: Critical population size with respect to the ratio $\frac{1}{2D}$ for BIC metric.

proportional to the growth of $\frac{1}{2D}$ and therefore to determine the growth of N_{crit} with respect to the size of the problem, it is sufficient to compute the growth of $\frac{1}{2D}$ with respect to the same parameter.

The following section starts by computing the probabilities of the partial solutions after applying binary tournament selection. These probabilities are then used to determine the growth of D in two cases: (1) the fitness contributions of X_1 and X_2 are correlated, and (2) the fitness contributions of X_1 and X_2 are independent. The two cases are distinguished by a parametrization of the fitness contribution of X_1 and X_2. The growth of D is then substituted to Equation 4.25 to determine the growth of N_{crit}.

4.3.5 Block Probabilities After Binary Tournament

The initial population is generated at random with the uniform distribution and therefore the probability of any instantiation of the variables in the considered block of k binary variables is given by

$$p_{init}(x) = \frac{1}{2^k}\cdot, \tag{4.26}$$

where $x = (x_1, \ldots, x_k)$ denotes the bits in the considered block.

binary tournament selection selects two parents at random and chooses the one with the better fitness. Denote the probability of a tournament between the competing blocks x and y in a particular order by p_{tourn} (note that x and y correspond to the same partition in the problem). Using

Equation 4.26,

$$p_{tourn} = p_{init}(X)p_{init}(Y) = \frac{1}{2^{2k}}. \tag{4.27}$$

The ordering of x and y in the tournament does not affect the result of the tournament and, therefore, the probability of x after binary tournament selection is given by

$$p(x) = \sum_{y_1,\ldots,y_k} 2p_{tourn}p(F(x) > F(y)), \tag{4.28}$$

where $F(x)$ and $F(y)$ denote the fitness distribution of the blocks x and y, respectively (see Equation 4.17); and $p(F(x) > F(y))$ denotes the probability that the solution with the block x wins the tournament over the solution with the block y. The probability of x winning the tournament over y can be rewritten as

$$p(F(x) > F(y)) = p(F(x) - F(y) > 0). \tag{4.29}$$

Since both $F(x)$ and $F(y)$ are normally distributed (see Equation 4.17), $F(x) - F(y)$ follows the normal distribution with the mean equal to the difference of the individual means of $F(x)$ and $F(y)$, and the variance equal to the sum of the variances of the two distributions. The difference of the mean fitness of $F(x)$ and $F(y)$ is equal to the difference of their contributions to the overall fitness denoted by $g(x)$ and $g(y)$, because the $F(x)$ and $F(y)$ are not correlated (due to the assumptions) and, consequently, the contributions of the remaining bits cancel out. Thus,

$$F(x) - F(y) \sim N(g(x) - g(y), 2\sigma_N^2). \tag{4.30}$$

That yields

$$p(F(x) > F(y)) = \Phi\left(\frac{g(x) - g(y)}{\sqrt{2}\sigma_N}\right), \tag{4.31}$$

where $\Phi(x)$ denotes the cumulative probability density function of the zero-mean normal distribution with the standard deviation of 1. The resulting probability of x after binary tournament selection is thus given by

$$p(x) = \sum_{y} \frac{1}{2^{2k-1}}\Phi\left(\frac{g(x) - g(y)}{\sqrt{2}\sigma_N}\right). \tag{4.32}$$

The ratio of the fitness differences to the deviation of collateral noise decreases with the problem size and is usually a very small number for moderate–to–large problems. Therefore, a linear approximation of the cumulative density function in the above equation can be used, where $\Phi(x) = \frac{1}{2} + \frac{x}{\sqrt{2\pi}}$, yielding

$$p(x) = \sum_{y_1,\ldots,y_k} \frac{1}{2^{2k-1}} \left(\frac{1}{2} + \frac{g(x) - g(y)}{2\sqrt{\pi}\sigma_N} \right). \tag{4.33}$$

In most cases we are interested only in the marginal probabilities of the instances x_1 and x_2 of the first two variables. These can be computed by marginalization:

$$\begin{aligned} p(x_1, x_2) &= \sum_{x_3,\ldots,x_k} p(x) \\ &= \sum_{x_3,\ldots,x_k} \sum_{y_1,\ldots,y_k} \frac{1}{2^{2k-1}} \left(\frac{1}{2} + \frac{g(x) - g(y)}{2\sqrt{\pi}\sigma_N} \right) \\ &= \frac{1}{2^{2k-1}} \left(2^k \sum_{x_3,\ldots,x_k} \frac{g(x)}{2\sqrt{\pi}\sigma_N} + 2^{k-2} \sum_{y_1,\ldots,y_k} \left(\frac{1}{2} - \frac{F(y)}{2\sqrt{\pi}\sigma_N} \right) \right) \\ &= \frac{1}{4} \left(1 + \frac{\bar{g}(x_1, x_2) - \bar{g}}{\sqrt{\pi}\sigma_N} \right), \end{aligned} \tag{4.34}$$

where $\bar{g}(x_1, x_2)$ denotes the average fitness contribution of the partition X with $X_1 = x_1$ and $X_2 = x_2$, and $\bar{g}$ denotes the average block fitness contribution of the partition X; that is,

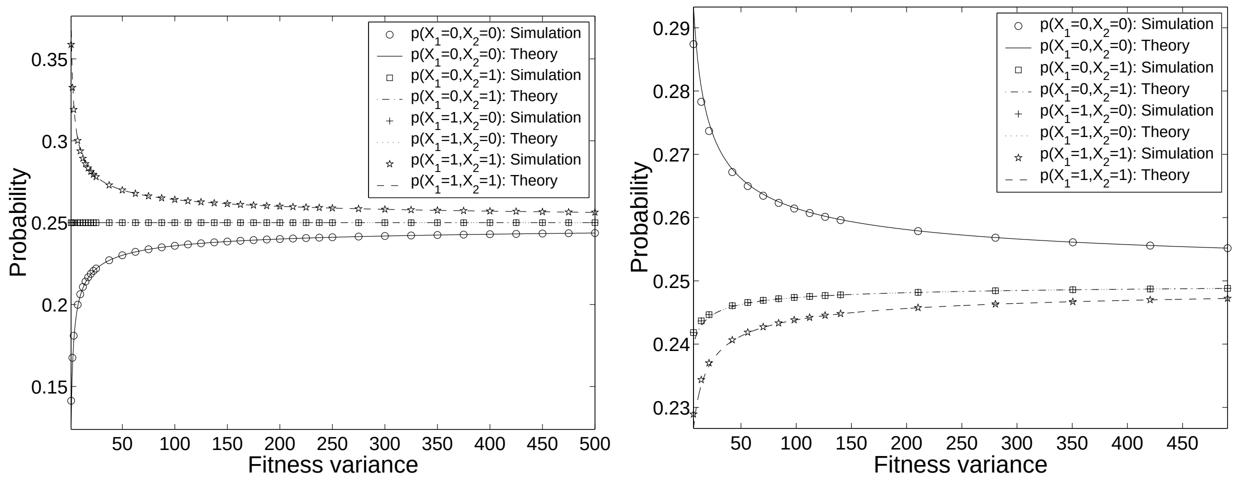
$$\bar{g}(x_1, x_2) = \frac{1}{2^{k-2}} \sum_{x_3,\ldots,x_k} g(x), \tag{4.35}$$

and

$$\bar{g} = \frac{1}{2^k} \sum_{x_1,\ldots,x_k} g(x). \tag{4.36}$$

The pairwise frequencies therefore depend only on the average block fitnesses of the corresponding two bits. The dynamics of the two bits in a partition of order k can be accurately approximated by a special case of a two-bit partition with the fitness defined according to the average building-block fitnesses $\bar{g}(x_1, x_2)$. This property simplifies the derivations and allows the analysis of pairwise statistical dependencies independently of the order of the partition in which the two variables are located.

The above approximations of the pairwise probabilities are verified in Figure 4.4. The values computed according to the above approximations are compared to the actual values computed



(a) Onemax.

(b) Trap of order 5.

Figure 4.4: Pairwise frequencies versus their approximation with respect to noise.

by the simulation of binary tournament selection using infinite populations on onemax and trap functions. The simulation directly encodes all the assumptions and computes the probabilities without any approximations. In both cases, the results of the simulation and the approximations are practically indistinguishable.

4.3.6 General Two-Bit Case

In the general two-bit case, the fitness of the two variables X_1 and X_2 can be written as

$$\bar{g}(X_1, X_2) = a_0 + a_1 X_1 + a_2 X_2 + a_{12} X_1 X_2, \tag{4.37}$$

where a_0, a_1, a_2, and a_{12} are constants. In the above equation, if the contribution of X_1 is independent of X_2, then $a_{12} = 0$. On the other hand, if $a_{12} \neq 0$, the contributions of the variables X_1 and X_2 are correlated.

The probability of any instantiation of the two variables X_1 and X_2 after binary tournament selection can be computed using Equation 4.34, yielding

$$
\begin{aligned}
p(X_1, X_2) &= \frac{1}{4}\left(1 + \frac{\bar{g}(X_1, X_2) - \bar{g}}{\sqrt{\pi}\sigma_N}\right) \\
&= \frac{1}{4}\left(1 + \frac{a_0 + a_1 X_1 + a_2 X_2 + a_{12} X_1 X_2 - \frac{4a_0 + 2a_1 + 2a_2 + a_{12}}{4}}{\sqrt{\pi}\sigma_N}\right) \\
&= \frac{1}{4}\left(1 + \frac{a_1(4X_1 - 2) + a_2(4X_2 - 2) + a_{12}(4X_1 X_2 - 1)}{4\sqrt{\pi}\sigma_N}\right)
\end{aligned}
\tag{4.38}
$$

By summing the above equations over X_1 and X_2, respectively, we get

$$
\begin{aligned}
p(X_1) &= \frac{1}{2}\left(1 + \frac{a_1(4X_1 - 2) + a_{12}(2X_1 - 1)}{4\sqrt{\pi}\sigma_N}\right) \\
p(X_2) &= \frac{1}{2}\left(1 + \frac{a_2(4X_2 - 2) + a_{12}(2X_2 - 1)}{4\sqrt{\pi}\sigma_N}\right)
\end{aligned}
\tag{4.39}
$$

The above equations can be used to compute the frequencies of any instantiation of X_1 and X_2, yielding the following set of equations:

$$
\begin{aligned}
p(X_1 = 0, X_2 = 0) &= \frac{1}{4}\left(1 + \frac{-2a_1 - 2a_2 - a_{12}}{4\sqrt{\pi}\sigma_N}\right) = \frac{1}{4}(1 + \chi_{00}) \\
p(X_1 = 0, X_2 = 1) &= \frac{1}{4}\left(1 + \frac{-2a_1 + 2a_2 - a_{12}}{4\sqrt{\pi}\sigma_N}\right) = \frac{1}{4}(1 + \chi_{01}) \\
p(X_1 = 1, X_2 = 0) &= \frac{1}{4}\left(1 + \frac{2a_1 - 2a_2 - a_{12}}{4\sqrt{\pi}\sigma_N}\right) = \frac{1}{4}(1 + \chi_{10}) \\
p(X_1 = 1, X_2 = 1) &= \frac{1}{4}\left(1 + \frac{2a_1 + 2a_2 + 3a_{12}}{4\sqrt{\pi}\sigma_N}\right) = \frac{1}{4}(1 + \chi_{11})
\end{aligned}
\tag{4.40}
$$

In the last set of equations, the parameters χ_{ij} are defined to be equal to the terms they replace. Additionally, the probabilities of the instantiations of X_1 and X_2 can be computed as follows:

$$
\begin{aligned}
p(X_1 = 0) &= \frac{1}{2}\left(1 + \frac{-2a_1 - a_{12}}{4\sqrt{\pi}\sigma_N^2}\right) = \frac{1}{2}(1 - \chi_1) \\
p(X_1 = 1) &= \frac{1}{2}\left(1 + \frac{2a_1 + a_{12}}{4\sqrt{\pi}\sigma_N^2}\right) = \frac{1}{2}(1 + \chi_1) \\
p(X_2 = 0) &= \frac{1}{2}\left(1 + \frac{-2a_2 - a_{12}}{4\sqrt{\pi}\sigma_N^2}\right) = \frac{1}{2}(1 - \chi_2) \\
p(X_2 = 1) &= \frac{1}{2}\left(1 + \frac{2a_2 + a_{12}}{4\sqrt{\pi}\sigma_N^2}\right) = \frac{1}{2}(1 + \chi_2)
\end{aligned}
\tag{4.41}
$$

Again, the parameters χ_i are defined to be equal to the terms they replace.

Next, we compute the order of the growth of D in two separate cases. The first case considers two nonlinearly interacting variables where $a_{12} > 0$. The second case considers two variables whose contributions are independent and thus $a_{12} = 0$. In both the cases, the marginal entropies $H(X_1)$, $H(X_2)$, and $H(X_1, X_2)$ are first computed. These are then used to compute D, which can be expressed in terms of the marginal entropies as

$$D = H(X_1) - H(X_1|X_2) = H(X_1) + H(X_2) - H(X_1, X_2). \tag{4.42}$$

Dependent Case: $a_{12} > 0$

First, let us compute an approximation of the entropy of X_1 defined as

$$H(X_1) = -\sum_{x_1} p(x_1) \log_2 p(x_1). \tag{4.43}$$

Using the set of equations 4.41, the entropy of X_1 can be computed as

$$\begin{aligned} H(X_1) &= -\frac{1}{2}(1-\chi_1)\log_2 \frac{1}{2}(1-\chi_1) - \frac{1}{2}(1+\chi_1)\log_2 \frac{1}{2}(1+\chi_1) \\ &= -\frac{1}{2}\left(\log_2(1-\chi_1^2) + \chi_1(\log_2(1+\chi_1) - \log_2(1-\chi_1))\right) + 1 \end{aligned} \tag{4.44}$$

Since χ_1 is very small for moderate–to–large sized problems (it approaches zero as σ_N approaches infinity), we can use a linear approximation of the logarithm near 1. In particular,

$$\begin{aligned} \log_2(1-\chi_1^2) &\approx -\frac{\chi_1^2}{\ln 2}, \\ \chi_1 \log_2(1+\chi_1) &\approx \frac{\chi_1^2}{\ln 2}, \\ \chi_1 \log_2(1-\chi_1) &\approx -\frac{\chi_1^2}{\ln 2}. \end{aligned} \tag{4.45}$$

Thus,

$$\begin{aligned} H(X_1) &= -\frac{\chi_1^2}{2\ln 2} + 1 \\ &= -\frac{4a_1^2 + 4a_1 a_{12} + a_{12}^2}{32\pi\sigma_N^2 \ln 2} + 1. \end{aligned} \tag{4.46}$$

$H(X_2)$ can be computed analogously to $H(X_1)$, yielding

$$H(X_2) = -\frac{4a_2^2 + 4a_2 a_{12} + a_{12}^2}{32\pi\sigma_N^2 \ln 2} + 1.$$

The joint entropy $H(X_1, X_2)$ is given by

$$H(X_1, X_2) = -\sum_{x_1, x_2} p(x_1, x_2) \log_2 p(x_1, x_2) = -(A_{00} + A_{01} + A_{10} + A_{11}),$$

where

$$A_{ij} = p(X_1 = i, X_2 = j) \log_2 p(X_1 = i, X_2 = j). \tag{4.47}$$

The terms A_{ij} can be approximated as follows:

$$\begin{aligned} A_{ij} &= \frac{1}{4}\left((1 + \chi_{ij}) \log_2 \frac{1}{4}(1 + \chi_{ij})\right) \\ &= \frac{1}{4}\left(\log_2(1 + \chi_{ij}) + \chi_{ij} \log_2(1 + \chi_{ij}) - 2(1 + \chi_{ij})\right) \end{aligned} \tag{4.48}$$

Since χ_{ij} is very small, the last equation can be simplified using the following approximations:

$$\begin{aligned} \log_2(1 + \chi_{ij}) &\approx \frac{2\chi_{ij} - \chi_{ij}^2}{2 \ln 2}, \\ \chi_{ij} \log_2(1 + \chi_{ij}) &\approx \frac{\chi_{ij}^2}{\ln 2}. \end{aligned} \tag{4.49}$$

Thus,

$$A_{ij} = \frac{1}{4}\left(\frac{2\chi_{ij} + \chi_{ij}^2}{2 \ln 2} - 2(1 + \chi_{ij})\right). \tag{4.50}$$

By substituting the approximations of A_{ij} and the equations for χ_{ij}, we get

$$H(X_1, X_2) = -\frac{1}{8}\left(\frac{16a_1^2 + 16a_2^2 + 12a_{12}^2 + 16a_1 a_{12} + 16a_2 a_{12}}{16\pi\sigma_N^2 \ln 2}\right) + 2 \tag{4.51}$$

Thus, the difference D between the marginal and conditional entropies can be approximated by

$$D = \frac{a_{12}^2}{32\pi\sigma_N^2 \ln 2}. \tag{4.52}$$

Therefore, if the fitness contributions of X_1 and X_2 are not independent, then D grows inversely proportionally to the variance of collateral noise.

By substituting Equation 4.52 into Equation 4.25, we can infer that

$$N_{crit} = O(\sigma_N^{2.1}). \tag{4.53}$$

Using the assumption that $\sigma_N^2 \propto n$ where n is the number of variables in the problem, we can imply that

$$N_{crit} = O(n^{1.05}). \tag{4.54}$$

In other words, the critical population size for discovering a dependency between the two variables that are nonlinearly correlated grows approximately linearly with the size of the problem.

Independent Case: $a_{12} = 0$

Note that in this case, $a_{12} = 0$. Therefore,

$$\chi_1 = \frac{a_1}{2\sqrt{\pi}\sigma_N}, \quad \chi_2 = \frac{a_2}{2\sqrt{\pi}\sigma_N}. \tag{4.55}$$

We can now write χ_2 in terms of χ_1 as

$$\chi_2 = \frac{a_2}{a_1}\chi_1 = b\chi_1, \tag{4.56}$$

where $b = a_2/a_1$. The entropy of X_i (here we're interested in X_1 and X_2 only) can be written as

$$\begin{aligned} H(X_i) &= -\frac{1}{2}\left((1+\chi_i)\log_2\left(\frac{1+\chi_i}{2}\right) + (1-\chi_i)\log_2\left(\frac{1-\chi_i}{2}\right)\right) \\ &= -\frac{1}{2}\left((1+\chi_i)\log_2(1+\chi_i) + (1-\chi_i)\log_2(1-\chi_i) - 2\right) \\ &= -\frac{1}{2}\left(\log_2((1+\chi_i)(1-\chi_i)) + \chi_i\log_2\left(\frac{1+\chi_i}{1-\chi_i}\right) - 2\right). \end{aligned} \tag{4.57}$$

Since χ_i is very small, the logarithms in the last equation can be approximated as

$$\log_2\left((1+\chi_i)(1-\chi_i)\right) \approx -\frac{2\chi_i^2+\chi_i^4}{2\ln 2}, \tag{4.58}$$

$$\log_2\left(\frac{1+\chi_i}{1-\chi_i}\right) \approx \frac{6\chi_i+2\chi_i^3}{3\ln 2}. \tag{4.59}$$

Using the above approximations, the entropy $H(X_i)$ is given by

$$H(X_i) = -\frac{1}{2}\left(\frac{6\chi_i^2+\chi_i^4}{6\ln 2} - 2\right). \tag{4.60}$$

Thus, the entropies $H(X_1)$ and $H(X_2)$ are given by

$$H(X_1) = -\frac{6\chi_1^2+\chi_1^4}{12\ln 2} + 1. \tag{4.61}$$

$$\begin{aligned} H(X_2) &= -\frac{6\chi_2^2+\chi_2^4}{12\ln 2} + 1, \\ &= -\frac{6b^2\chi_1^2+b^4\chi_1^4}{12\ln 2} + 1. \end{aligned} \tag{4.62}$$

The only other term remaining to compute the entropy difference D is the joint entropy, $H(X_1, X_2)$, which is given by

$$H(X_1, X_2) = -\left(A_{00} + A_{01} + A_{10} + A_{11}\right),$$

where

$$\begin{aligned} \chi_{00} &= -(1+b)\chi_1, \\ \chi_{01} &= -(1-b)\chi_1, \\ \chi_{10} &= (1-b)\chi_1, \\ \chi_{11} &= (1+b)\chi_1, \end{aligned} \tag{4.64}$$

and

$$\begin{aligned} A_{ij} &= \frac{1}{4}\left((1+\chi_{ij})\log_2\left(\frac{1+\chi_{ij}}{4}\right)\right) \\ &= \frac{1}{4}\left(\log_2\left(1+\chi_{ij}\right) + \chi_{ij}\log_2\left(1+\chi_{ij}\right) - 2\left(1+\chi_{ij}\right)\right). \end{aligned} \tag{4.65}$$

From the above equation, we get

$$
\begin{aligned}
A_{00} &= \frac{1}{4}\left(\log_2\left(1-(1+b)\chi_1\right)-(1+b)\chi_1\log_2\left(1-(1+b)\chi_1\right)-2\left(1-(1+b)\chi_1\right)\right),\\
A_{01} &= \frac{1}{4}\left(\log_2\left(1-(1-b)\chi_1\right)-(1-b)\chi_1\log_2\left(1-(1-b)\chi_1\right)-2\left(1-(1-b)\chi_1\right)\right),\\
A_{10} &= \frac{1}{4}\left(\log_2\left(1+(1-b)\chi_1\right)+(1-b)\chi_1\log_2\left(1+(1-b)\chi_1\right)-2\left(1+(1-b)\chi_1\right)\right),\\
A_{11} &= \frac{1}{4}\left(\log_2\left(1+(1+b)\chi_1\right)+(1+b)\chi_1\log_2\left(1+(1+b)\chi_1\right)-2\left(1+(1+b)\chi_1\right)\right).
\end{aligned}
$$

Summing the above equations for A_{ij} gives us $H(X_1, X_2)$:

$$
\begin{aligned}
H(X_1,X_2) &= -\frac{1}{4}\left(\log_2\left((1-(1+b)^2\chi_1^2)(1-(1-b)^2\chi_1^2)\right)+(1+b)\chi_1\log_2\left(\frac{1+(1+b)\chi_1}{1-(1+b)\chi_1}\right)\right.\\
&\quad \left.+(1-b)\chi_1\log_2\left(\frac{1+(1-b)\chi_1}{1-(1-b)\chi_1}\right)-8\right). \qquad (4.66)
\end{aligned}
$$

Using the approximations from equations 4.58, and 4.59,

$$
\begin{aligned}
H(X_1,X_2) &= -\frac{1}{4\ln 2}\left((1+b)^2\chi_1^2+\frac{1}{6}(1+b)^4\chi_1^4+(1-b)^2\chi_1^2+\frac{1}{6}(1-b)^4\chi_1^4\right)+2,\\
&= -\frac{1}{4\ln 2}\left(\left((1+b)^2+(1-b)^2\right)\chi_1^2+\frac{1}{6}\left((1+b)^4+(1-b)^4\right)\chi_1^4\right)+2,\\
&= -\frac{1}{2\ln 2}\left(1+b^2\right)\chi^2-\frac{1}{12\ln 2}\left(1+6b^2+b^4\right)\chi_1^4+2. \qquad (4.67)
\end{aligned}
$$

The entropy difference D can be computed by substituting equations 4.61, 4.62, and 4.67 into Equation 4.42:

$$
\begin{aligned}
D &= \frac{1}{2\ln 2}\left(1+b^2\right)\chi^2+\frac{1}{12\ln 2}\left(1+6b^2+b^4\right)\chi^4-\frac{1}{2\ln 2}\left(1+b^2\right)\chi^2-\frac{1}{12\ln 2}\left(1+b^4\right)\chi^4,\\
&= \frac{b^2}{2\ln 2}\chi_1^4. \qquad (4.68)
\end{aligned}
$$

Recall that $\chi_1 = \frac{a_1}{2\sqrt{\pi}\sigma_N}$, and $b = a_2/a_1$. Therefore, the above equation can be written in terms of the variance of collateral noise σ_N^2 as

$$
D = \frac{1}{32\ln 2}\left(\frac{a_1 a_2}{\pi}\right)^2\sigma_N^{-4}. \qquad (4.69)
$$

By substituting Equation 4.69 into Equation 4.25, we can imply that if $a_{12} = 0$, then

$$N_{crit} = O(\sigma_N^{4.2}), \tag{4.70}$$

which yields

$$N_{crit} = O(n^{2.1}). \tag{4.71}$$

In other words, the population size to discover the dependency between the two variables that are independent with respect to the fitness function grows approximately quadratically with the problem size.

The above theory assumes that X_1 has no parents before the decision is made on whether the edge $X_2 \rightarrow X_1$ should be added into the network (see Figure 4.2(a)). The following section discusses an extension of the theory to the general case where X_1 already has a number of parents before the decision regarding the edge $X_2 \rightarrow X_1$ is made (see Figure 4.2(b)). Subsequently, the section justifies the assumption that the frequencies follow their expected behavior by incorporating the effects of a finite population sizing into the model.

4.3.7 General Case: Multiple Parents of X_1 Exist

Above we computed the required population size for the addition of the first edge into X_1. How does the situation change if some edges that end in X_1 are already present in the current model? This section indicates that even in that case, the overall growth of the population size does not change much, although the population size must grow exponentially with the order of the considered dependencies.

The condition for adding the edge $X_2 \rightarrow X_1$ into the network that already contains the edges $X_3 \rightarrow X_1$ to $X_k \rightarrow X_1$ is given by

$$BIC(X_1|X_2,\ldots,X_k) > BIC(X_1|X_3,\ldots X_k). \tag{4.72}$$

Using the definition of BIC, the last equation can be rewritten as

$$-H(X_1|X_2,\ldots,X_k)N - 2^{k-2}\log_2 N > -H(X_1|X_3,\ldots,X_k) - 2^{k-3}\log_2 N. \tag{4.73}$$

Denoting $D = H(X_1|X_3 \dots X_k) - H(X_1|X_2 \dots X_k)$ yields

$$N - \frac{\log_2 N}{2^{3-k} D} > 0. \tag{4.74}$$

Analogously to the case of the first-edge addition, the critical population size is the larger of the two solutions of the above equation and the growth of D determines the growth of the critical population size.

To determine the growth of D, let us first discuss the form of the nonlinearities that should be discovered in this case. If the contribution of X_1 does not depend on X_2 given the values of X_3 to X_k, then the edge $X_2 \to X_1$ is not required, because there is no additional nonlinearity that must be covered in the model. However, if there is some combination of values of X_3 to X_k for which the contributions of X_1 and X_2 are correlated, the edge $X_2 \to X_1$ should be added to reflect the nonlinearity.

The discussion in the above paragraph suggests that the nonlinearities that are *conditioned* on the particular values of X_3 to X_k are important to cover. In that case, the growth of D can be approximated by partitioning the population according to the instantiations of X_3 to X_k, and looking at each subpopulation separately. Note that the size of each partition of the population is approximately $N/2^{k-2}$, because the probabilities of the blocks of $k-2$ bits get closer to each other asymptotically. Figure 4.5 shows an example partitioning of the population according to the first 3 bits.

Using the subpopulations of the partitioning according to $(X_3, \dots, X_k)$, the overall D can be computed as the weighted sum of D's for each subpopulation, because

$$H(X_1|X_3, \dots, X_k) = \sum_{x_3,\dots,x_k} p(x_3, \dots, x_k) H_{x_3,\dots,x_k}(X_1), \tag{4.75}$$

and

$$H(X_1|X_2, X_3, \dots, X_k) = \sum_{x_3,\dots,x_k} p(x_3, \dots, x_k) H_{x_3,\dots,x_k}(X_1|X_2), \tag{4.76}$$

where $H_{x_3,\dots,x_k}(X)$ denotes the entropy of X in the partition of $(x_3, \dots, x_k)$. Consequently, if the fitness contributions of X_1 and X_2 are correlated in at least one partition, the dominant term in

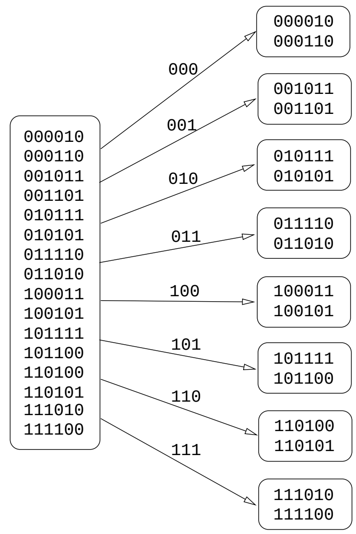


Figure 4.5: Partitioning the population according to the first three bits. For each configuration of the first three bits, a new population is created that consists of all the individuals that contain that configuration.

D will be the one coming from that partition, and the dependency will be found if the size of that partition grows as $O(n^{1.05})$. Since the size of each partition is approximately $N/2^{k-2}$, the overall growth of the population size can be bounded by $O(2^{k-2}n^{1.05})$. If the contributions of X_1 and X_2 are correlated in more than one partition, the population-sizing bound can be decreased accordingly. On the other hand, if the contributions of X_1 and X_2 are independent in every context of X_3 to X_k, the population size to discover this unnecessary dependency $X_2 \rightarrow X_1$ should grow as $O(2^{k-2}n^{2.1})$.

Therefore, the theory for the case of adding the first edge that ends in X_1 can be extended to the general case in a straightforward manner, yielding the overall bound on the population size of

$$N_{crit} = O(2^k n^{1.05}), \tag{4.77}$$

where k is the maximum order of the subproblems in the problem decomposition. The above result has two important implications:

(1) **Sufficient population size.** The sufficient population size for discovering the nonlinearities in the problem and encoding them in the learned Bayesian network grows approximately with $O(2^k n^{1.05})$. Assuming the fixed order of decomposition, the growth is $O(n^{1.05})$.

(2) **Favored nonlinearities.** The magnitude of the nonlinearities affects the population sizing. The higher the magnitude of the nonlinearities, the smaller the population size. Assuming a particular population size, only the strongest nonlinearities can be covered, because the population size grows exponentially with the order of the covered dependencies.

Both the dependent case and the independent one assume that the frequencies after selection are equal to their expected values. The following section analyzes the effects of using finite populations on the accuracy of the actual frequencies and incorporates the results of the analysis into the developed model.

4.3.8 Getting the Frequencies Right

The population-sizing model presented above assumes that the frequencies after binary tournament selection follow their expected behavior. However, in practice we can only use finite populations and the model should also consider the effects of the finite population sizing on the actual, observed, frequencies and, consequently, on the model building in BOA. The purpose of this section is to analyze the effects of the finite population sizing and apply the results of the analysis to the presented population-sizing model. In particular, a lower bound on the population size is computed that ensures that the actual frequencies of each block of k bits are close enough to their expected values with high confidence, where "close enough" will be defined later.

The bound is computed in two steps; the first step assumes that the probability of x winning a random tournament is equal to its expected value, the second step describes how the assumptions of the first step can be satisfied.

Assume that the probability of x being a winner of one tournament is equal to its expected probability $p(x)$ after selection. Let us denote the actual probability (relative frequency) of x in the selected set of solutions after performing m tournaments by $p_m(X)$. Note that after m tournaments there are m solutions selected (m is the size of the selected population) and, since the tournaments are stochastic, $p_m(X)$ is a random variable.

The distribution of $p_m(X)$ is binomial, because $p_m(X)$ is equal the number of successes in m independent trials divided by the number of trials, each trial with the probability of success equal

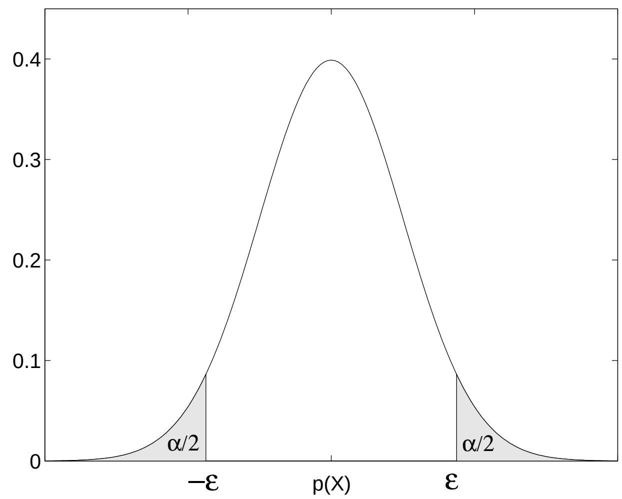


Figure 4.6: To ensure that $p_m(X)$ is within ϵ of its mean, the areas under the tails that start ϵ-away from the mean must sum to α. The property can be satisfied for large enough m.

to $p(x)$. The mean of $p_m(x)$ is $p(x)$ and the variance of $p_m(x)$ is $p(x)(1-p(x))/m$:

$$p_m(x) \sim Bin\left(p(x), \frac{p(x)\,(1-p(x))}{m}\right). \tag{4.78}$$

For moderate values of m the binomial distribution can be approximated by the normal distribution, yielding

$$p_m(X) \sim N\left(p(X), \frac{p(X)\,(1-p(X))}{m}\right). \tag{4.79}$$

With confidence α the actual frequency $p_m(X)$ is within ϵ from its expected value $p(X)$, if

$$\Phi\left(-\frac{\epsilon\sqrt{m}}{\sqrt{p(X)(1-p(X))}}\right) \leq \frac{\alpha}{2}, \tag{4.80}$$

where Φ is the cumulative density of the unit normal distribution (see Figure 4.6). For m, we get

$$m \geq \left(\frac{\Phi^{-1}\left(\frac{\alpha}{2}\right)\sqrt{p(X)(1-p(X))}}{\epsilon}\right)^2, \tag{4.81}$$

where Φ^{-1} is the inverse cumulative density of the unit normal distribution. The number of tournaments must therefore grow inversely proportionally with the square of ϵ.

How should we set the value of ϵ for the population sizing in BOA? Let us first get back to the block probabilities and their dynamics with the problem size. The frequencies of any two-bit partition approach 0.25 inversely proportionally to the standard deviation σ_N of collateral noise. It would be therefore reasonable to set the error ϵ to decrease at the same rate so that the same relative accuracy could be achieved for the entire spectrum of problem sizes. For instance, the distance of the frequencies to their asymptotic value could deviate by at most 1% independently of the size of the problem. In that case, the actual population size could be bounded by the two extreme cases at an arbitrary level of confidence.

The frequencies for bigger partitions behave similarly, but they are scaled down by an additional factor of 2^{k-2} where k is the order of the considered partition (see Equation 4.32). So the accuracy of the frequencies should also increase proportionally to 2^k. Therefore, it is reasonable to require that

$$\epsilon \propto \frac{1}{2^k \sigma_N}, \tag{4.82}$$

where k is the size of the building blocks we must consider to find the optimum. Using the assumption that $\sigma_N^2 \propto n$ (see Equation 4.18), we get

$$\epsilon \propto \frac{1}{2^k \sqrt{n}}, \tag{4.83}$$

where n is the size of the problem. Substituting the last equation into Equation 4.81 yields

$$m = O(2^k n). \tag{4.84}$$

Above we assumed that the expected probability of x winning a random tournament was equal to its expected value. How can this assumption be justified? There are two approaches. The first approach is to look at the moments of the fitness distribution in the initial population and require that the actual observed moments are close to their expected values; this approach was studied in Pelikan, Goldberg, and Cantú-Paz (2000a). However, it is much simpler to use the basic properties of tournament selection to simplify the computation; this approach is discussed next.

Note that if the candidate solutions participating in each tournament were generated at random, the actual probability of x winning a random tournament would indeed be equal to its expected value. The problem is that some solutions must be *reused* to perform N tournaments of size 2 using the population of size N. To eliminate this problem, the initial population can be made twice as large as the required size of the selected population,

$$N \geq 2m, \tag{4.85}$$

where m is the lower bound on the size of the selected population.

Assuming that $N \geq 2m$, any set of m solutions can be selected with confidence α and error at most ϵ. Of course, we need to select N solutions. Then, two sets of size m must be selected. After selecting the two sets, the error does not change (because the probabilities are averaged over the two sets), but the confidence of the model changes to $(2\alpha - \alpha^2)$. Therefore, to achieve a specified confidence level β, the alpha must first be determined such that

$$2\alpha - \alpha^2 \leq \beta \tag{4.86}$$

In any case, the size of the population increases at most by a factor equal to the size of the tournaments and a constant factor related to the confidence (for a fixed confidence level). Since size of the tournaments is fixed, the bound on the population size remains of the following form:

$$N = O(2^k n). \tag{4.87}$$

Therefore, for a constant bound k on the order of the subproblems, the population size to ensure that the frequencies retain the same relative error with arbitrary confidence grows linearly with the problem size. Since the growth of the population sizes in both the dependent case and the independent one was at least linear as well, the population-sizing model for the BIC metric is applicable to the case of finite populations.

The following section presents empirical results that verify the model and the approximations made in its derivation.

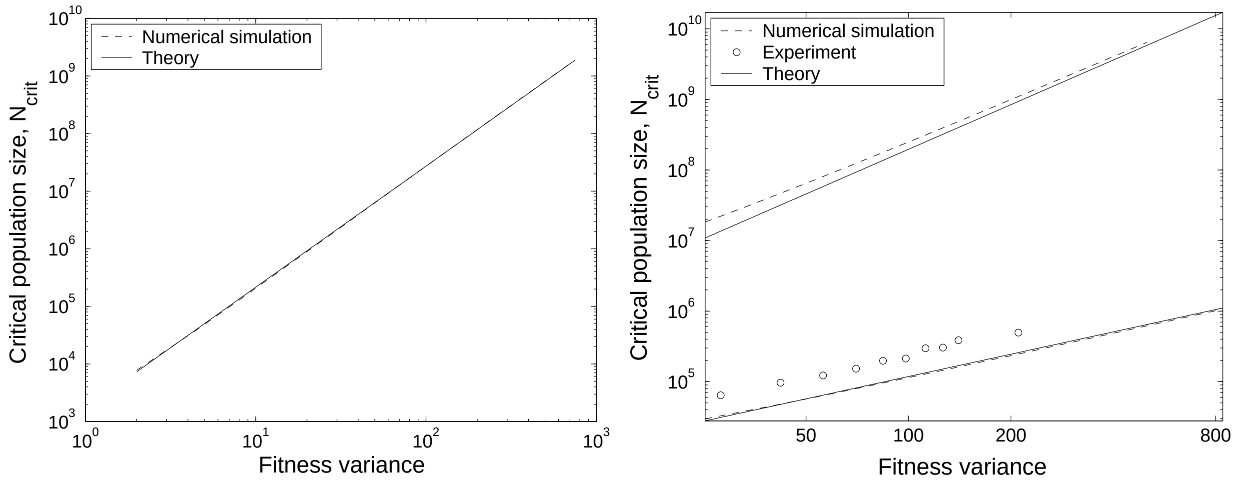
4.3.9 Critical Population Size: Empirical Results

Figure 4.7(a) shows the critical population size for onemax fitness function defined in Equation 1.1 on page 13. Both the simulation for infinite populations (based on the exact theoretical model for the frequencies using an infinite population) as well as the final approximate result are shown. We can see that the match between the theory and the infinite-population simulation is very good and that the critical population size for discovering a dependency between the variables whose fitness contributions are independent increases approximately quadratically with the fitness variance, which is proportional to the size of the problem.

Figure 4.7(b) shows the critical population size for the trap function of order 5 compared to the empirical results for the simulation with finite and infinite populations. The correlated bits are both selected from one of the trap subfunctions, while the independent bits are selected from two different subfunctions. The infinite-population simulation was again performed according to the assumptions stated in Section 4.3.3. The simulation for a finite population was performed by simulating the actual binary tournament selection on a finite population and increasing the population size until the probability of discovering the dependency was more than 95% in 100 independent runs. The exact theoretical results and the approximations match very well. We can also see that the use of a finite population introduces additional noise that increases the population-sizing requirements for a reliable detection of the correct dependencies, but that the growth of the appropriate population size is still approximately linear.

The results in Figure 4.7(b) also indicate that there is a large range of population sizes that result in a reliable discovery of nonlinear dependencies but still do not introduce unnecessary dependencies between the variables whose fitness contributions are correlated. Moreover, the range grows with the problem size.

Our theoretical analysis considered only the *binary* tournament selection. Figure 4.8(a) indicates that the range of population sizes leading to the discovery of good dependencies but ensuring that the algorithm is not misled by the bad dependencies grows with the selection pressure. The bad news is that the growth of the required population sizes grows slightly faster with an increased selection pressure. For the tournament size of $s = 2$, the actual growth of the population size is approximately $O(n^{1.035})$. For the tournament size of $s = 16$, the growth increases to $O(n^{1.242})$.



(a) Onemax.

(b) Trap of order 5.

Figure 4.7: Critical population size for onemax and the composed trap of order 5 for BIC metric.

On the other hand, the order of the growth of the population size required to discover the bad dependencies decreases from 1.974 for $s = 2$ to 1.572 for $s = 16$. Nonetheless, as mentioned above, the range of adequate population sizes still increases with the selection pressure.

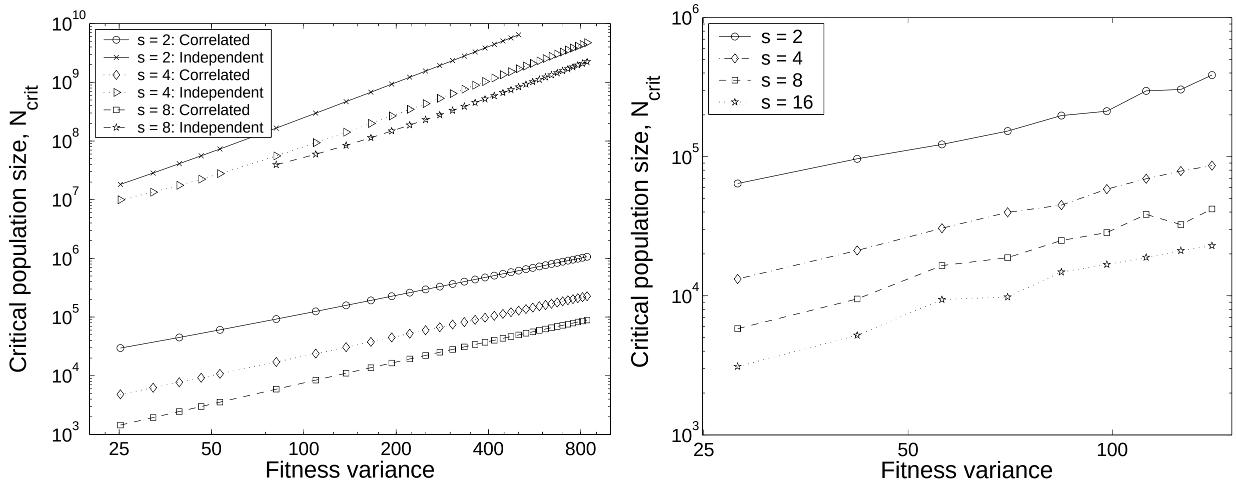
Figure 4.8(b) shows that increasing the tournament size up to $s = 16$ decreases the critical population size even for the case of a finite population. However, for high selection pressures, the positive effects of increasing the selection pressure can be expected to decrease and actually harm the performance of the algorithm in practice due to the premature convergence.

We observed similar results regarding the discovery of dependencies of higher order for both the onemax and trap function.

4.3.10 Summary of Population Sizing in BOA

This section summarizes the results of the aforementioned population-sizing models for GAs and BOA.

The requirements on the population size in BOA due to the four factors of the BOA population sizing (see Section 4.3) are listed in Table 4.2. For uniformly scaled subproblems and a bounded size of the subproblems, the population-sizing requirements for the BIC metric to distinguish between



(a) Simulation with infinite population.

(b) Experiment with finite population.

Figure 4.8: The effects of increasing the selection pressure on the critical population size. As the selection pressure increases, the critical population size decreases. The reason for this behavior is that increasing the selection pressure results in increasing the effect of each nonlinearity on the frequencies after selection.

the good and the bad dependencies, are dominant. The population size must grow as

$$N_{uni} = O(n^{1.05}). \tag{4.88}$$

For exponentially scaled problems, the model building is easy because the entire population is used to determine the model over only a small subset of the variables that are being optimized at the time. The effects of genetic drift become the dominant factor, yielding

$$N_{exp} = O(n). \tag{4.89}$$

The following section provides background of GA theory for estimating the number of generations until convergence. Section 4.5 relates the GA time-to-convergence theory to BOA. Subsequently, Section 4.6 combines the BOA population-sizing theory with the estimated number of generations until convergence to compute the total number of evaluations.

Factor	Population size
Initial supply	$O(2^k)$
Decision making	$O(2^k\sqrt{n})$
Genetic drift	$O(\sqrt{n})$ to $O(n)$
Model building	$O(2^k n^{1.05})$ for putting in the right edges, or $O(2^k n)$ for getting the frequencies right.

Table 4.2: An overview of the factors influencing the population sizing in BOA and the corresponding bounds. The first three factors are adopted from GA population sizing theory. The last factor is introduced by the need for building a proper model in BOA.

4.4 Background of GA Time-to-Convergence Theory

Once we have an adequate population size to find a solution of a specified quality, another important question is how many generations it will take the algorithm to converge. As discussed above, it is important to distinguish between the problems with uniform and exponential scaling. The number of generations until convergence in the exponential case was discussed above in context of the drift population-sizing model and the interested reader should refer to the work cited in the previous section. This section focuses on the uniformly scaled subproblems.

Mühlenbein and Schlierkamp-Voosen (1993) estimated the number of generations until the convergence on onemax, assuming an infinite population size and perfect mixing by population-wise uniform crossover, as follows:

$$t_{conv} = \left(\frac{\pi}{2} - \arcsin\left(2p-1\right)\right)\frac{\sqrt{n}}{I}, \tag{4.90}$$

where p is the initial proportion of ones on each position; n is the problem size; and I is the selection intensity. The selection intensity is defined as the difference between the mean fitness of the population before selection and the population after selection normalized by the standard deviation of the fitness values in the population before selection:

$$I(t) = \frac{\overline{f}(t+1) - \overline{f}(t)}{\sigma(t)}, \tag{4.91}$$

where $\overline{f}(t+1)$ is the average fitness of the population in generation $t+1$; $\overline{f}(t)$ is the average fitness in generation t; and $\sigma(t)$ is the standard deviation of the fitness values in generation t. For many commonly used selection schemes—such as the tournament, truncation, ranking, and (λ, μ)

selection—the selection intensity is a constant (Mühlenbein & Schlierkamp-Voosen, 1993).

The above result suggests that with a large enough population, the number of generations until convergence is proportional to the square root of the problem size and inversely proportional to the selection intensity. The convergence model was later rederived and extended by Miller and Goldberg (1996a) to take into account additional normally distributed noise in the fitness function.

The above two convergence models assume that the population is large enough for the convergence to follow its expected behavior with an infinite population size. In practice, the population size must usually be sufficiently large for the convergence to the optimum; in that case the GA dynamics is indeed close to the case with infinite populations. However, the dynamics changes if the populations are much smaller, which is often the case in parallel GA implementations that distribute the population on several processors and evolve each subpopulation in isolation. Ceroni, Pelikan, and Goldberg (2001) incorporated the effects of small populations on the overall number of generations until convergence. The primary use of the convergence model of Ceroni et al. (2001) is in maximizing the speed-up of parallel GAs.

The following section considers the number of generations until convergence in BOA.

4.5 Time to Convergence in BOA

The last piece we must collect to complete the puzzle of BOA scalability is the time to convergence. The following two sections compute the number of generations until BOA convergence in two extreme cases: (1) uniformly scaled subproblems, and (2) exponentially scaled subproblems. In the first case, the fitness contributions of all the subproblems of the decomposition are scaled the same and all the partitions converge in parallel. In the second case, the subproblems can be ordered so that the fitness contributions of any subproblem overshadow the contributions of the remaining subproblems in the ordering; in this case, the partitions corresponding to the different subproblems converge sequentially, one partition after another, in a domino-like fashion.

4.5.1 Uniform Scaling

The number of generations until the convergence of BOA can be modeled analogously to onemax and population-wise uniform crossover. In that case, Mühlenbein and Schlierkamp-Voosen (1993) showed that the number of generations until convergence grows as

$$G_{onemax} = \left(\frac{\pi}{2} - \arcsin\left(2p - 1\right)\right)\frac{\sqrt{n}}{I}, \tag{4.92}$$

where p is the proportion of ones on each position in the initial generation, n is the problem size, and I is the selection intensity (see Equation 4.91). For most commonly used selection methods—such as tournament and truncation selection—the selection intensity is constant and the number of generations is therefore bounded by

$$G_{onemax} = O(\sqrt{n}). \tag{4.93}$$

Although the exact approximation of G given in Equation 4.92 is correct only for a simple model with no interactions applied to the onemax case, the model can be used to accurately model the convergence time of BOA on many other decomposable problems where the order of each subproblem is bounded by a constant and the contributions of all the subproblems are scaled the same (see Miller and Goldberg (1996a) and Pelikan, Goldberg, and Cantú-Paz (2000a)). When the dynamics of the fitness variance is similar to the onemax case, Equation 4.92 approximates the time to convergence very well. Even if this is not the case, the time to convergence can still be accurately approximated by fitting G according to Equation 4.93. The reason for that behavior is that the time convergence can be upper-bounded by the the number of generations it would take to converge if the initial population contained only two partial solutions in each partition—the building block and its toughest competitor. In this case, the assumptions of the onemax convergence model hold if a proper probabilistic model is used, and the number of generations can be computed using Equation 4.92.

Figure 4.9 (from Pelikan, Goldberg, and Cantú-Paz (2000a)) shows the number of generations until convergence of BOA with truncation selection with $s = 2$, which selects the best half of the population in each generation. The empirical results on the trap and onemax fitness functions are compared to the prediction according to Equation 4.92. In both cases, the results match the original approximation for onemax with the model that contains no interactions.

The above theory assumes that the population is sufficiently large. To incorporate the effects

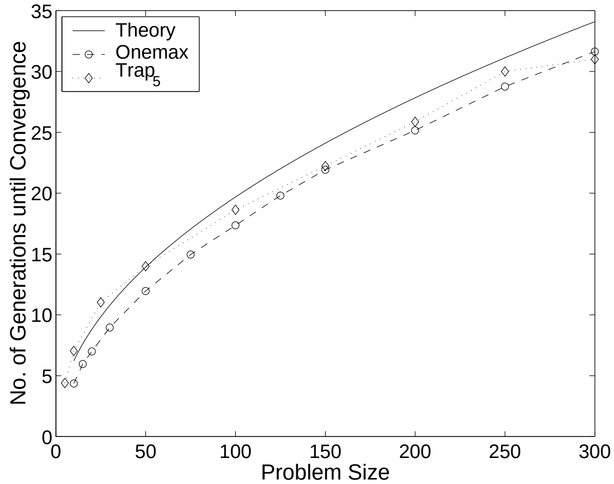


Figure 4.9: Number of generations until convergence of BOA on onemax and the composed trap of order 5. The problem sizes range from 0 to 300 bits. Large populations are used to approximate the assymptotic behavior.

of rather small populations, the convergence model of Ceroni, Pelikan, and Goldberg (2001) can be adopted in a similar fashion. Additionally, the above model assumes no external noise in the fitness function. To incorporate the effects of external noise, the noisy convergence model of Miller and Goldberg (1996a) can be used.

4.5.2 Exponential Scaling

The above convergence model assumed that all the subproblems converge in parallel. Under that assumption, the number of generations until convergence is $O(\sqrt{n})$. However, if the scaling of the subproblems is such that the subproblems converge sequentially, the time to convergence can further increase (Thierens, Goldberg, & Pereira, 1998; Lobo, Goldberg, & Pelikan, 2000; Rothlauf, 2001; Albert, 2001).

Let us get back to the example problem of exponentially scaled subproblems—the binary integer (see Equation 4.7). The binary integer converges one bit at a time, and its overall time to convergence is therefore proportional to the number of bits (Thierens, Goldberg, & Pereira, 1998):

$$G_{binInt} = O(n), \tag{4.94}$$

where n is the number of bits in the problem (problem size).

Since number of generations until convergence increases with the number of phases (different levels of scaling), the linear time gives us an upper bound on the time to convergence for decomposable problems of bounded difficulty. Regardless of the scaling of the different building blocks in the problem, the time to convergence should be somewhere between $O(\sqrt{n})$ and $O(n)$.

4.6 How Does BOA Scale Up?

Section 4.1 has argued that the number of fitness evaluations is an important quantity for determining the overall computational complexity of BOA. The section has stated that the number of evaluations can be bounded by a product of the population size and the number of generations until convergence. Boundary cases have been then analyzed and the bounds on both the population size as well as the number of generations until convergence have been presented. This section puts all the pieces of the theory together and computes the overall number of function evaluations until convergence to the optimum.

The overall result is summarized in Table 4.4. Under the assumption that the problem is decomposable into the subproblems of bounded order, there are two extreme cases. One one hand, if the subproblems in the decomposition are scaled the same, the total number of evaluations is given by

$$E_{uni} = O(n^{1.55}). \tag{4.95}$$

On the other hand, if the subproblems are scaled exponentially, the total number of fitness evaluations is given by

$$E_{exp} = O(n^2). \tag{4.96}$$

In both cases, the total number of evaluations is bounded by a low-order polynomial of the problem size.

Let us now compare the above result to the local search; in particular, consider the stochastic hill climber with bit-flip mutation described in Section 2.2.1. The hill-climber requires $O(n^k \ln n)$ fitness evaluations for the problem decomposable into subproblems of order k. While in BOA the difficulty of the problem (expressed in the form of k) is hidden in the constants preceding a polynomial of the fixed order, in the hill climber the difficulty directly affects the order of the growth

Problem Size	BOA	HC
$n = 100$	1.00	1.00
$n = 200$	2.93	32.00
$n = 300$	5.49	243.00
$n = 400$	8.57	1,024.00
$n = 500$	12.12	3,125.00

Table 4.3: The factor by which the number of evaluations increases in BOA and the hill climber (HC) on the problem decomposable into the uniformly scaled subproblems of order $k = 5$ (e.g., composed trap of order 5). Compared to the problem of size $n = 100$, the number of evaluations required by BOA to solve the problem of size $n = 500$ increases approximately 12.12 times, while the number of evaluations required by the stochastic hill climber increases $3,125$ times.

of the number of evaluations. Although for $k = 1$ or $k = 2$ this does not make a big difference, for moderate values of k the difference becomes the one between the tractable and the intractable (recall the comparison presented in the previous chapter in figures 3.11 and 3.12).

To get a better idea of the difference between BOA and the hill climber, consider the following example (see Table 4.6 for a summary of the example). Assume that both algorithms are capable of solving the problem of size $n = 100$ decomposable into subproblems of order $k = 5$ (e.g., the composed trap of order 5). To solve the same problem of twice the size, $n = 200$, the number of evaluations required by BOA would increase about 2.93 times, whereas the number of evaluations required by the hill climber would increase 32 times. If we were to solve the problem of size $n = 300$, the number of evaluations compared to the problem of size $n = 100$ for BOA would be larger by a factor of 5.49, while for the hill climber it would be 243. The difference further increases with the problem size. For instance, for the problem of size $n = 500$, the factor is only 12.12 for BOA but it is $3,125$ for the hill climber.

The situation becomes somewhat more complicated if the order of the building blocks in the problem is not bounded by a constant but grows with the problem size instead. It can be argued that if the character of the subproblems does not change much and their order grows logarithmically with the problem size, the performance further increases to a polynomial of higher order, because of the terms 2^k in the population-sizing bounds. Further refinement and generalization of the given theory remain for the future research.

The following section verifies the scalability estimates presented above on several problems of bounded difficulty.

	Uniform scaling	Exponential scaling
Population size	$O(n^{1.05})$	$O(n)$
Time to convergence	$O(\sqrt{n})$	$O(n)$
Total evaluations	$O(n^{1.55})$	$O(n^2)$

Table 4.4: A summary of the total number of evaluations in BOA. Assuming a fixed order of problem decomposition, the overall number of fitness evaluations is expected to grow subquadratically or quadratically with the size of the problem.

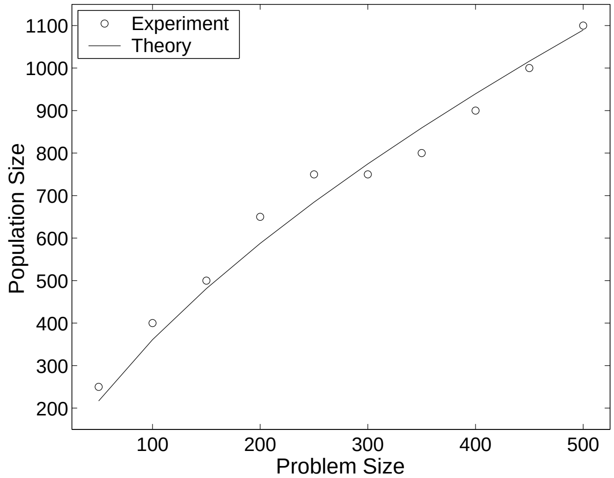
4.7 Empirical Verification of BOA Scalability

This section verifies the above estimates of the population size and the number of generations until convergence by comparing the theoretical estimations and the actual performance of BOA. The first part considers uniformly scaled problems and the second one considers exponentially scaled problems.

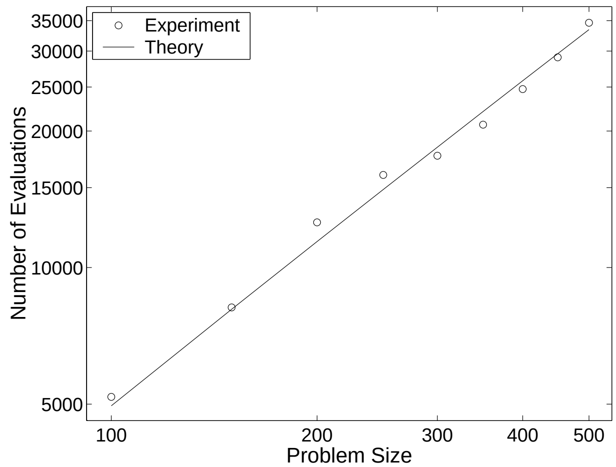
All the experiments are done within the same framework. For each problem and each problem size, 30 independent runs have been performed, and the population size was determined by the bisection method as the minimal population size for the algorithm to succeed in all of the 30 runs. Binary tournament selection without replacement was used. The worst half of the original population is replaced by the same number of offspring. BIC metric was used to build the model based on the selected population. Each generation, the model was created from scratch. All results were averaged over the 30 independent runs performed as described above.

4.7.1 Uniform Scaling

To verify the scalability results for uniform problems, BOA was tested on onemax, the composed trap of order 5, and the composed deceptive function of order 3. Onemax is defined as the sum of bits in the input string (see Section 2.2.1). The composed trap function of order 5 is defined as the sum of single trap functions of order 5 over non-overlapping 5-bit partitions of the solution strings (see Section 2.2.2 on page 27 for the detailed definition). Again, the partitioning in the composed trap is fixed but there is no information about the positions in each partition. The two optima in each trap are given by $f_{low} = 4$ and $f_{high} = 5$. The composed deceptive function of order 3 (Pelikan et al., 1998) is defined as the sum of single deceptive functions of order 3 over non-overlapping 3-bit partitions of the solution strings. Similarly as in the trap, the partitioning in the composed



(a) Population size (arithmetic scale).



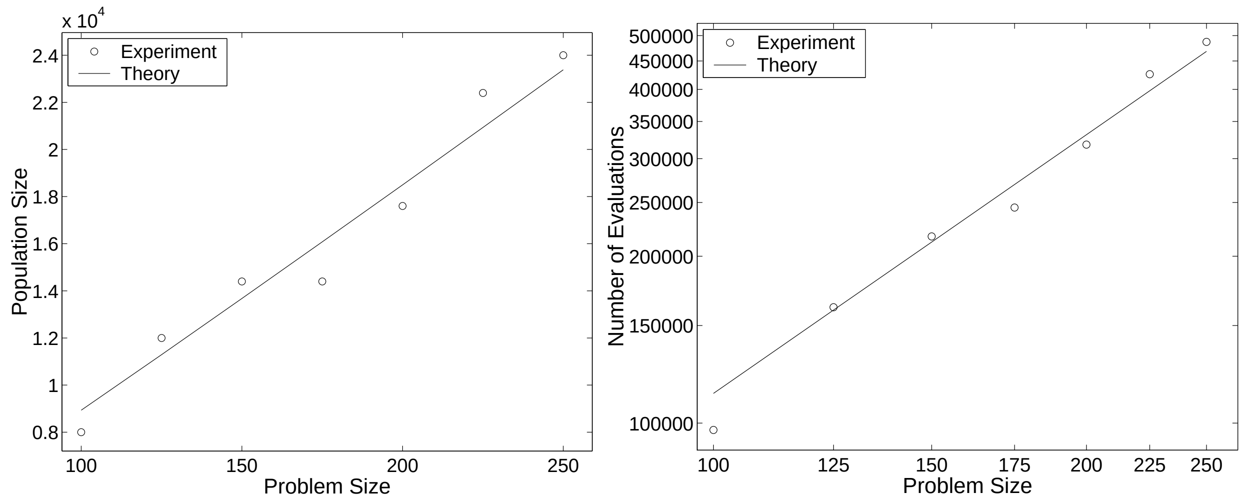
(b) Number of evaluations (log scale).

Figure 4.10: The performance of BOA on onemax for the problem sizes ranging from $n = 100$ to $n = 500$ bits. The required population size and the number of evaluations until the optimum was found are shown and compared to the theory. The empirical results match theory well.

deceptive function is fixed but there is no information about the positions in each deceptive block. The fitness contribution of each block of 3 bits is given by Equation 3.6.

On onemax, the model learned by BOA does not have to encode any interactions. Therefore, the model-building population sizing does not have to be considered and the expected performance of BOA should be close to that of the simple GA with population-wise uniform crossover. The population sizing should follow the gambler's ruin model (Harik, 1999), and the number of generations should follow the onemax convergence model for population-wise uniform crossover (Mühlenbein & Schlierkamp-Voosen, 1993). Therefore, the total number of evaluations should grow as $O(n \log n)$. Figure 4.10 shows the total number of evaluations until the convergence of BOA on onemax. The results much the theory well. However, due to the errors in sampling the solutions using finite populations, BOA finds some unnecessary dependencies; consequently, BOA processes larger partitions than necessary, and its performance increases compared to the simple GA with uniform crossover, which represents a "perfect" model for this case. The increase can be explained by the gambler's ruin population-sizing model (Harik, 1999), which predicts that the population size should grow exponentially with the size of the building blocks.

Figure 4.11 shows the population size and the total number of evaluations until the optimum



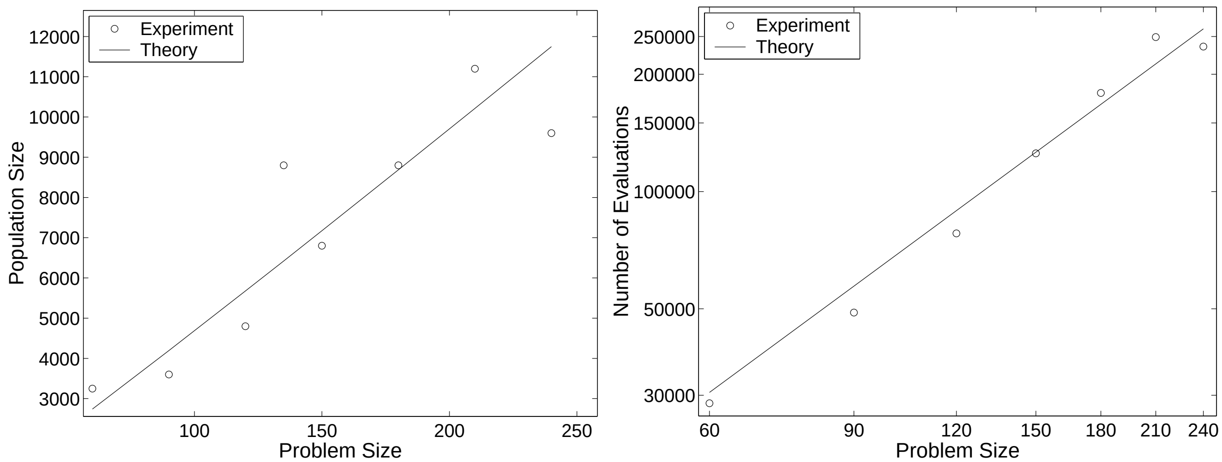
(a) Population size (arithmetic scale).

(b) Number of evaluations (log scale).

Figure 4.11: The performance of BOA on the composed trap function of order 5 for the problem sizes ranging from $n = 100$ to $n = 250$ bits. The required population size and the number of evaluations until the optimum was found are shown and compared to the theory. The results match the theory well.

was found on the composed trap function of order 5 of varying problem size. The size of the tested problems ranged from $n = 100$ to $n = 250$. Figure 4.12 shows the population size and the total number of evaluations until the optimum was found on the composed deceptive function of order 3 of varying problem size. The size of the tested problems ranged from $n = 60$ to $n = 240$.

The latter two problems (trap and deceptive) require that the model learned by BOA is correct. The population-sizing model for building a correct model must therefore be considered. The expected number of evaluations should grow as $O(n^{1.55})$. The results show that the BOA population-sizing theory approximates the expected number of evaluations well, although as discussed in the previous chapter, a better fit is obtained by $O(n^{1.65})$. The differences appear due to the effects of finite populations and the approximations made in developing the theory. However, in both cases the growth follows the expected results accurately and we can thus conclude that the theory approximates the actual performance well.



(a) Population size (arithmetic scale).

(b) Number of evaluations (log scale).

Figure 4.12: The performance of BOA on the composed deceptive function of order 3 for the problem sizes ranging from $n = 60$ to $n = 240$ bits. The required population size and the number of evaluations until the optimum was found are shown and compared to the theory. The empirical results match theory well.

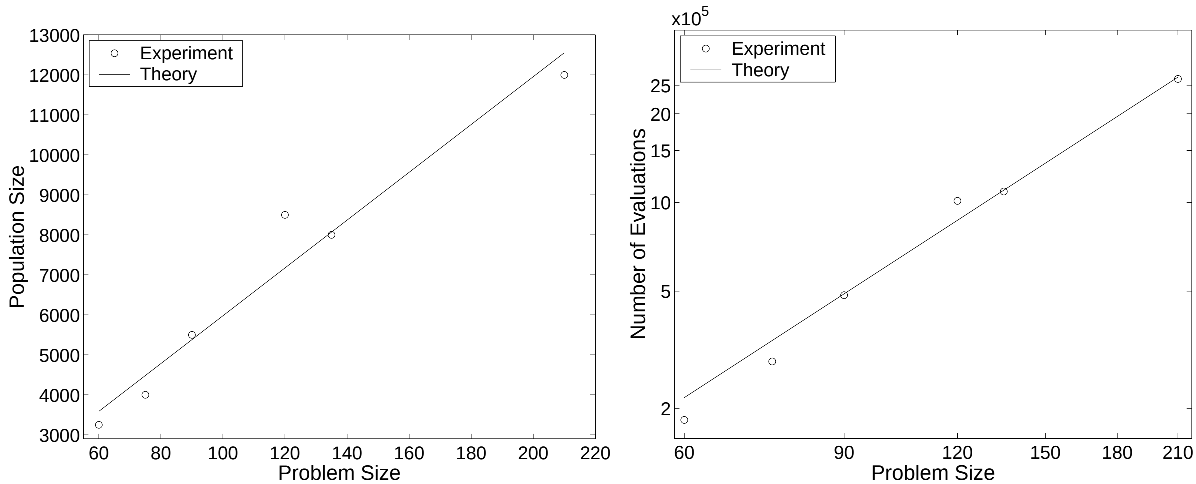
4.7.2 Exponential Scaling

To compare the results for exponentially scaled problems, we modified the composed deceptive function of order 3 by scaling the partitions corresponding to the deceptive functions so that the signal decreases exponentially in a specified sequence of partitions. To ensure that a particular partition in the sequence depends more than all the subsequent partitions according to the sequence, it is sufficient to multiple the ith partition by c^i, where c satisfies the following inequality:

$$0.1c^{\frac{n}{3}} > \frac{n}{3} - 1. \tag{4.97}$$

The last equation restricts c so that the smallest signal coming from ith partition is greater than the sum of the maximum signals coming from the remaining partitions on positions 1 to $(i-1)$ in the sequence.

Figure 4.13 shows the required population size and the total number of evaluations until the optimum was found for the function described above. The size of the tested problems ranged from $n = 60$ to $n = 210$. Similarly as in the case of uniform scaling, the theoretical growth of the population size and the number of evaluations was fitted to the empirical results. The growth



(a) Population size (arithmetic scale).

(b) Number of evaluations (log scale).

Figure 4.13: The performance of BOA on the exponentially scaled composed deceptive function of order 3 for the problem sizes ranging from $n = 60$ to $n = 240$ bits. The required population size and the number of evaluations until the optimum was found are shown and compared to theory. The empirical results match theory well.

follows the expected results accurately and we can thus conclude that the theory approximates the actual performance well.

4.8 Summary

This chapter developed a BOA scalability theory for problems decomposable into subproblems of bounded order. The total number of evaluations until reliable convergence to the optimum was used to measure BOA's complexity. Additionally, the chapter verified the theory on an array of decomposable problems. A summary of the key points of this chapter follows:

- The number of evaluations of the objective function until an algorithm converges to the optimum with high confidence is an important quantity for analyzing the scalability of stochastic optimization techniques. The total number of evaluations in BOA is proportional to a product of the required population size and the number of generations until convergence:

$$E = O(N \times G).$$

- There are four factors influencing the population sizing in BOA:

  1. Initial supply of building blocks.
  2. Decision making between building blocks and their competitors.
  3. Genetic drift (delayed supply of building blocks).
  4. Model building.

  The first three factors were studied in GA population-sizing theory (Goldberg, Deb, & Clark, 1992; Harik, Cantú-Paz, Goldberg, & Miller, 1999; Thierens, Goldberg, & Pereira, 1998; Lobo, Goldberg, & Pelikan, 2000; Rothlauf, 2001; Albert, 2001). The chapter analyzed the last factor—the model building.

- The analysis provided a near-linear bound on the size of the population required for finding a correct model for uniformly scaled decomposable problems of order k with binary tournament selection and the BIC metric:

  $$N_{uni} = O(2^k n^{1.05}).$$

  For exponentially scaled problems, the genetic-drift population-sizing model becomes dominant and the population size should grow as

  $$N_{exp} = O(n).$$

- The time to convergence was considered separately for two boundary cases of scaling the fitness contributions of the subproblems in a proper decomposition: (1) uniform scaling, and (2) exponential scaling. For uniform scaling, all subproblems converge in parallel and the overall convergence time grows with the square root of the problem size,

  $$G_{uni} = O(\sqrt{n}).$$

  For exponentially scaled subproblems the convergence proceeds sequentially, one or a few subproblems at a time, and the overall number of generations grows at most linearly with the problem size,

$$G_{exp} = O(n).$$

- The total number of evaluations until convergence can be estimated by

$$E_{uni} = O(2^k n^{1.55})$$

for uniformly scaled subproblems, and

$$E_{exp} = O(n^2)$$

for exponentially scaled subproblems.

- The hill climber's performance depends on the order k of the subproblems in a proper problem decomposition and the overall time to convergence grows as $O(n^k \ln n)$. On the other hand, in BOA the order k is hidden in the constants in front of the polynomial of a fixed, low order; BOA converges in $O(n^{1.55})$ or $O(n^2)$ independently of k (assuming k is upper-bounded by a constant).

- The difference between BOA and the hill climber grows with k. For example, for the problem of size $n = 500$ decomposable into equally scaled subproblems of order $k = 5$, the number of evaluations required by BOA increases by a factor of about 12 compared to the same problem of size $n = 100$, while for hill climber the factor becomes $3,125$.

Chapter 5

The Challenge of Hierarchical Difficulty

Thus far, we have examined the Bayesian optimization algorithm (BOA), empirical results of its application to several problems of bounded difficulty, and the scalability theory supporting those empirical results. It has been shown that BOA can tackle problems that are decomposable into subproblems of bounded order in a scalable manner and that it outperforms local search methods on difficult decomposable problems. But can BOA be extended beyond problems of bounded difficulty to solve other important classes of problems? What other classes of problems should be considered?

The purpose of this chapter is twofold. First, the chapter poses the challenge of solving problems that are not decomposable into subproblems of bounded order, but can be decomposed over a number of levels of difficulty into a hierarchy. The chapter identifies the important features that must be incorporated into BOA to solve such problems in a scalable manner. Second, the chapter presents a class of hierarchically difficult problems that challenge any optimization algorithm because of the three inherent features of this class of problems. First of all, the designed problems are not decomposable into subproblems of bounded difficulty. Second, the problems contain exponentially many local optima that make the problems unsuitable for any local search. Finally, the problems deceive any optimization technique based on local operators away from the global optimum except for several special cases.

The chapter starts by motivating the use of hierarchical decomposition in reducing the complexity of the problem. Section 5.2 introduces basic concepts of the hierarchical problem solving and presents the important features that must be incorporated into BOA to extend its applicability

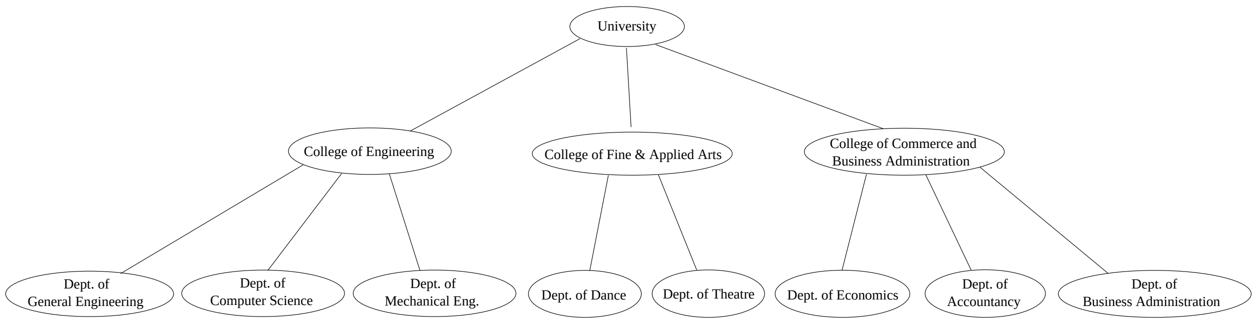


Figure 5.1: The hierarchical structure of a university. The figure shows only a small fraction of the complex organizational hierarchy.

to hierarchical problems. Section 5.3 proposes the class of problems called hierarchical traps that challenge any problem solver. Finally, Section 5.4 summarizes the chapter.

5.1 Hierarchical Decomposition

Many complex systems and processes in business, engineering, science, as well as nature, are hierarchical. By hierarchy, we mean a system composed of subsystems each of which is a hierarchy by itself until we reach some bottom level (Simon, 1968). Interactions within each subsystem are of much higher magnitude than the interactions between the subsystems. There are plenty of hierarchy examples around us. A human body is composed of organs, organs are composed of tissues, tissues are composed of cells, and so forth. A university (see Figure 5.1) is composed of colleges, colleges are composed of departments, departments are composed of laboratories and offices, and so forth. A program code is composed of procedures and functions, procedures are composed of single commands and library calls, commands are composed of machine code or assembly language, and so forth.

Why do we talk about hierarchy when what we are really interested in is problem solving and optimization? Most of the complex problems that humans have successfully solved could be tackled only because of the use of hierarchical decomposition. Single-level decomposition discussed in the previous chapters by itself simplifies the problems by allowing the solver to focus on multiple simpler problems instead of one large problem. However, not every problem can be decomposed into tractable subproblems; such decomposition may be obstructed due to the rich interaction structure of the problem or the lack of feedback for discriminating alternative solutions to the different subproblems in a fine enough decomposition. Hierarchical decomposition adds a new level of complexity reduction by allowing the decomposition to go down a number of levels until we finally get to the problem that we can solve. Of course, as discussed above, the subproblems in the decomposition on each level are allowed to interact, but the interactions within each subproblem must be of much higher magnitude than those between the subproblems.

5.2 Computer Design, von Neumann, and Three Keys to Hierarchy Success.

This section identifies three important issues one must consider to solve difficult hierarchical problems quickly, accurately, and reliably. An intuitive example drawn from computer design informally motivates each issue, which is then formalized and discussed in context of computational optimization and search.

For many of us, computers have become just as important part of our everyday life as cars, refrigerators, and toasters. However, the computer design is an extremely difficult task and many great minds contributed to the effort of making that dream come true. One of the most important contributions to this effort was that of John L. von Neumann who decomposed the design of the "general-purpose computing machine" into four basic components, each of which focused on one task, in particular, arithmetic, memory, control, and connection with the human operator. These four subsystems have formed the basis of most computers up to date as the arithmetic/logic unit, the memory, the control unit, and the input/output devices (see Figure 5.2).

Further reduction of complexity calls for the use of hierarchy. Each of the four components can be simplified by an additional level of decomposition. For example, a typical hard drive[1] is composed of the controller, heads, magnetic disks, and so forth. Decomposition can go down a number of levels until the problems on the bottom level become tractable.

Although the components on each level in the computer decomposition interact, the design of these components can be oftentimes considered independently. For example, the design of the memory unit does not depend much on the design of the input/output interface. That is why

[1] Hard disk drive—often shortened as hard disk or hard drive—is a frequently used permanent memory device. Hard drives are much slower than on-chip memories, but they allow permanent storage of large amounts of data and are much cheaper. Hard disks also allow a relatively fast access to any location of the media.

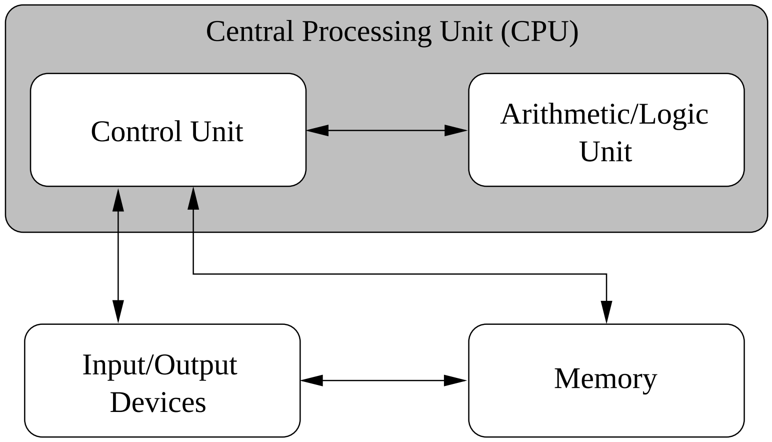


Figure 5.2: Von Neumann's computer architecture consists of the four main units: (1) control unit, (2) arithmetic/logic unit, (3) input/output unit, and (4) memory unit. The component composed of the control unit and the arithmetic/logic unit forms what we now call the central processing unit (CPU).

the researchers that are trying to make computer memory chips smaller, faster, and cheaper, do not have to consult each their invention or experience with the ones developing easier-to-use, more powerful, and cheaper input/output devices. Decomposition (and repeated decomposition) is where the simplification of the problem comes from. That leads to the first key to hierarchy success:

Issue 1: Proper decomposition. On each level of problem solving, we must be capable of decomposing the problem properly. The decomposition allows the solver to focus on the different subtasks in separation and thus reduce complexity. The decomposition can be as simple as the partitioning of the decision variables in the problem into several disjoint subsets but it can be as complex as the graphs with a bounded number of neighbors of each node.

When a particular memory device, such as a hard drive, has been constructed, it becomes irrelevant to consider the details concerning the material or technology used; only several important features suffice. Elimination of irrelevant features of the subsystems from the lower level of the hierarchy reduces the complexity of juxtaposing the subsystems on the current level. In the hierarchical problem solver, it is desirable to eliminate irrelevant details of the solutions from lower levels and represent these in a compact way. This leads to the second key to hierarchy success:

Issue 2: Chunking. The solutions to the subproblems from the lower level can be seen as *chunks* of the solutions that are used as basic building blocks for constructing the solutions on the current level. In other words, the partial solutions to each subproblem can be treated as the instances of a single variable that is used to compose the solutions on the next level. The hierarchical problem solver must be capable of representing these chunks of solutions from lower levels in a compact way so that only relevant features are considered.

There are many alternative processors, motherboards, memory chips, and hard drives, which can be used to construct the computer system. The requirements on the final design may suggest which of the alternatives is the best one, but the feedback on the current level may be insufficient to do so. Some of these hard-to-decide choices in the history of computer architecture include the number of machine instructions of CPU[2] and the number of registers used by CPU. Furthermore, the features of one component may strongly influence the requirements on another component. For instance, better control devices for hard drives enable faster processing of the requests and allow more complex hardware ensuring the physical task of storing and retrieving the data. That is why it is necessary that multiple alternatives are maintained for each component and the final choice is made only when there is enough feedback to discard some alternatives. This leads to the third key of hierarchy success:

Issue 3: Preservation of alternative candidate solutions. The hierarchical problem solver must be capable of preserving multiple alternative solutions to each subproblem. There are two reasons for doing this:

(1) On the current level there may not be a sufficient feedback to discriminate among a few best alternative solutions to the considered subproblem.

(2) Although the subproblems on the current level are considered independent, interactions on some higher level or levels may lead to new information that favors some of the alternatives over others.

The previous chapters have shown that BOA is capable of learning a proper decomposition of the problem as the optimization proceeds and using the learned decomposition to propagate promising partial solutions corresponding to each subproblem. BOA is thus a good starting point in the design

[2]CPU stands for the central processing unit that is composed of the control unit, the arithmetic/logic unit, and the set of registers.

of a hierarchical problem solver or optimizer. However, the remaining keys to hierarchy success—the chunking and the preservation of alternative solutions—must first be incorporated into the basic algorithm. We return to this topic in the following chapter. Here we continue by describing the class of hierarchical problems that challenge any optimization algorithm, because they are not decomposable on a single level and because they are intractable for most popular optimization techniques to date. In addition to providing a challenge for the optimizers, the designed class of problems should clarify the ideas of hierarchical problem solving and decomposition.

5.3 The Design of Challenging Hierarchical Problems

It is common in the design of material machines like airplanes and toasters to test the design at the boundary of its design envelope. To test an algorithm on whether it is capable of exploiting decompositional bias in a scalable manner, composed traps introduced earlier in this thesis can be used. However, all the previously discussed problems were solvable by a single-level decomposition without forcing the use of a more robust and general bias based on hierarchy.

The purpose of this section is to design a class of challenging hierarchical problems that can be used to test the scalability of the algorithms on difficult hierarchical problems. The design is guided by the three keys to hierarchy success presented in the previous section: (1) proper decomposition, (2) chunking, and (3) preservation of alternative solutions. The section starts by introducing a general class of hierarchically decomposable test problems. Next, the section motivates the design of the class of *hierarchical trap functions* and presents several such functions.

5.3.1 Example: Tobacco Road

To get a better idea of what we mean by a hierarchical function—in particular, a challenging hierarchical function—let us start with an example called the *tobacco road function* introduced by Goldberg (1998). To better understand the example, please follow Figure 5.3.

The tobacco road function is defined on two levels. On the bottom level, the input string is partitioned into partitions of 6 bits each, where each partition is evaluated using the folded trap function of order 6 (see Figure 5.4). The partitioning can be chosen arbitrarily but it remains fixed for all evaluations. The folded trap has two global optima, one in 000000 and the other one in

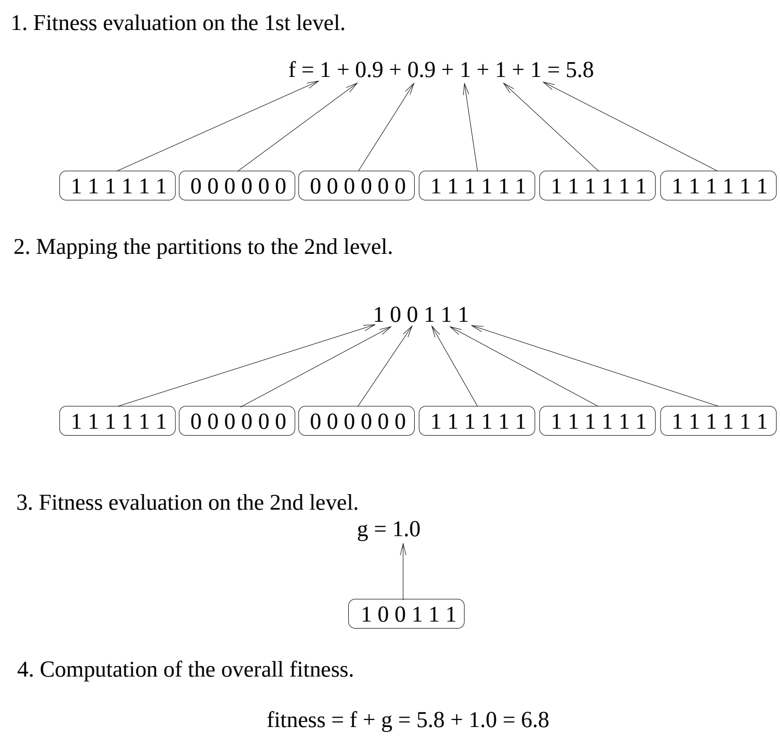


Figure 5.3: The tobacco road function is defined on two levels. The solution on the bottom level is first evaluated using the folded traps and then mapped to form the solution on the second level. The second-level solution is then evaluated using the traps and the overall fitness is computed by adding the contributions of both the levels together.

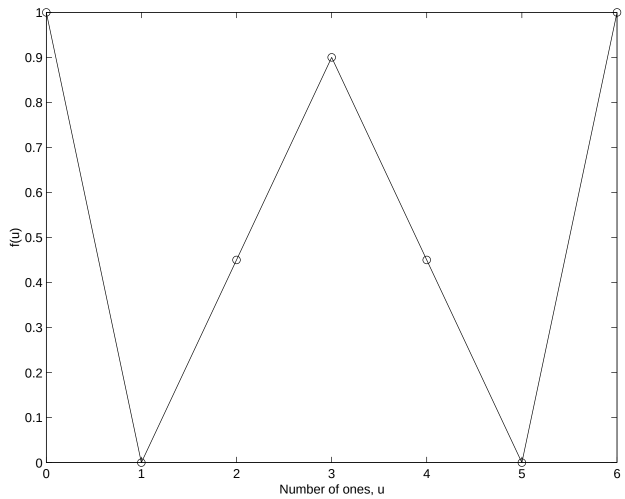


Figure 5.4: The folded trap function contains two global optima, one in the string of all zeroes, and one in the string of all ones. Additionally, the folded trap contains a number of deceptive attractors in the strings with half zeroes and half ones. Deception and multimodality make folded traps a great challenge for any optimization method.

111111. Additionally, the folded trap function contains a large number of local optima in all the strings that contain half ones and half zeroes, which is right in the middle between the two global optima. Similarly as the single trap, the folded trap cannot be decomposed without losing the global optima and converging to some of the local ones. Moreover, the deception in the folded trap is somewhat more harmful than the one in the single trap, since the folded trap contains many more local optima than the global ones.

The bits in each partition corresponding to one of the folded traps are then mapped to one particular bit on the next level: 000000 maps to 0, 111111 maps to 1, and the rest maps to a null value that stands for an undefined bit and is denoted by '-'. After mapping the partitions, the next level contains six times less bits than the bottom level and some of those bits are undefined.

The new bits on the second level are then partitioned into partitions of 6 bits. Again, the partitioning is fixed for all evaluations. Note that each partition of 6 bits on the second level is determined by some subset of 36 bits on the bottom level. Each of the 6-bit partitions of the second level contribute to the overall fitness function by the trap function (see Figure 1.4). If some of the bits in a particular partition are undefined, the contribution of the partition is set to 0.

Finally, the fitness contributions of both the levels are added together to determine the overall fitness of the evaluated solution.

The tobacco road function can be solved in either one or two stages. If the algorithm approaches the tobacco road in one stage, the decomposition must contain chunks of 36 bits. Although it is possible to consider subproblems of such a high order, this is not necessary. Solving the problem in two stages yields much more promising results. In particular, the first phase can focus on optimizing the bottom level yielding solutions with blocks 000000 and 111111 on all partitions of the bottom-level decomposition. If the algorithm was capable of recognizing that blocks 000000 and 111111 actually represent only two alternatives and treated them as if they were single bits, the second stage would proceed analogously to the first stage and the blocks would be combined to form the optimum.

The difference between the two approaches—the single-stage, single-level approach and the two-stage, hierarchical approach—is in the order of subproblems that must be considered to find the optimum. An important observation is that the number of fitness evaluations will differ by a factor of about 2^{30}, because the overall complexity grows exponentially with the order of the problem decomposition. In other words, just considering the problem on two levels decreases the number of evaluations by a factor of more than one billion—more precisely, $1,073,741,824$.

The tobacco road function is also extremely difficult for any local search. Since the blocks of 36 bits must be considered if approached on a single level, the performance of the hill climber would grow with $O(n^{36} \ln n)$. Although n^{36} is a polynomial, solving the problem in $O(n^{36} \ln n)$ evaluations becomes intractable incredibly fast.

Therefore, there is no question whether the tobacco road functions are challenging for BOA or any other optimization algorithm. Furthermore, the difference between the hierarchical and single-level approaches increases with the number of levels and even more difficult problems could be therefore designed in a straightforward manner.

The following section formalizes the notion of hierarchically decomposable problems. Subsequently, the class of hierarchical trap functions is introduced where the number of levels grows with the problem size.

5.3.2 Hierarchically Decomposable Functions

The idea of using hierarchical functions in challenging GAs and other optimization techniques dates back to the works on royal road functions of Mitchell, Forrest, and Holland (1992), hierarchical if-and-only-if functions of Watson, Hornby, and Pollack (1998), tobacco road functions of Goldberg (1998), hyperplane-defined functions of Holland (2000), and others (Pelikan & Goldberg, 2000b; Pelikan & Goldberg, 2001). This section attempts to formalize all of the aforementioned approaches similarly as it was done by Watson, Hornby, and Pollack (1998), and Pelikan and Goldberg (2000b).

Hierarchically decomposable functions (HDFs) are defined on multiple levels. The solution on each level is partitioned according to the structure of the function. The bits can be partitioned arbitrarily, but the partitioning must be fixed for all evaluations. The block of bits in each partition is evaluated according to some function. Each block is then mapped to the next level into a single symbol of some alphabet. The mapping functions can be chosen arbitrarily. The overall fitness is computed by summing the contributions of all the partitions on all the levels. More formally, an HDF can be defined by the following three components (see Figure 5.5 for the definition of the components for the tobacco road function):

1. **Structure.** The structure of the function defines the way in which the variables on each level are partitioned. Partitions are used to both contribute to the fitness of the solution and map to the next level. The structure forms a tree with one-to-one mapping between the leaves of the tree and the variables in the problem. Here we only use balanced trees of fixed order[3]. The levels are numbered from the bottom—the first level is composed of the leaves, and the last level is made of the children of the root. In most cases we omit the root because it is unnecessary.

2. **Mapping functions.** The mapping functions, also called the interpretation functions, define how to map solutions from a lower level to become inputs of both the contribution and interpretation functions on the current level.

3. **Contribution functions.** The contribution functions define how much the partitions of

[3]Balanced trees are those where for every internal node (node with children), the number of descendants of each its child is the same. In the tree of fixed order, the number of children (direct successors) is equal either to the given order (internal node) or to 0 (leaf).

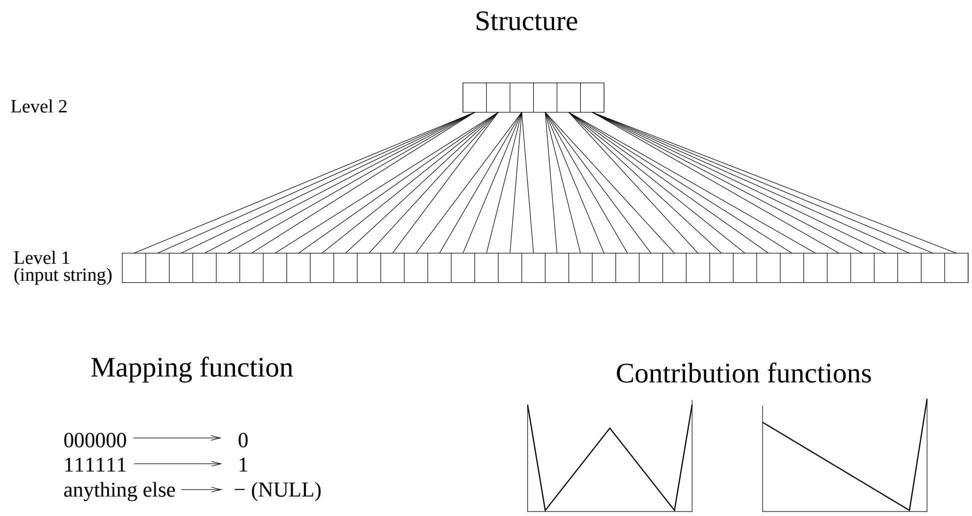


Figure 5.5: The three components defining the tobacco road function. The size of the example problem is 36 bits.

the decomposition on each level contribute to the overall fitness. Since the mapping of the solution to the next level reduces the number of partitions on the next level, it is often useful to scale the contributions on each level so that the total contribution of each level remains the same.

Before describing hierarchical traps, we present the aforementioned functions that contain some form of hierarchy and discuss their problem difficulty. Later we shall argue that hierarchical traps challenge optimization methods in a qualitatively different way and that without "cheating" there is no way of getting around the difficulty introduced by hierarchical traps.

5.3.3 Another Example: Royal Road

The class of royal road functions was proposed by Mitchell, Forrest, and Holland (1992) as a way of describing the hierarchical nature of optimization in GAs and providing an example function that GAs were born to solve. Although royal roads lack in several aspects of the problem difficulty that we are interested in, the idea of hierarchy in GA problem solving manifested by GA behavior on the royal roads was an interesting and influential one.

A general definition of the royal road functions is given by the list of fitness contributions of a subset of partial solutions (Mitchell, Forrest, & Holland, 1992). Here we only discuss a simple royal road function adopted from Mitchell et al. (1992) that contains hierarchy and thus relates to

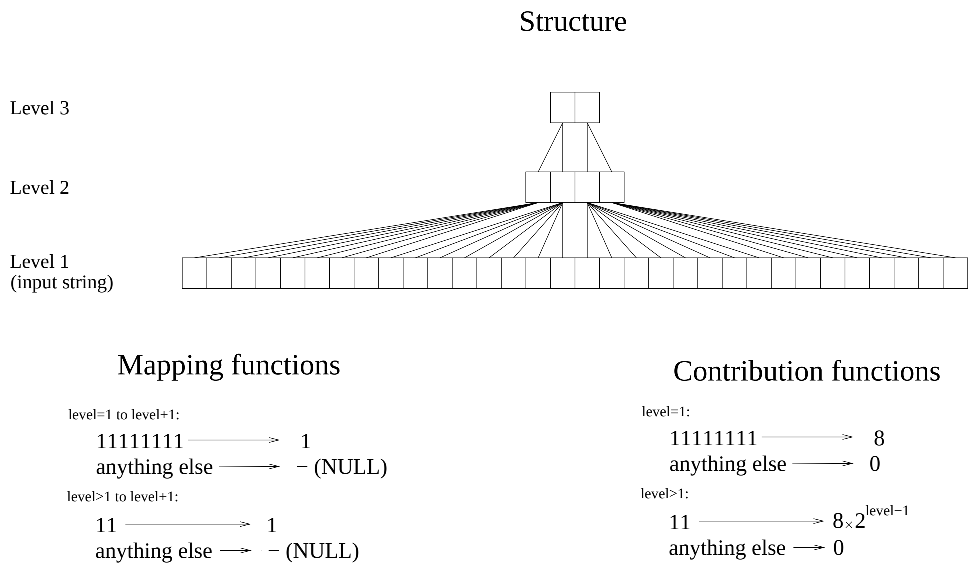


Figure 5.6: The three components defining the royal road function. The size of the example problem is 32 bits.

the topic of hierarchical difficulty studied in this chapter.

The value of the royal road for the input binary string is computed in several stages. In the first stage, the input binary string solution is partitioned into partitions of 8 consecutive bits each. Each block of bits in these partitions contributes to the overall fitness by 8 if the block is 11111111; otherwise, the block does not contribute to the overall fitness at all. Each subsequent stage merges the pairs of neighboring blocks together, yielding half as many blocks of twice as large order. In the second stage, for instance, blocks of 16 consequent bits each are created. Each of the created blocks of bits then contributes to the overall fitness by the number of bits included in the block if all the bits in the block are 1. So in the second stage, for instance, each block 1111111111111111 contributes to the overall fitness by 16. The evaluation continues until the block covers the entire string; the string length should therefore be limited to the powers of 2 multiplied by 8. See Figure 5.6 for the three-component HDF definition of the royal road.

The royal road described above has one global optimum in the string of all ones and no local optima. The fitness contributions of the blocks of ones grow with the size of the blocks and all the signal leads to the global optimum. Consequently, GAs are able to solve the royal roads efficiently. However, because of the same reasons, the royal roads are easy even for the stochastic hill climber described in Section 2.2.1. The reason for this is that in the royal roads, there is nothing to mislead the hill climber to the wrong direction and it is sufficient that the hill climber is capable of crossing 8-bit plateaus. Even more importantly, the royal roads do not challenge the algorithm to obey the three keys to hierarchy success presented before and are efficiently solvable by any of the standard GAs.

5.3.4 Yet Another Example: Hierarchical If-and-Only-If (HIFF)

The hierarchical if-and-only-if function was proposed by Watson, Hornby, and Pollack (1998) as an example of a function that is not separable and should therefore challenge even those GAs that are capable of linkage learning. Although it is not necessary to introduce interdependencies of unbounded order to make traditional GAs fail[4], HIFF joins other decomposable problems of bounded difficulty in the club of functions that are difficult to solve using traditional GA operators.

The structure of HIFF is a balanced binary tree. The input to the contribution and mapping functions therefore consists of two symbols. A single mapping function is used on all levels where 00 is mapped into 0, 11 is mapped into 1, and everything else is mapped into the null symbol '-'. On each level, blocks 00 and 11 contribute to the overall fitness by 2^{level}, where *level* is the number of the current level (again, the bottom level is level 1). Anything else does not contribute to the overall fitness. In Watson's definition of HIFF (Watson, Hornby, & Pollack, 1998), the leaves in the tree (the single bits) contribute to the fitness by 1, so the entire fitness is increased by the problem size n. Since the structure must be a balanced binary tree, the size of the problem should be restricted to the powers of 2. Figure 5.7 shows the three HDF components defining HIFF.

HIFF has two global optima, one in the string of all ones and one in the string of all zeroes. To successfully solve HIFF, the algorithm must preserve either zeroes or ones on all string positions to ensure that the optimum can be reached. Although the loss of particular bits on some positions can be taken care of by mutation or other local operator, the complexity of changing any partition grows exponentially with its order and, therefore, there must be a limited number of such problematic positions. There are two alternative ways of solving HIFF. The algorithm can decide whether to go after zeroes or ones, or preserve both alternatives as the optimization proceeds. In the latter case, the algorithm must ensure preservation of the partitions on the current level of optimization,

[4]Recall composed traps, which beat uniform crossover (see Section 2.2.2).

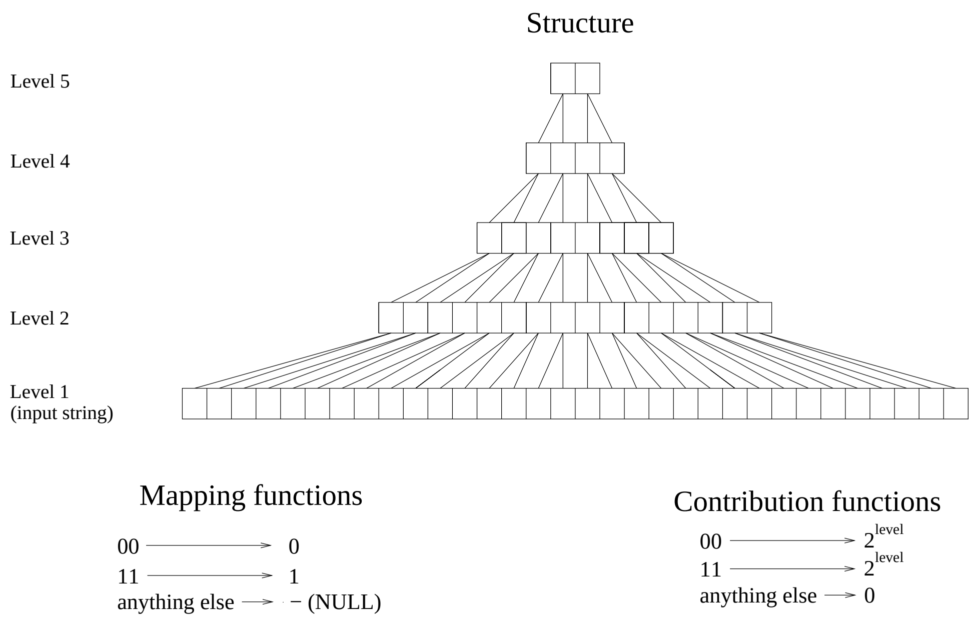


Figure 5.7: The three components defining the HIFF function. The size of the example problem is 32 bits.

because mixing zeros and ones moves the optimization one or more levels down. Of course, the chunks of ones and zeroes must be combined together. The last issue brings us back to linkage learning.

The following section presents hierarchical traps, which add new difficulty to tobacco road functions and HIFF by combining the ideas of the two designs into one.

5.3.5 Hierarchical Trap Functions: The Ultimate Challenge

Before designing hierarchical traps, let us recall the important goals of the design:

1. **Force proper decomposition.** Hierarchical traps must force the algorithm to learn a proper decomposition. In other words, if the algorithm fails to do this, the computational complexity will grow with the problem size prohibitively fast or the optimum will not be found at all.

2. **Force preservation of alternative solutions.** Hierarchical traps must force the algorithm to preserve alternative partial solutions. Failing to do this should result in either intractable or unsuccessful search for the global optimum.

3. **Force chunking.** Hierarchical traps must force the algorithm to manipulate large pieces of solutions. Similarly as in the above two cases, failing to do this should result in either intractable or unsuccessful search for the global optima.

To ensure the achievement of all the above goals, the features of the tobacco road function and HIFF are combined with the ones of challenging problems decomposable on the single level represented by the trap functions. We start by presenting the general definition of hierarchical traps. Subsequently, we specialize the general structure yielding two important hierarchical trap problems that are used in the remainder of the thesis.

The three components defining a general hierarchical trap are listed in the following (see also Figure 5.8):

1. **Structure.** Hierarchical trap functions use a balanced k-ary tree as the underlying structure, where $k \geq 3$. The minimal k is given by the minimal order of deceptive functions that are to be used as contribution functions. Since we are interested in the scalability of the tested algorithm and the problem sizes must grow as powers of k, it is reasonable to set k to a small value, preferably 3.

2. **Mapping Functions.** The mapping functions map blocks of all 0's and 1's to 0 and 1, respectively, similarly as in the tobacco road and HIFF. Everything else is mapped to '-'.

3. **Contribution Functions.** Contribution functions are based on the trap function of order k (see Figure 1.4):

$$trap_k(u) = \begin{cases} f_{high} & \text{if } u = k \\ f_{low} - u\frac{f_{low}}{k-1} & \text{otherwise} \end{cases} .$$

   The height of the two optima in each trap (denoted by f_{low} and f_{high}) depends on the current level. The difficulty of hierarchical traps can be tuned by parameters f_{low} and f_{high} on the different levels.

Let us now specify the parameters for the two hierarchical traps that are going to be used in our experiments, denoted by f_{htrap1} and f_{htrap2}, respectively. In both f_{htrap1} and f_{htrap2}, the underlying structure is a ternary tree (i.e., $k = 3$). The two hierarchical traps differ in the parameters f_{low} and f_{high} on each level.

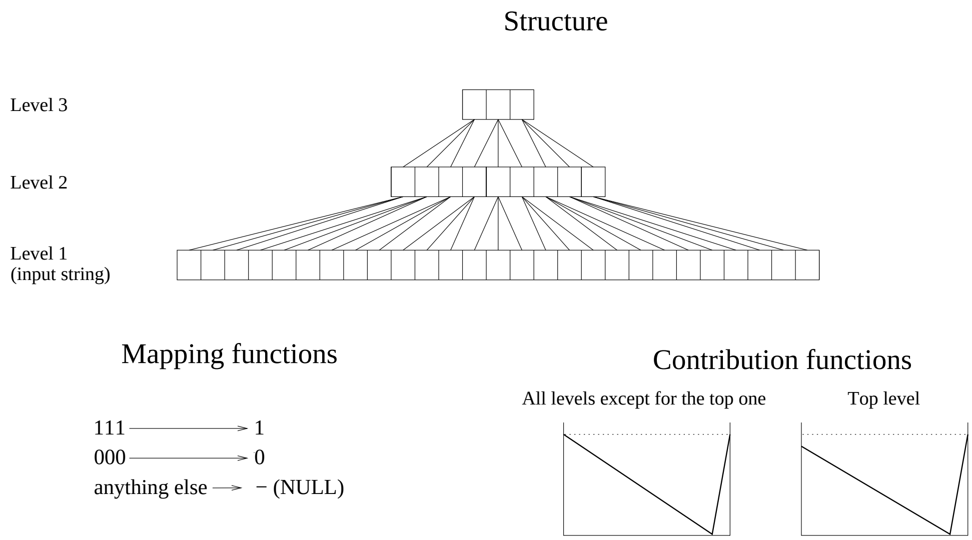


Figure 5.8: The three components defining the hierarchical trap function of order $k = 3$. The size of the example problem is 27 bits. The problem size of $n = 27$ implies that there are three levels in the problem.

The hierarchical trap f_{htrap1} sets $f_{high} = f_{low} = 1$ on all levels except for the top one. That means that in the traps on all the levels except for the top one, the two optima are equally good. In the trap on the top level, the optimum in 111 has a value of $f_{high} = 1$ and the optimum in 000 has the value of $f_{low} = 0.9$. The contributions on each level are multiplied by 3^{level} so that the total contribution of each level is the same. There are several features of f_{htrap1} worth mentioning at this point:

- There is no way of deciding whether 000 is better than 111 in any subproblem on all levels except for the top one, and even that only if the global optimum has already been found. For example, the solution 111111000 has the same fitness as the solution 000111000, although the latter solution is much farther from the global optimum in 111111111.

- For each subproblem on any level, 000 is easier to find than 111 because it is surrounded by high-quality solutions. Therefore, we can expect that the number of ones decreases as the search goes up a number of levels.

- Because of the character of traps, small, local changes in non-optimal solutions lead *away from the optimum* in most cases.

The hierarchical trap f_{htrap2} also uses the trap with $f_{high} = 1$ and $f_{low} = 0.9$ on the top level. However, f_{htrap2} makes the bias toward the solutions containing many zeroes even stronger by making the peak f_{low} higher than the peak f_{high} on all levels except for the top one. In this case, the optimum in 111 is not only isolated but also local. To ensure that the optimum still remains in the string of all ones, values of f_{low} and f_{high} must satisfy the following inequality:

$$(l-1)\,(f_{low} - f_{high}) < 0.1, \tag{5.1}$$

where l is the total number of levels. The last equation must be satisfied so that the difference between the peaks on the top level is greater than the sum of the differences on the remaining levels. We set $f_{high} = 1$ and $f_{low} = 1 + 0.1/k$ on all levels except for the top one. Similarly as in the previous case, the contributions on each level are multiplied by 3^{level}. There is one additional difficulty of searching f_{htrap2} compared to f_{htrap1}:

- If the algorithm does not preserve 111 in spite of that 000 performs slightly better, the optimum cannot be found. Recall the example presented earlier in this section. For f_{htrap2}, despite that 111111000 is much closer to the global optimum in 111111111 than 000111000 is, the fitness of 000111000 is greater than the fitness of 111111000 (while in f_{htrap1} the fitness is the same for both solutions).

Hierarchical traps presented above differ from the previously proposed hierarchical functions in several aspects. The path to the global optimum in hierarchical traps is much windier than the one in the royal roads. Not only are the solutions with many zeroes easier to find, but if these are disrupted the search must go down one or more levels. Unlike HIFF, hierarchical traps do not allow the algorithm to decide whether to go toward zeroes or ones; the global optimum is in the string of all ones, although for all levels except for the top one it is either impossible to distinguish between zeroes and ones, or zeroes look even better than ones. The bias toward the solution of all zeroes distinguishes hierarchical traps from HIFF and introduces a bigger challenge. Hierarchical traps and tobacco roads share many similarities. However, hierarchical traps contain more levels than tobacco roads do and by using low-order subproblems they allow an easier scalability analysis.

5.4 Summary

This chapter posed the challenge of solving problems that are not decomposable into subproblems of bounded order, but can be decomposed over a number of levels of difficulty into a hierarchy. The chapter identified three important features for scalable optimization of difficult hierarchical problems. Finally, the chapter designed a class of difficult hierarchical problems called hierarchical traps. A summary of the key points of this chapter follows:

- Hierarchy is a core component of human understanding of complex systems. Even more importantly, hierarchy is a powerful tool for reducing problem complexity in human problem solving.

- Hierarchy can also be used as a tool for reducing the complexity in computational optimization. Similarly as in human problem solving, hierarchy extends the class of problems that can be solved quickly, accurately, and reliably; it can tackle problems that are not decomposable into tractable subproblems on a single level but can be solved by decomposing the problems down a number of levels of difficulty.

- There are three important issues we must consider in the design of an optimizer capable of exploiting a hierarchical decomposition:

    1. Proper decomposition.
    2. Chunking.
    3. Preservation of alternative candidate solutions.

- The discovery and utilization of a proper decomposition were tackled in the design of the Bayesian optimization algorithm and can be adopted at no extra cost. However, BOA must be extended to incorporate the chunking and the preservation of alternative candidate solutions to solve difficult hierarchical problems in a scalable manner.

- In addition to extending BOA to a scalable hierarchical problem solver, it is important to design a class of problems that challenge the algorithm and bound the class of problems that the algorithm can solve. A new test bed can be used as a crash test for the solvers that attempt to solve the class of hierarchical problems and anything easier.

- Hierarchical traps extend the problems containing hierarchy from the past and create a novel challenge for the problem solvers and optimizers. While composed traps can be used to test the algorithms on their capability of solving decomposable problems, hierarchical traps can be used to test optimizers on their ability to solve complex hierarchical problems.

Chapter 6

Hierarchical Bayesian Optimization Algorithm

The previous chapter has discussed how hierarchy can be used to reduce the complexity of the problem at hand. Additionally, the chapter has identified the three important components that must be incorporated into BOA to make it capable of solving difficult hierarchical problems in a scalable manner and proposed a number of artificial problems that can be used to test the scalability of optimization methods that attempt to exploit hierarchy.

The purpose of this chapter is to extend BOA to solve difficult hierarchical problems quickly, accurately, and reliably. The chapter discusses several such extensions and implements one of the extensions to form the basis of hierarchical BOA (hBOA). Hierarchical BOA is then tested on the challenging hierarchical problems presented in the previous chapter.

The chapter starts by discussing the first two keys to hierarchy success—a proper decomposition and chunking. Section 6.1 describes one of the approaches to tackle both keys at once; in particular, the section incorporates local structures into Bayesian networks to allow a compact representation of the local conditional distributions for each variable. Section 6.2 focuses on the last key to hierarchy success—the preservation of alternative solutions—and reviews approaches to maintaining useful diversity in genetic and evolutionary computation. The section classifies presented diversity-maintenance techniques into several categories and discusses them in the context of BOA. Section 6.3 summarizes the extensions of the original BOA that comprise hierarchical BOA. Section 6.4 analyzes the performance of hierarchical BOA on several hierarchical problems, including hierarchical traps and HIFF. Section 6.5 relates the performance of hierarchical BOA with the BOA scalability theory presented in Chapter 4. Finally, the chapter is summarized.

6.1 Proper Decomposition and Chunking

Chapters 3 and 4 have shown that BOA is capable of learning and utilizing a proper decomposition of problems that are decomposable into subproblems of a bounded order on a single level. Learning and utilization of a proper decomposition is ensured by (1) constructing a Bayesian network that captures important nonlinearities in the set of promising solutions and (2) sampling the learned network to generate new candidate solutions. Bayesian networks can also be used to encode, learn, and utilize the decomposition of hierarchical problems. However, additionally to representing the dependencies and independencies in the problem on the current level of the search, a graphical model must incorporate some form of chunking as described in Section 5.2.

The purpose of this section is to review alternative approaches to incorporating chunking into BOA without compromising BOA's capability of decomposing the problem properly. Since the issues of chunking and learning a proper decomposition are strongly correlated, the two issues are approached together.

The section first discusses different facets of chunking in detail. The section then describes how local structures—such as default tables, decision trees, and decision graphs—can be used to enhance Bayesian networks and, in turn, ensure chunking.

6.1.1 Chunking Revisited

The primary goal of chunking is to allow groups of variables from each subproblem of the lower level to be merged into a single variable (or an intact block) that encodes all the relevant information needed to distinguish alternative partial solutions to the particular subproblem. These merged variables serve as the basic building blocks for composing solutions on the next level. Note that there are two problems that must be considered:

1. **Merging.** The model must be capable of merging a group of variables corresponding to each subproblem from the lower level into a single variable or a block. This can be done either explicitly or implicitly, but the model must allow such merging to take place.

2. **Representing partial solutions efficiently.** The model must represent partial solutions compactly so that only relevant information is considered. A compact representation is necessary for ensuring that large partial solutions can be encoded in the model (either as an intact block or a new, single, variable).

Merging the variables into groups can be incorporated by creating specialized models that can encode dependencies and independencies among the groups of variables as opposed to dependencies and independencies among the variables themselves. Nonetheless, since Bayesian networks have been shown to be capable of representing the chunks of solutions of decomposable problems in BOA, the issue of merging the variables can be taken care of using traditional Bayesian networks. But in either case, an efficient representation of the relevant features of each chunk must be ensured. To summarize, there are two approaches to incorporating chunking into BOA:

1. **Explicit chunking.** Modify the model so that partitions of the problem are allowed to group into a single component of the model. The relationships between the chunks can then be represented by Bayesian networks as in BOA or in some other way. Example models that allow explicit chunking are the marginal product models (MPM) of ECGA (Harik, 1999), Huffman networks that combine MPM with Bayesian networks (Davies & Moore, 1999), and Bayesian networks with hidden variables (Cooper & Herskovits, 1992; Geiger, Heckerman, & Meek, 1996). Of course, the models must represent their parameters efficiently so that large groups of variables can be processed.

2. **Implicit chunking.** Ensure that the used model is capable of representing the chunks, although the overall structure might not correspond to an intuitive notion of chunking based on merging the variables into groups explicitly. Additionally, the models must represent their parameters efficiently so that large chunks can be represented. An example approach from this class are Bayesian networks with local structures, which are going to be discussed in the following section.

Since in both approaches, a compact representation of the models is necessary and Bayesian networks can encode the chunks at no extra cost (if a compact representation is used), the remainder of this section focuses on implicit approaches to chunking. In particular, the section describes local structures that can be used to ensure that the dependencies of high order can be encoded by the model.

6.1.2 Local Structures in Bayesian Networks

The number of conditional probabilities that must be specified in the Bayesian network grows exponentially with the order of interactions encoded by the network. Although in some cases, simple models suffice and there is no need for a compact representation, the exponential growth of the number of parameters of the model can become a bottleneck on more complex problems.

This section describes how local structures can be used to make the representation of the conditional distributions in Bayesian networks more compact and to allow the models to include high-order interactions. Additionally to providing a technique for chunking, local structures allow learning and sampling more complex models with a reasonable number of parameters.

The section starts by providing an example conditional probability table that motivates the use of local structures. Next, the section describes default tables that specify only a subset of the conditional probabilities and set the remaining probabilities to a specified constant. Finally, the section introduces decision trees and graphs that reduce the number of parameters in a more general fashion.

Motivating Example

Let us start with a simple example that motivates the use of default tables and other local structures in Bayesian networks. Assume that a binary variable X_1 in the Bayesian network learned by BOA is conditioned on 3 other binary variables denoted by X_2, X_3, and X_4. To encode the conditional probabilities of the different values of a binary variable conditioned on k other variables, the model must specify 2^k conditional probabilities. Note that there are 2^{k+1} conditional probabilities that must be specified but half of them can be computed by using the following equation:

$$p(X_i = 0|\Pi_i) = 1 - p(X_i = 1|\Pi_i), \tag{6.1}$$

where X_i denotes the considered variable and Π_i denotes the set of X_i's parents. In our example, the probabilities are set as shown in Table 6.1.

X_2	X_3	X_4	$p(X_1 = 0 \mid X_2, X_3, X_4)$
0	0	0	0.75
0	0	1	0.25
0	1	0	0.25
0	1	1	0.25
1	0	0	0.20
1	0	1	0.20
1	1	0	0.20
1	1	1	0.20

Table 6.1: Conditional probability table for the variable X_1 that is conditioned on the variables X_2, X_3, and X_4. Only the probabilities of one of the values of X_1 must be stored, because the remaining probabilities can be computed using $p(X_1 = 1|X_2, X_3, X_4) = 1 - p(X_1 = 0|X_2, X_3, X_4)$.

Although three of the probabilities shown in Table 6.1 are 0.25 and four of them are 0.20, the conditional probability table lists each single probability. Listing all the probabilities is not a big deal for the dependencies of order three, but it will become a major bottleneck if we want to represent the dependencies of order 50, for instance. Local structures allow the use of regularities in the conditional probability tables and encode the probabilities in a more compact way.

The remainder of this section describes three types of local structures—default tables, decision trees, and decision graphs—that can be used to encode the conditional probabilities in a Bayesian network. Of course, there are many other ways to approach the same problem, but the approaches presented in this section should provide a sufficiently powerful means for ensuring a compact representation of Bayesian networks. The aforementioned conditional probability table will be used throughout the section to illustrate the effectiveness of the different types of local structures and their basic principle.

6.1.3 Default Tables

A default table lists several probabilities with the corresponding instances of the variables in the condition first, and sets the remaining probabilities to a default value. The default value is computed as an average of all the covered conditional probabilities weighted by the probabilities of the conditional parts of those probabilities. If a particular conditional probability is listed, its value is determined from the corresponding entry in the table. If the conditional probability is not listed, its value is assumed to be equal to the default value.

X_2	X_3	X_4	$p(X_1 = 0 \mid X_2, X_3, X_4)$
0	0	0	0.75
0	0	1	0.25
0	1	0	0.25
0	1	1	0.25
default			0.20

Table 6.2: A default table reducing the number of conditional probabilities from Table 6.1 by 3 without compromising the accuracy of the model.

X_2	X_3	X_4	$p(X_1 = 0 \mid X_2, X_3, X_4)$
0	0	0	0.75
default			0.23

Table 6.3: A default table reducing the number of conditional probabilities by 6. In this case, the conditional probabilities of 0.20 and 0.25 are both stored in the default row of the table, yielding a different assignment of the probabilities than the one observed in the data. However, the default table may still lead to increased likelihood of the overall model.

There are two intuitive ways to compress the conditional probability table from Table 6.1 using default tables. The first alternative is to list the first four probabilities and set the remaining probabilities to the default value of 0.20. The second alternative is to list the first probability and the last four ones first, and set the remaining probabilities to the default value of 0.25. In the first case, more entries in the table are eliminated and therefore higher compression is achieved. The resulting default table is shown in Table 6.2.

The default table presented above reduces the number of stored probabilities without losing any information at all. However, it is possible to group the probabilities that are not equal but that are very similar. For example, the conditional probability table shown in Table 6.1 can be further reduced as shown in Table 6.3 by merging the probabilities 0.20 and 0.25 into a single default value. Although such an assignment results in that the probabilities encoded in the network are different than those observed in the data, the assignment might still lead to an increase in the likelihood (quality) of the model[1]. In that case, the introduced inaccuracy in the probabilities pays off in the end.

There is an important difference between the conditional probability tables and the default

[1]Recall that both the Bayesian metrics and the MDL metrics include some form of penalty for model complexity. In both cases, the penalty relates to the number of parameters of the model in some way. Therefore, despite that the model that merges unequal probabilities together will reflect the actual distribution of the data less accurately, the complexity-penalty term can decrease by a higher amount.

tables. In a conditional probability table, it is sufficient to list 2^k numbers to encode the conditional probabilities of a variable with k binary variables in the condition. Assuming a fixed ordering of the probabilities, it is sufficient to look at the position of each number to find out which probability the number actually represents. For example, Table 6.1 orders the probabilities alphabetically according to the values of the variables in the condition; the first number in this ordering represents the probability with the condition 000, the second number in this ordering represents the probability with the condition 001, and so forth. On the other hand, the default table must specify the instance that the listed probability corresponds to, because it is not known in advance which of the probabilities are specified and which ones are not. Clearly, there are several tricks to make the representation more efficient, but it is beyond the scope of this section to discuss these.

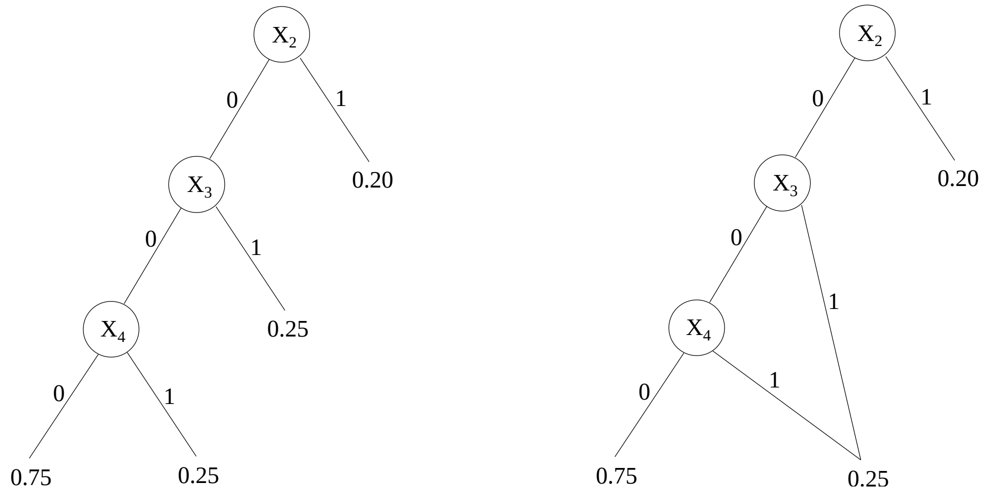
The limitations of default tables are clear; a default table is capable of encoding the similarities among the members of only *one* subset of probabilities. That is why one must choose whether to group the probabilities 0.20 or 0.25, but it is impossible to group all the instances with the probability 0.20 and, at the same time, group all the instances with the probability 0.25. Next, we describe how more sophisticated structures—such as decision trees and decision graphs—can be used to exploit more regularities in the set of conditional probabilities for the considered variable.

6.1.4 Decision Trees and Graphs

A decision tree is a directed acyclic graph where each node except for the root has exactly one parent; the root has no parents. The internal nodes (all nodes except for the leaves) of the tree are labeled by a variable (feature). When a node is labeled by a variable v, we say that the node is a split on v. Edges from a split on v are labeled by non-empty distinct exhaustive subsets of possible values of v.

Given an assignment of all the variables, a traversal of the decision tree starts in the root. On each split on v, the traversal continues to the child along the edge containing the current value of v. For each assignment of the involved variables, there is only one possible way of traversing the tree to a leaf, because the edges coming from each split must be labeled by distinct subsets of the values.

Each leaf of a decision tree contains a quantity or information of interest associated with all the



(a) Decision tree. (b) Decision graph.

Figure 6.1: The decision tree and the decision graph that encode the probabilities shown in Table 6.1.

instances that end up the traversal of the tree in that leaf. To use decision trees for representing the conditional probabilities of a particular variable, each leaf stores the conditional probabilities of the variable given that the variables contained in the path from the root to the leaf are fixed according to the path.

Let us return to the example from Table 6.1 discussed in the context of default tables. To represent the full conditional distribution, it is sufficient to construct the tree shown in Figure 6.1(a). The decision tree reduces the number of conditional probabilities that must be stored from 8 to only 4.

In some cases, it might be useful to store less probabilities at the expense of the accuracy of the resulting model. Analogously to the case with default tables, the reduction of the number of parameters of the model might result in increasing the overall quality of the model.

Decision Graphs

Note that in the decision tree from Figure 6.1(a), there are two leaves that store the same conditional probability of 0.25. If the two leaves could be merged into a single one, the number of required probabilities could be further reduced by 1. However, after merging the two leaves, the new node would have two parents instead of one; consequently, the resulting graph would not be a tree. In spite of that, the resulting structure would be still useful in the same way.

Decision graphs extend decision trees by allowing each node to have multiple parents and therefore can find a use for more regularities in the conditional probabilities associated with the Bayesian network. For example, the decision graph encoding the probabilities from the aforementioned example is shown in Figure 6.1(b). Using a decision graph reduces the number of conditional probabilities that must be stored by more than a half (only 3 out of 8 probabilities must be stored).

Parsing a decision graph is analogous to parsing a decision tree. For each instance, there is exactly one way of traversing the graph down to a particular leaf, which stores the corresponding probability. Unlike decision trees, decision graphs allow encoding any equality constraints on the conditional probabilities. In other words, there can be as few leaves in the decision graph as there are different conditional probabilities. By introducing additional equality constraints, the overall likelihood of the model can be further increased.

Bayesian Network with Decision Graphs

A Bayesian network with decision graphs contains one decision graph for each variable. The decision graph for X_i is denoted by G_i. Each decision graph encodes the conditional probabilities for the corresponding variable.

Note that although the decision graphs encode the conditional probabilities for a particular network structure, the network structure can be constructed from the decision graphs themselves. The set of parents of each variable is the set of variables on which there exists a split in the decision graph corresponding to the considered variable. As long as the decision graphs contain valid probabilities and there are no cycles in the corresponding Bayesian network, the network is fully specified by the set of decision graphs, one graph per variable.

As an example, consider the decision graph shown in Figure 6.1(b), which stores the conditional probabilities of X_1 for the considered Bayesian network. The decision graph reveals that the probability distribution of X_1 is affected by the variables X_2, X_3, and X_4, and it is independent of any other variable. Therefore, X_2, X_3, and X_4 are the parents of X_1 in the underlying network structure.

There are three major advantages of using decision graphs in learning Bayesian networks:

(1) **Parameter compression.** Fewer conditional probabilities can be used to represent the model. This saves memory and time for both the model construction as well as the utilization of the constructed model.

(2) **More complex models.** Decision graphs allow learning a more complex class of models. Although all the models can be represented by the full joint probability distribution represented by a fully connected Bayesian network, the full joint probability distribution requires an exponential number of parameters. Decision graphs reduce that number of parameters so that learning the model becomes tractable even when there are no conditional independencies in the model.

(3) **Better learning.** The construction of a Bayesian network with decision graphs can proceed in smaller and more specific steps. That often results in better models with respect to their likelihood (Chickering, Heckerman, & Meek, 1997; Friedman & Goldszmidt, 1999).

Using decision graphs improves the utility of Bayesian networks, but the semantics of Bayesian networks with decision graphs remains the same. Therefore, one could expect that although the expressiveness of Bayesian networks will improve by using decision graphs, the methods for measuring the quality of candidate models and the methods for constructing a model given data should not change much. Indeed, the scoring metrics for measuring the quality of each candidate model as well as the greedy algorithm for constructing the model can be adapted to the new situation in a straightforward manner.

In the following, we first describe the changes that must be incorporated into the two metrics presented earlier in this thesis—the Bayesian-Dirichlet metric (BD) and the Bayesian information criterion (BIC). Subsequently, we define the split and merge operators, which can be used to construct decision graphs. Finally, we present an algorithm for the construction of Bayesian networks with decision graphs. The presented algorithm exploits the new representation of Bayesian networks by directly manipulating decision graphs instead of updating the decision graphs as a consequence of changing the network itself.

Bayesian Score for Networks with Decision Graphs

Chickering, Heckerman, and Meek (1997) derived the Bayesian score for Bayesian networks with decision graphs. The resulting metric can be computed analogously to the case with traditional Bayesian networks, yielding

$$p(D|B) = \prod_{i=0}^{n-1} \prod_{l \in L_i} \frac{\Gamma(m_i'(l))}{\Gamma(m_i(l) + m_i'(l))} \prod_{x_i} \frac{\Gamma(m_i(x_i, l) + m_i'(x_i, l))}{\Gamma(m_i'(x_i, l))}, \tag{6.2}$$

where L_i is the set of leaves in the decision graph G_i for X_i; $m_i(l)$ is the number of instances in D which end up the traversal through the graph G_i in the leaf l; $m_i(x_i, l)$ is the number of instances that have $X_i = x_i$ and end up the traversal of the graph G_i in the leaf l; $m_i'(l)$ represents the prior knowledge about the value of $m_i(i, l)$; and $m_i'(x_i, l)$ represents the prior knowledge about the value of $m_i(x_i, l)$. Again, the uninformative prior $m_i'(x_i, l) = 1$ is used in the K2 variant of the BD metric for Bayesian networks with decision graphs.

As mentioned earlier in the thesis, the Bayesian metrics tend to be more sensitive to the noise in data and, in practice, they often lead to overly complex models. However, Bayesian metrics allow the use of prior information, which can bias the metric to favor simpler models and eliminate the aforementioned problem. Although in traditional Bayesian networks we have not succeeded in finding a robust prior that would work well in general, in Bayesian networks with decision graphs we have resolved this problem. This robust prior is described next.

To adjust the prior probability of each network according to its complexity, the description length of the parameters required by the network is first computed. One frequency in the data set of size N can be encoded using $\log_2 N$ bits; however, only half of the bits suffice to encode the frequencies with a sufficient accuracy (Friedman & Yakhini, 1996). Therefore, to encode all the parameters, $0.5(\sum_i |L_i|) \log_2 N$ bits are needed, where $\sum_i |L_i|$ is the total number of leaves in all the decision graphs. To favor simpler networks to the more complex ones, the prior probability of a network can decrease exponentially with the description length of the set of parameters they require (Friedman & Goldszmidt, 1999). Thus,

$$p(B) = c 2^{-0.5(\sum_i |L_i|) \log_2 N}, \tag{6.3}$$

where c is a normalization constant required for the prior probabilities of all the network structures to sum to 1. The value of a normalization constant does not affect the result, because we are interested in only relative quality of networks and not the absolute value of their marginal likelihood.

From our experience, the above prior is sufficient to bias the model construction to networks with less parameters and avoid superfluously complex network structures. For local structures in Bayesian networks, the K2 score with the above prior actually seems to outperform the BIC metric in its robustness.

A similar prior was proposed by Chickering, Heckerman, and Meek (1997), who decrease the prior probability of each network exponentially with the number of parameters of the network:

$$p(B) = c\kappa^{\sum_i |L_i|},$$

where $\kappa \in (0, 1)$. Usually, κ is rather small, for instance, $\kappa = 0.1$.

BIC for Bayesian Networks with Decision Graphs

The Bayesian information criterion (BIC) for Bayesian networks with decision graphs can also be computed analogously to the case with traditional Bayesian networks (see Equation 3.4). The difference is that instead of summing over the instances of each variable and its parents, the sums must go over all the leaves in the tree. Additionally, the number of parameters is not anymore proportional to the number of instances of the parents of all the variables, but it is equal to the number of leaves. Thus,

$$BIC(B) = \sum_{i=1}^{n} \left(N \sum_{l \in L_i} m_i(x_i, l) \log_2 \frac{m_i(x_i, l)}{m_i(l)} - |L_i| \frac{\log_2(N)}{2} \right). \tag{6.4}$$

Decision Graph Construction: Operators on Decision Graphs

To construct a decision graph for binary variables, two operators are sufficient. The first operator is a *split*, which splits a leaf on some variable and creates two new children of the leaf, connecting each of them with an edge associated with one possible value of this variable (0 or 1). The second operator is a *merge*, which merges two leaves into a single one and introduces a new equality

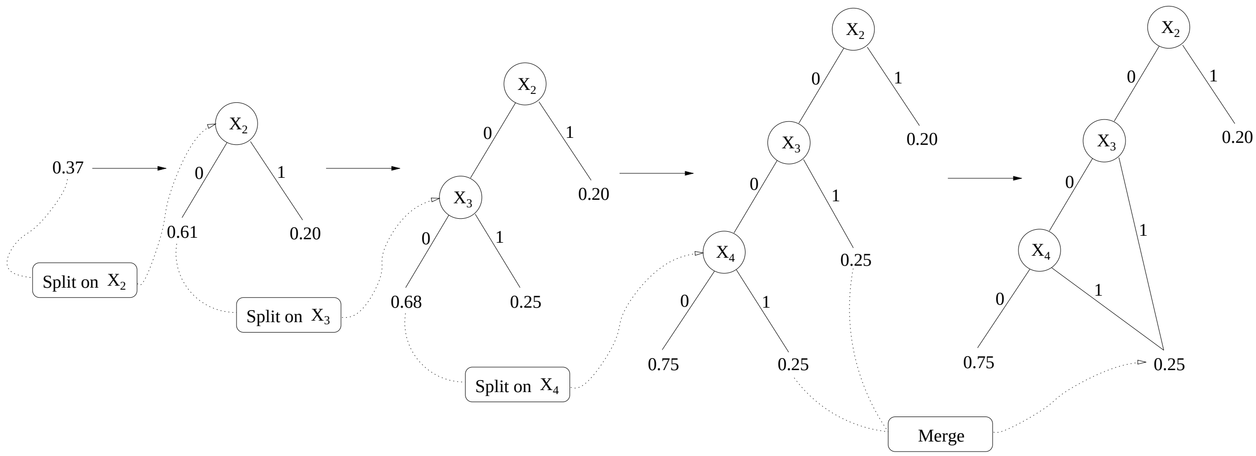


Figure 6.2: The *merge* and *split* operators for decision graphs. The decision graph shown earlier in the text is constructed from an empty decision graph consisting of one leaf by a sequence of splits and merges.

constraint on the parameter set.

Figure 6.2 shows the operators that can be used to construct the decision graph shown in Figure 6.1(b) from an empty decision graph that contains only the marginal probability of the considered variable without any condition. A sequence of three splits and one merge leads to the resulting graph shown earlier in this chapter.

For the variables that can obtain more than two values, two versions of the split operator can be considered: (1) a complete split, which creates one child for each possible value of the variable (as above), and (2) a binary split, which creates one child corresponding to one particular value of the variable and another child for all the remaining values. These two operators are equivalent for binary variables. Of course, other alternatives can also be considered.

Constructing Bayesian Networks with Decision Graphs

The greedy algorithm for constructing the Bayesian networks with decision graphs described in this section differs from the greedy algorithm for constructing traditional Bayesian networks presented in Section 3.3.2. The difference is that the algorithm for constructing the networks with decision graphs does not directly manipulate the network but it modifies the decision graphs corresponding to the variables in the network. The network B is initialized to an empty network that contains no edges. The decision graph G_i for each variable X_i is initialized to a single-leaf graph, containing

A greedy algorithm for network construction using decision graphs

(1) Initialize a decision graph G_i for each node X_i to a graph containing only a single leaf.

(2) Initialize the network B into an empty network.

(3) Choose the best split or merge that does not result in a cycle in B.

(4) If the best operator does not improve the score, finish.

(5) Execute the chosen operator.

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

(7) Go to (3).

Figure 6.3: The pseudo-code of the greedy algorithm for constructing the Bayesian network with decision graphs.

only probabilities $p(X_i)$ as shown earlier in Figure 6.2.

Each iteration, all operators (e.g., all possible merges and splits) that can be performed on all decision graphs G_i are examined. The operator that improves the score the most is performed on the corresponding decision graph. The operators that can be performed include (1) splitting a leaf of some decision graph on a variable that was not encountered on the path from the root to the leaf and (2) merging two leaves into a single leaf.

When performing a split operator, no cycles must be introduced to the network B. To guarantee that the final network remains acyclic, the network B can be updated after each split. After splitting a leaf of the graph G_i on X_j, an edge $X_j \rightarrow X_i$ is added to the network B. If a cycle would be introduced by the new edge, the split must not be allowed. The above restriction could be alleviated. The use of decision trees allows Bayesian multinets (Geiger & Heckerman, 1996) with one or more distinguished variables. However, for the sake of computational efficiency and the simplicity of the implementation, we do not take this under consideration.

It is important to notice the difference between the algorithm that directly modifies the network and the one that modifies the decision graphs. Adding an edge into a Bayesian network and using a full conditional probability table to store the required probabilities corresponds to splitting all leaves of the decision graph corresponding to the terminal node of the edge on the variable corresponding to the initial node of the edge. However, by modifying only the decision graph, finer steps can be performed and that might improve the quality of the resulting model.

The following section focuses on the last key to hierarchy success—the preservation of alternative solutions.

6.2 Preservation of Alternative Candidate Solutions

This section focuses on methods that attempt to maintain useful diversity in genetic and evolutionary algorithms. The preservation of alternatives is provided by encouraging local competition for the space in the population in some way; similar solutions compete with each other, whereas dissimilar solutions compete only rarely or never. Using an analogy from ecological systems, solutions that are similar share the same niche, and methods for preserving diversity are thus often referred to as *niching methods*.

The purpose of this section is to propose a niching method that will ensure the third key to hierarchy success—the preservation of alternative solutions—in hierarchical BOA. The section starts with an overview of niching methods. Methods that have been successfully applied in the domain of traditional GAs are discussed in the context of BOA. One of the methods is then adopted to ensure the preservation of alternative candidate solutions in hierarchical BOA.

6.2.1 Background of Niching

Niching methods localize competition in genetic and evolutionary algorithms so that different groups of candidate solutions compete in different niches. Consequently, diversity can be maintained for a long time. The purpose of niching in genetic and evolutionary optimization is twofold:

(1) **Generate multiple solutions.** In some real-world applications it is important to find multiple solutions and let the expert or experiment decide which of the solutions is the best after all. This is usually the case when the fitness function does not fully determine which solution is the best in practice but only focuses on several aspects of solution quality, or when for the sake of efficiency instead of using a complete fitness function one uses only its approximation that is more computationally efficient.

(2) **Preserve alternative solutions.** The reason for preserving multiple alternative solutions is that in some difficult problems one cannot clearly determine which alternative solutions are really on the right track until the optimization proceeds for a number of generations. Without effective niching, the population is subject to genetic drift (random effects, see Section 4.2.3) which may destroy some alternatives before it becomes clear whether or not they are the ones we are looking for.

There are three fundamentally different approaches to niching:

(1) crowding,

(2) fitness sharing, and

(3) spatial separation.

Crowding modifies the selection procedure to localize competition by taking into account both the fitness as well as the genotype (solution string) or phenotype (semantics of the solution string) of competing candidate solutions (Cavicchio, 1970; De Jong, 1975; Harik, 1994; Mahfoud, 1992; Mengshoel & Goldberg, 1999). *Fitness sharing* modifies the fitness landscape before selection so that the number of solutions corresponding to each niche is proportional to the fitness of that niche (Goldberg & Richardson, 1987; Deb & Goldberg, 1989; Horn, 1993). *Spatial separation* isolates several subpopulations of candidate solutions instead of keeping the entire population in one location (Grosso, 1985; Cohoon, Hegde, Martin, & Richards, 1987; Gorges-Schleuter, 1989; Collins & Jefferson, 1991; Davidor, 1991; Mühlenbein, 1991). Candidate solutions can migrate between different locations (islands or demes) at certain intervals and allow the population at each location develop in isolation. Since competition among candidate solutions in each island is separated from other islands, solutions in different islands can differ significantly and diversity can be preserved for a long time. However, spatial separation does not force niching, it only *allows* niching to take place. That is why spatial separation is sometimes omitted in the context of niching.

Some related work studies the preservation of diversity from a different point of view. The primary goal of these techniques is not the preservation of multiple solutions or alternative search regions, but the avoidance of *premature convergence*. Mauldin (1984) proposed one such approach, which injects randomly generated candidate solutions into the current population at certain intervals. On the other hand, Baker (1985) proposed a method that controls selection to prevent premature convergence. However, the avoidance of premature convergence is not the primary purpose of using niching in our work (although the preservation of alternatives and premature convergence are related). Various techniques for niching were also proposed in the area of multi-objective optimization (Schaffer, 1984; Horn & Nafpliotis, 1993; Fonseca & Fleming, 1993; Deb, 2001), but these methods are not applicable to single-criterion optimization.

The section continues with a brief overview of existing techniques in the three classes of niching methods: (1) crowding, (2) fitness-sharing, and (3) spatial separation models. Subsequently, the section describes restricted tournament replacement used in hierarchical BOA.

Niching by Crowding

All niching methods localize competition in some way. This section reviews methods that localize competition by modifying selection and replacement strategies.

Preselection (Cavicchio, 1970), deterministic crowding (Mahfoud, 1992), probabilistic crowding (Mengshoel & Goldberg, 1999), and the gene invariant genetic algorithm (GIGA) (Culberson, 1992) are applicable to GAs with two-parent recombination methods. In preselection (Cavicchio, 1970), one candidate solution is created by crossing over two parent solutions, and the new solution replaces the inferior parent. Preselection encourages competition among similar candidate solutions, because an offspring and its parents usually share many similarities. Deterministic crowding (Mahfoud, 1992) pairs each offspring with a more similar parent and the solutions within each pair compete. The offspring replaces the parent if it is better; otherwise the offspring is discarded. Probabilistic crowding (Mengshoel & Goldberg, 1999) is a probabilistic extension of Mahfoud's deterministic crowding. In probabilistic crowding, the winner of the parent-offspring tournament is chosen according to the probabilities proportional to the fitness of the competitors. The gene invariant genetic algorithm (GIGA) (Culberson, 1992) starts by selecting two solutions with fitness proportionate selection. After crossing over the selected solutions, the offspring replace their parents. In this fashion, the proportion of 1s and 0s in each position is preserved (gene invariance).

In crowding (De Jong, 1975), for each new candidate solution a subset of the original population is first selected. The new solution then replaces the most similar candidate solution in the selected subset. Earlier in the run only little will change compared to random replacement. However, as the run continues, candidate solutions will create groups of similar solutions, which compete for space with other members of the same group. Crowding was later used by Goldberg (1983) for a pipeline design by learning classifier systems.

Harik (1994) proposed restricted tournament selection (RTS) as an extension of De Jong's crowding. Restricted tournament selection selects two parents at random with uniform distribution. Two candidate solutions are then generated by applying crossover to the two selected parents. For each new candidate solution, a subset of the original population is first selected as in crowding. However, instead of automatically replacing the closest solution from the selected subset, the two solutions compete and the one that has a higher fitness wins. In this fashion, the selection step is performed by elitist replacement with a flavor similar to that of crowding. No extra selection operator is required. Harik showed that RTS performed well on a number of multimodal problems and that it was able to locate all optima even on the functions that are highly multimodal and difficult to solve. We will support Harik's argument by a similar experiment presented at the end of this chapter.

A number of techniques that *restrict mating* in some way to promote niching were proposed. Hollstein (1971) required similar candidate solutions to mate as long as their fitness improved. When the trend changes and the quality of the "family" decreases, crossbreeding across the families is allowed. Booker (1982) discussed the need for restricted mating to prevent the formation of lethals. Perry (1984) introduced multiple contexts in which the fitness function varied according to externally specified partial solutions that defined species. Candidate solutions could migrate between different contexts. The technique of Perry is mainly interesting for its biological background.

In the context of discrete PMBGAs, Pelikan and Goldberg (2000a) divided the population of selected parents in each generation into a specified number of clusters. A mixture of Gaussians was used to separate candidate solutions in the selected population by using k-means clustering. Each cluster was processed separately and the offspring of each cluster was given a fraction of the new population of the size proportional to their average fitness.

De Jong's crowding and Harik's restricted tournament selection are two most straightforward approaches to incorporating niching into BOA. Both methods can be incorporated without affecting the model building or its utilization in BOA. However, it becomes difficult or impossible to use other crowding methods, such as Mahfoud's deterministic crowding, Mengshoel's probabilistic crowding, and Culberson's GIGA.

Niching by Fitness Sharing

The basic idea of fitness sharing is to create a number of artificial niches and give each niche a number of copies proportional to the quality of solutions in the niche. Although the motivation for fitness sharing dates back to Holland (1975), the first practical fitness-sharing method was developed by Goldberg and Richardson (1987). Promising solutions are selected with the probability proportional to their fitness. However, before selection, the fitness of each candidate solution is modified according to

$$f'(X) = \frac{f(X)}{\sum_Y sh(d(X,Y))},$$

where $\sum_Y$ runs over all candidate solutions Y in the current population; $d(X,Y)$ is the distance between solutions X and Y; and $sh(d)$ is the sharing function. The sharing function defines the degree of similarity of the two solutions that are located at distance d from each other as follows:

$$sh(d) = \begin{cases} 1 - \frac{d}{\sigma_{share}} & \text{if } d < \sigma_{share} \\ 0 & \text{otherwise} \end{cases},$$

where σ_{share} is the sharing threshold that defines the maximum distance between two candidate solutions that share a niche. After updating the fitness as described above, the number of copies of solutions in each niche is proportional to the average fitness of the niche.

For a successful application of fitness sharing, it is necessary to determine a proper value of the sharing threshold σ_{share}. Deb and Goldberg (1989) calculated the value of σ_{share} from the desired number of niches by dividing the problem domain (set of all candidate solutions) into a specified number of equally sized hyperspheres.

One of the drawbacks of fitness sharing is that it experiences difficulty with maintaining optima that are close to each other or those that are distributed irregularly. On the other hand, fitness sharing is capable of preserving all the optima for long periods of time (see for example the study of Horn (1993) who analyzed the stability of sharing for the case with two niches). Furthermore, unlike the approaches based on crowding, the fitness proportionate selection is capable of maintaining the size of each niche so that the number of copies of solutions in each niche is proportional to the average fitness of that niche.

Fitness sharing directly modifies the fitness before selection takes place. Consequently, the frequencies of partial solutions after selection are "disturbed" by sharing. Since those frequencies are the only input of BOA regarding the nonlinearities in the problem, BOA's capability of building a good model might suffer. That is why fitness sharing does not seem to be an appropriate approach to niching in BOA.

Niching by Spatial Separation

There are two reasons why spatial separation should be desirable in genetic and evolutionary computation: (1) In nature the populations are actually divided into a number of subpopulations that (genetically) interact only rarely or do not interact at all. (2) Separating a number of subpopulations allows for an effective parallel implementation and is therefore interesting from the point of view of computational efficiency. This section reviews and discusses niching methods based on spatial separation.

Spatial separation localizes competition by introducing a sort of geographical location of each candidate solution. Unlike in fitness sharing, in spatial separation the location of each solution does not depend on its genotype (the representation of the solution) or phenotype (the semantics of the solution). Amount of information exchange between the groups of candidate solutions from different locations is controlled by a specific strategy, which might depend on the distance or the relationship between the locations.

Much work in spatial separation was inspired by the shifting balance theory (Wright, 1968) and the theory of punctuated equilibria (Eldredge & Gould, 1972). One approach is to divide the population into a number of subpopulations. Each subpopulation evolves in its own "island" and candidate solutions migrate between the islands at a certain rate. In this way, genetic material (partial solutions) is exchanged within each of the subpopulations often while its flow to other subpopulations is reduced. This approach was studied by Grosso (1985), inspired mainly by the theory of Wright, and by Cohoon, Hegde, Martin, and Richards (1987), whose work is primarily inspired by the theory of Eldredge and Gould. Another approach introduces some kind of distance metric in the population and forces local competition and mating. This approach was studied by Gorges-Schleuter (1989), Collins and Jefferson (1991), Davidor (1991), Mühlenbein (1991), and others.

The primary drawback of spatial separation is that there is no direct mechanism that *forces* niching; instead, spatial separation *allows* for niching, because different islands can converge to different regions of the search space. Therefore, while spatial separation is a great approach for using multiprocessor architectures to process large populations in parallel, it doesn't seem to be a good approach for ensuring that useful diversity is maintained.

There are several additional problems with using spatial separation in BOA. Spatial separation requires isolated processing of a number of subpopulations. Although dividing the population might be beneficial for parallelization, it negatively affects the performance of BOA on single-processor machines, because the overall number of candidate solutions must be much larger to ensure that each subpopulation represents a large enough sample of promising solutions to find a good model. Furthermore, there must be many islands to ensure that useful diversity is maintained and (due to the independence of spatial separation on candidate solutions) one can never be sure that this indeed takes place.

6.2.2 The Method of Choice: Restricted Tournament Replacement (RTR)

The overview of niching methods discussed a number of methods that have been designed to ensure that useful diversity is maintained. Each niching method described in the above overview can be advantageous in some situations. So what niching method should be used in hierarchical BOA?

As follows from the discussion of each method, restricted tournament selection of Harik (1994) and crowding of De Jong (1975) seem to be the best choices. Hierarchical BOA uses restricted tournament selection, because this niching method introduces an additional source of selection pressure that has proven advantageous according to a number of empirical results. However, since restricted tournaments are used for replacement in a generational GA (as opposed to the steady-state GA) and the primary selection pressure comes from selection, we call the used method *restricted tournament replacement* (RTR).

The following section describes hierarchical BOA and discusses how hierarchical BOA differs from the original BOA described in Chapter 3.

6.3 Hierarchical BOA

Above we discussed the important mechanisms that must be incorporated into BOA to extend its applicability to difficult hierarchical problems. The purpose of this section is to describe hierarchical BOA (hBOA) (Pelikan & Goldberg, 2001) and discuss the ways in which hierarchical BOA differs from the original BOA.

Recall that in BOA, the population is updated for a number of iterations (generations), each consisting of four steps: (1) selection, (2) model building, (3) model sampling, and (4) replacement. Each generation of hBOA also consists of the above four steps. However, instead of using traditional Bayesian networks, hBOA uses Bayesian networks with decision graphs to ensure the first two keys to hierarchy success—a proper decomposition and chunking. Therefore, the steps (2) and (3) of the basic BOA must be modified to incorporate the decision graphs into the learning and sampling of Bayesian networks.

The second modification consists of using RTR as a replacement strategy. For each new candidate solution generated by sampling the learned model in step (3), a random subset of candidate solutions is first selected from the original population. The size of the selected subsets is fixed to some constant, called the *window size*, denoted by w. The new solution is then compared to each candidate solution in the selected subset and the fitness of the most similar candidate solution of the subset is compared to that of the new solution. If the new solution is better, it replaces the most similar solution of the subset; otherwise, the new solution is discarded.

From our experience, a good heuristic is to set the window size to the number of bits in the problem:

$$w = n. \tag{6.5}$$

The reason for setting $w = n$ is that the number of niches n_n that can be maintained in a population

Hierarchical Bayesian Optimization Algorithm (hBOA)

(1) set $t \leftarrow 0$
randomly generate initial population $P(0)$

(2) select a set of promising strings $S(t)$ from $P(t)$

(3) build the network B by constructing decision graphs G_i encoding cond. probabilities (see Figure 6.3)

(4) generate a set of new strings $O(t)$ according to the joint distribution encoded by B with dec. graphs G_i

(5) create a new population $P(t+1)$ using RTR to incorporate each member of $O(t)$ into $P(t)$.
set $t \leftarrow t+1$

(6) if the termination criteria are not met, go to (2)

Figure 6.4: The pseudo-code of the hierarchical Bayesian optimization algorithm.

of candidate solutions is equal to some fraction of the population size: $n_n = O(N)$. RTR with the window size w can maintain the number of niches that is equal to some fraction of the window size: $n_n = O(w)$. Since in BOA, the population is expected to grow approximately linearly with the size of the problem, $N = O(n)$, to maximize the number of niches that can be maintained by BOA without affecting the population sizing, $w = O(n)$.

The pseudo-code of hBOA is shown in Figure 6.4.

6.4 Experiments

In the previous chapter we have promised that if one incorporates chunking and niching into BOA, the resulting method would be capable of scalable optimization of difficult hierarchical problems. We have also designed a class of challenging hierarchical problems—hierarchical traps—that can be used to test whether an algorithm can solve hierarchical problems in a scalable manner.

The purpose of this section is to present experimental results of applying hierarchical BOA presented in the previous section to the challenging hierarchical problems presented in Chapter 5.

Additionally, the algorithm is tested on the HIFF function (Watson, Hornby, & Pollack, 1998) and the folded trap function. In all cases, hBOA passes the test and proves to be a scalable hierarchical optimizer.

The section starts by introducing the experimental methodology. Empirical results are then presented and discussed.

6.4.1 Methodology

For each problem instance, 30 independent runs are performed and hierarchical BOA is required to find the optimum in all the 30 runs. The performance of hierarchical BOA is measured by the average number of evaluations until the optimum is found. The population size for each problem instance is determined empirically as the minimal population size for the algorithm to find the optimum in all the runs.

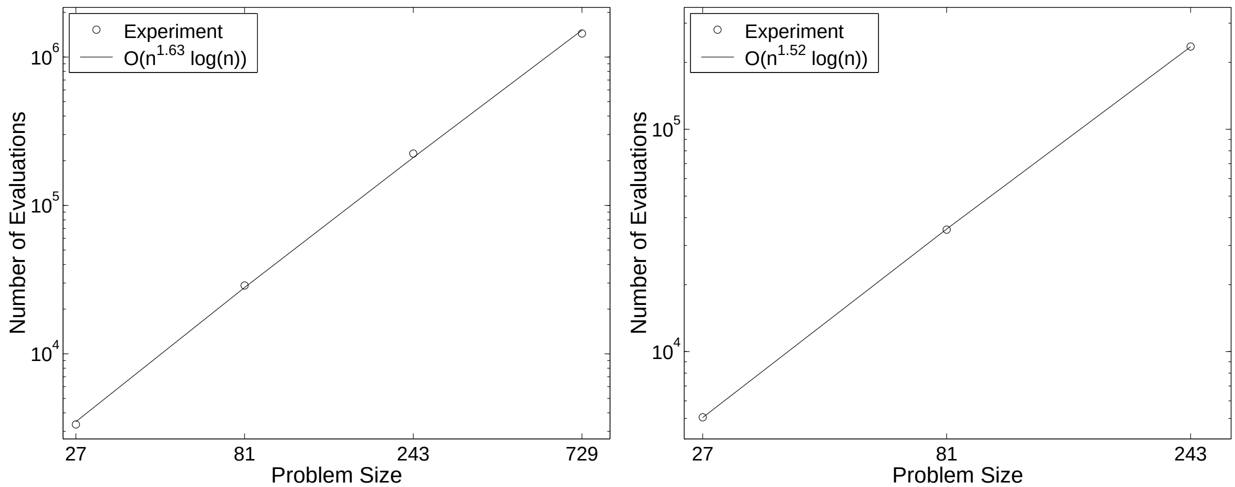
Binary tournament selection with replacement is used in all experiments and the window size for RTR is set to the number of bits in a problem. Bayesian networks with decision graphs are used and K2 metric with the term penalizing complex models is used to measure the quality of each candidate model.

6.4.2 Results

Figure 6.5 shows the performance of hierarchical BOA on the two hierarchical traps defined in Section 5.3.5. The number of bits in the first hierarchical trap ranged from $n = 27$ to $n = 729$; for the largest problem only 7 runs were performed due to the increased computational requirements. The number of bits in the second hierarchical trap ranged from $n = 27$ to $n = 243$. In both cases, the number of evaluations was approximated by a function of the form $O(n^k \log n)$, where the parameters k was set to fit the empirical results best. In both cases, the overall number of fitness evaluations grows subquadratically with the size of the problem.

Similar results were obtained on the HIFF function of Watson, Hornby, and Pollack (1998). The size of HIFF ranged from $n = 16$ bits to $n = 512$ bits. The number of fitness evaluations was approximated as above. Again, the overall number of fitness evaluations grows subquadratically.

An additional experiment was performed on the folded trap function to verify the effects of



(a) Hierarchical trap I.

(b) Hierarchical trap II.

Figure 6.5: The number of fitness evaluations until hierarchical BOA has found the optimum on the two hierarchical traps. The problem sizes range from $n = 27$ bits to $n = 729$ bits for the first trap, and $n = 27$ to $n = 243$ for the second trap. In both cases, the overall number of fitness evaluations grows subquadratically with the size of the problem.

niching on the preservation of alternative solutions. The folded trap of order 6 of size $n = 30$ bits (5 building blocks) was used as a test function. The tested function provides an interesting test; there are 32 global optima and $3,200,000$ local optima. With the population of size $N = 1,500$, the algorithm can only afford to maintain all the global optima. Figure 6.7 shows that hierarchical BOA was capable of discovering all the global optima after about 43 generations. The global optima were further propagated and maintained until the end of the run of 110 generations. An interesting observation is that although all the optima are multiply represented at the end of the run, the number of copies of the optima differs significantly. The reason for that behavior is that unlike fitness sharing, restricted tournament replacement does not require that the size of each niche be proportional to the average fitness in that niche.

6.5 Scalability of hBOA on Hierarchical Problems

The results presented in the above section indicated that hierarchical BOA is capable of solving the difficult hierarchical problems in subquadratic time. How does this relate to the BOA scalability theory presented in Chapter 4?

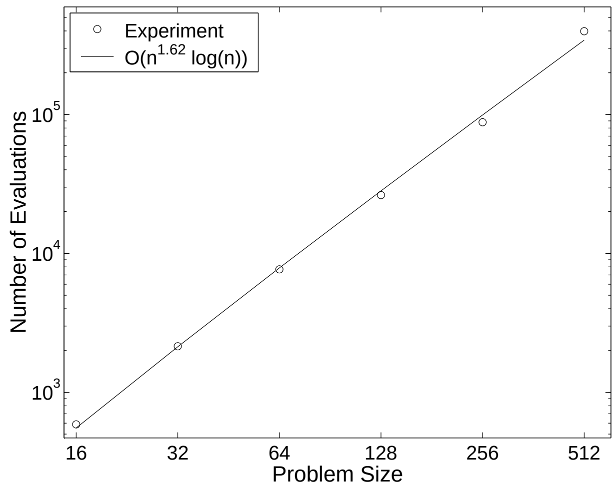


Figure 6.6: The number of fitness evaluations until hierarchical BOA has found the optimum on the hierarchical if-and-only-if (HIFF). The problem sizes range from $n = 16$ bits to $n = 512$ bits. The overall number of fitness evaluations grows subquadratically with the size of the problem.

The convergence of hierarchical BOA on the hierarchical problems proceeds sequentially from the bottom to the top level. On each level, the correct building blocks on that level must be discovered and their competitors must be eliminated. The number of evaluations required for hierarchical BOA to discover the correct building blocks on each level can be upper-bounded by the overall number of fitness evaluations required to solve the problem on the current level only. Using the BOA scalability theory, on each level l the number of evaluations can be upper bounded by $O(n_l^{1.55})$, where n_l is the number of subproblems from the lower level, which serve as the basic building blocks (or bits) for the current level. For example, in the hierarchical trap, $n_l = n/3^{l-1}$. The number of levels is proportional to the logarithm of the problem size on all the tested problems. In particular, hierarchical traps contain $\log_3 n$ levels, and HIFF contains $\log_2 n$ levels. For hierarchical traps, the overall number of evaluations can be therefore bounded by

$$\sum_{l=1}^{\log_3 n} O\left(\left[\frac{n}{3^{l-1}}\right]^{1.55}\right) = O(n^{1.55} \log n). \tag{6.6}$$

For HIFF, the same bound $O(n^{1.55} \log n)$ can be derived in a straightforward manner. Therefore, in both cases, the overall time to convergence is approximately equal to the number of levels times the time spent on a single-level problem of the same size. This result is confirmed by the empirical results.

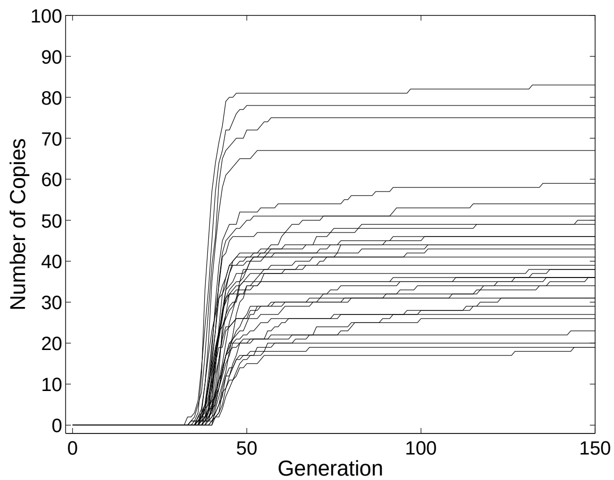


Figure 6.7: The number of copies of different global optima of the bipolar function. There are 32 optima in this function and all 32 are multiply represented at the end of the run.

If the number of levels grows faster than a logarithm of the problem size, the resulting bound is

$$E_{hboa} = O(n^{1.55} L(n)), \tag{6.7}$$

where $L(n)$ denotes the number of levels in the problem of size n. The bound given by the last equation could, however, change in some cases. Here we assumed that only two optima for each partition of the problem are preserved on each level to form the two building blocks that are to be juxtaposed on the next level. If the number of optima is not constant, the overall number of evaluations can be expected to increase due to the increased population size and number of generations. Similarly, if the size of the partitions on the lowest level grows with the size of the problem, the population size will have to increase according to the BOA population sizing.

6.6 How Would Other Methods Scale Up?

We performed a number of experiments with other search methods, including the traditional GAs, the one- and two-bit deterministic hill climbers, the stochastic hill climber, and the simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983). None of the tested algorithms was capable of solving hierarchical traps except for toy instances of at most $n = 27$ bits. That is why it was impossible to determine whether the growth of computational complexity of local search was exponential;

nonetheless, this section argues that exponential complexity can be expected using traditional perturbation operators within local search. Additionally, the section discusses the performance of traditional GAs briefly.

The reason for the poor performance of GAs with traditional recombination and no mutation is that without using an appropriate decomposition, the problem on any level of hierarchical traps requires exponentially large populations (Thierens & Goldberg, 1993). Therefore, even without hierarchy, GAs with traditional crossover yield intractable search. The following argument suggests that introducing mutation in GAs will not change the situation in this case (although mutation did yield polynomial search on composed traps, see sections 2.2.2 and 4.6).

Deterministic hill climbers start with a random string. In each iteration, they flip one or two bits of the current solution that result in the biggest increase in the fitness. If no more improvement is possible, the algorithms start over with a random starting point. Performing one- and two-bit flips is sufficient to discover some of the blocks 111 and 000 on the first level, but it fails in discovering the partial solutions on the higher levels. The only starting points from where the deterministic hill climber with one-bit flip is capable of reaching the optimum are in the strings that contain at most one 0 in each partition of 3 bits in the first-level decomposition. The probability of generating such a starting point is given by

$$\left(\frac{1}{2}\right)^{\frac{n}{3}},$$

because there are 4 allowable instances of the 3 bits out of 8 instances total, and there are $n/3$ such blocks in the string of length n. For the two-bit flips, any instance of the 3 bits that contains at most two 0s is allowed, so the probability of generating a good starting point grows as

$$\left(\frac{7}{8}\right)^{\frac{n}{3}}.$$

Therefore, in both cases, exponential number of restarts must be performed to reach the optimum. Consequently, the deterministic hill climbers were capable of solving only toy instances of 9 bits and failed in solving the next bigger problem of size $n = 27$ even with tens of millions restarts.

The stochastic hill climber with bit-flip mutation also requires exponentially many restarts or exponentially many steps. The argument can be supported by a Markov chain analysis of the stochastic hill climbing. It can be shown that to flip particular k bits at the same time by bit-flip mutation, approximately $O(n^k)$ trials are required (Mühlenbein, 1992). If there are several such groups, it is most difficult to flip the last group of k bits and therefore the same complexity can be expected (Mühlenbein, 1992). From any block of 3 bits, either 000 or 111 can be reached by flipping at most one bit; therefore, blocks 000 and 111 on the first level can be found in $O(n)$, yielding a string with half of the blocks equal to 000 and half of the blocks equal to 111. However, to find blocks 000000000 and 111111111 on the next level, three bits must be flipped at once (because of the deception of traps) and $O(n^3)$ trials are necessary. Continuing up a number of levels, the top level requires exponentially many trials to find the optimum.

An alternative approach to show the exponential number of trials of the stochastic hill climber is to bound the number of trials by a polynomial $O(n^k)$, and look at the starting points that allow the optimum to be found in the specified number of trials. This case is similar to the one where one or two bits are flipped deterministically, because $O(n^k)$ trials allows steps of size at most k flips. That results in an exponential number of restarts or finding a solution only up to a particular level. The maximum level that can be solved can be expected to be proportional to $\log k$.

The application of a cooling schedule of the simulated annealing relaxes the conditions somewhat; the groups of 3 and more bits do not have to be flipped at once, because the acceptance of each new solution is not deterministic. With a slow enough cooling schedule, it is possible to solve a problem of size $n = 27$; however, the complexity of the simulated annealing changes only marginally compared to hill climbers and exponentially many trials are needed. To summarize, although hierarchical BOA was capable of solving hierarchical traps of size $n = 729$, its competitors could solve a problem of only $n = 27$ bits (and even for this problem, most competitors failed).

6.7 Summary

This chapter described hierarchical BOA, which extends the original BOA by using local structures in Bayesian networks for more compact representation and restricted tournament replacement as a procedure for incorporating new candidate solutions into the original population. Additionally, the chapter applied hierarchical BOA to challenging hierarchical problems described in chapter 5. A summary of the key points of this chapter follows:

- Hierarchical BOA must extend the original BOA by incorporating (1) chunking and (2) niching.
- To ensure proper decomposition and chunking, the Bayesian networks with local structures can be used. The semantics of Bayesian networks remains the same when using local structures; the difference is that local structures allow a compact representation of model parameters (conditional probabilities), so that dependencies of large order can be encoded. Other models can be used for chunking—such as Huffman networks and Bayesian networks with hidden variables—but in any case an efficient representation must be enforced by incorporating some kind of local structures.
- Decision graphs are one of the most expressive local structures. Using decision graphs in Bayesian networks allows modifications to the learning algorithm so that the expressivity of decision graphs can be utilized. In particular, specialized dependencies are manipulated by splitting and merging the leaves in a decision graph for each variable.
- To ensure a proper preservation of alternative solutions, various niching methods proposed in genetic and evolutionary computation can be used. Hierarchical BOA uses restricted tournament replacement (RTR), which is based on restricted tournament selection, but is used in addition to traditional selection methods in a generational GA.
- Restricted tournament replacement incorporates each newly generated solution by first selecting a random subset of the original population and identifying a solution from this subset that is most similar to the new solution. The new solution that competes with the identified most similar solution of the subset; if the new solution outperforms the one in the original population, it replaces the other solution; otherwise, the new solution is discarded. The size of the subsets used by RTR is called the *window size*. A good heuristic is to choose the window size to be equal to the number of bits in the problem.
- Hierarchical BOA thus combines BOA with local structures and restricted tournament replacement.
- Hierarchical BOA is capable of solving challenging hierarchical problems in a subquadratic

number of fitness evaluations. In particular, the number of evaluations is approximately $O(n^{1.63}\log n)$ for the first hierarchical trap, $O(n^{1.52}\log n)$ for the second hierarchical trap, and $O(n^{1.56}\log n)$ for HIFF.

- The performance of BOA can be supported by a simple extension of BOA scalability theory. Using the extended theoretical model, the total number of evaluations required by hierarchical BOA to solve a problem of size n bits defined on $L(n)$ levels (the number of levels is a function of n) can be bounded by

$$O(n^{1.55}L(n)).$$

  Since for hierarchical traps and HIFF, the number of levels is proportional to a logarithm of the problem size, the theoretical model agrees with empirical results.

Chapter 7

Hierarchical BOA in the Real World

Earlier we have seen how BOA is capable of solving problems of bounded difficulty and just now we have designed and implemented hBOA, which is competent on difficult hierarchical problems. In both cases, special test functions have been used to test the algorithms on the boundary of their design envelopes. We have argued that if the algorithm is capable of solving artificial single-level and hierarchical problems of bounded difficulty, it should be applicable to other problems of the same difficulty or anything easier.

This chapter applies hierarchical BOA to several real-world problems to confirm that decomposition and hierarchical decomposition are useful concepts in approaching real-world problems. Two problems are considered: (1) two- and three-dimensional Ising spin glass systems with periodic boundary conditions, and (2) maximum satisfiability of logic formulas (MAXSAT). The chapter shows how easy it is to apply hierarchical BOA to combinatorial problems and achieve competitive or better results than problem-specific approaches. Additionally, the chapter relates the actual empirical performance of hierarchical BOA to the theory presented earlier in this thesis and suggests several ways to improve the performance of the algorithm on difficult real-world problems.

The chapter starts by defining the problem of determining the ground state of a given Ising spin-glass system. The scalability of hierarchical BOA is first empirically analyzed on a number of randomly generated instances of two-dimensional Ising spin-glass systems with periodic boundary conditions. The performance of hierarchical BOA is then improved by combining the algorithm with local search, and the hybrid is applied to an array of two- and three-dimensional spin glasses. Section 7.2 defines the maximum satisfiability (MAXSAT) problem and discusses its difficulty. To improve the performance, hierarchical BOA is again combined with local search. The hybrid is applied to several benchmark instances of MAXSAT and compared to other methods for solving MAXSAT. Finally, the chapter is summarized.

7.1 Ising Spin Glasses

The task of finding the ground state of a given Ising spin-glass system is a well known problem of statistical physics. In context of GAs, Ising spin-glass systems are usually studied because of their interesting properties, such as symmetry and a large number of plateaus (Pelikan, Goldberg, & Cantú-Paz, 1998; Pelikan & Mühlenbein, 1999; Naudts & Naudts, 1998; Van Hoyweghen, Goldberg, & Naudts, 2001; Van Hoyweghen, 2001; Mühlenbein, Mahnig, & Rodriguez, 1999).

The physical state of an Ising spin-glass system is defined by (1) a set of spins $(\sigma_0, \sigma_1, \ldots, \sigma_{n-1})$, where each spin σ_i can obtain a value from $\{+1, -1\}$, and (2) a set of coupling constants J_{ij} relating pairs of spins σ_i and σ_j. A Hamiltonian specifies the energy of the system as

$$H(\sigma) = -\sum_{i,j=0}^{n} J_{ij}\sigma_i\sigma_j, \tag{7.1}$$

where $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_{n-1})$ specifies the physical state of spins. The task is to find the state of spins for given coupling constants J_{ij} so that the energy of the system is *minimized*. The state that minimizes the energy is called the *ground state of the system*. To transform the problem into maximization, negative energy can be considered.

Minimization of spin-glass energy can be easily transformed into a well known combinatorial problem called *minimum-weight cut* (MIN-CUT); analogously, MIN-CUT problem instances can be transformed into Ising spin glasses. Since MIN-CUT is NP-complete[1] (Monien & Sudborough, 1988), the task of finding the ground state of unconstrained Ising spin-glass systems is NP-complete in its general form. Here we consider a special case, where the spins are arranged on a two- or three-dimensional grid and each spin interacts with only its nearest neighbors in the grid. Periodic boundary conditions are used to approximate the behavior of a large-scale system. Therefore, spins are arranged on a two- or three-dimensional toroid. Additionally, we constrain couplings constants

[1]NP-complete problems are solvable by nondeterministic Turning machines in polynomial time. However, there is no known algorithm for solving any NP-complete problem in polynomial time—if a polynomial algorithm was found for any NP-complete problem, all problems that are solvable by nondeterministic Turing machines could be solvable in polynomial time.

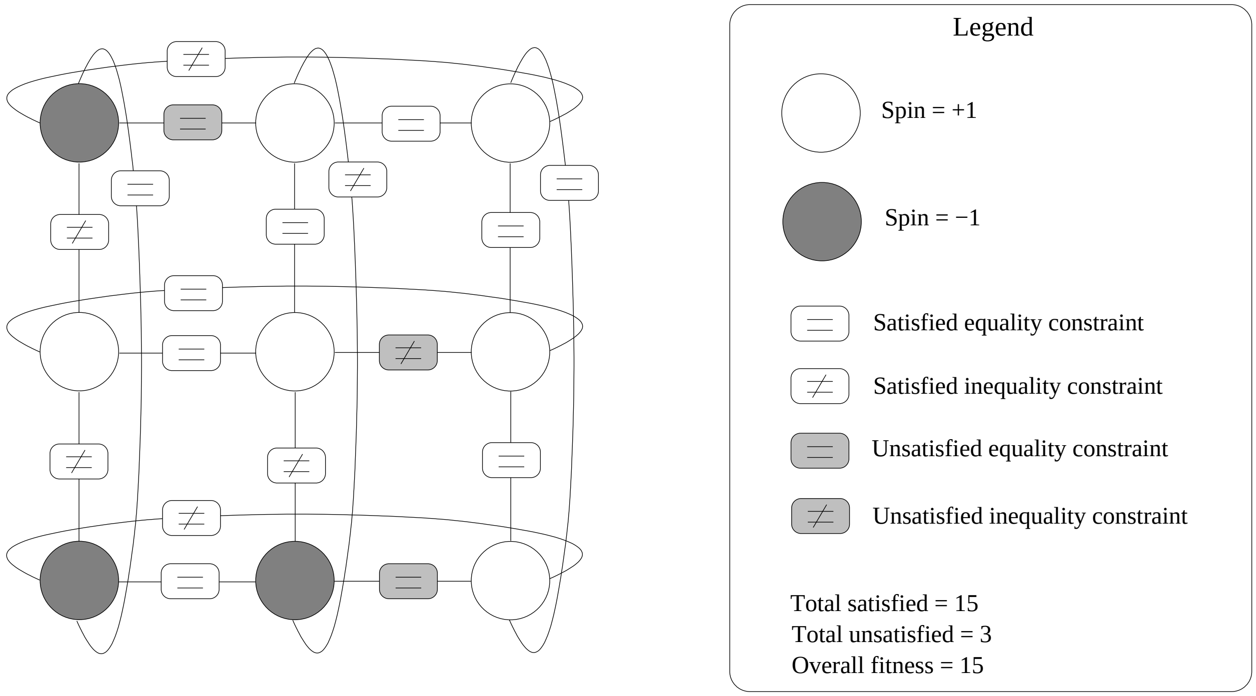


Figure 7.1: An example two-dimensional spin-glass system of size $n = 9$ spins, arranged on a toroid of size 3×3. Spins are shown as circles, non-zero coupling constants are shown as edges between pairs of spins. If $J_{ij} = -1$, the coupling constant introduces an equality constraint, if $J_{ij} = 1$, the coupling constant represents an inequality constraint. The more constraints are satisfied, the lower the energy of the underlying spin-glass system. The task is to maximize the number of satisfied constraints, which is equivalent to minimizing the energy of the underlying spin-glass system. The figure shows the values of the spins in the ground state of the system (state with minimal energy).

to contain only two values, $J_{ij} \in \{+1, -1\}$. In the two-dimensional case, several algorithms exist that can solve the restricted class of spin glasses in polynomial time. We will compare the best known algorithms to hierarchical BOA at the end of this section. However, none of these methods is applicable to three-dimensional spin glasses.

Figure 7.1 shows an example two-dimensional spin-glass system with 9 spins arranged in a 3×3 toroid.

7.1.1 Methodology

In hierarchical BOA, each state of the system is represented by a binary string of size n, where n is the total number of spins. Each bit in a solution string determines the state of the corresponding spin: 0 denotes the state -1, 1 denotes the state $+1$. To estimate the scalability of hierarchical BOA on two-dimensional spin-glass systems, we tested the algorithm on a number of two-dimensional spin-glass systems of sizes ranging from $n = 10 \times 10 = 100$ spins to $n = 16 \times 16 = 256$ spins. For each problem size, we generated 8 random problem instances. To generate a random instance, each coupling constant was set to -1 with probability 50%, otherwise the constant was set to $+1$. To confirm that a correct ground state was found for each system, we verified the results using the Spin Glass Ground State Server provided by the group of Prof. Michael Jünger[2].

For each problem instance, 30 independent runs are performed and hierarchical BOA is required to find the optimum in all the 30 runs. The performance of hierarchical BOA is measured by the average number of evaluations until the optimum is found. The population size for each problem instance is determined empirically as the minimal population size for the algorithm to find the optimum in all the runs. Binary tournament selection with replacement is used in all experiments and the window size for RTR is set to the number of bits (spins) in a problem. Bayesian networks with decision graphs are used and K2 metric with the term penalizing complex models is used to measure the quality of each candidate model.

7.1.2 Results

Figure 7.2 shows the total number of evaluations for each problem instance (averaged over 30 runs) and its average over all problem instances of the same size. Therefore, each point of the final average corresponds to $8 \times 30 = 240$ runs. Additionally, the figure shows the best-fit polynomial approximating the growth of the number of evaluations until the optimum was found. The total number of evaluations appears to grow polynomially as $O(n^{2.25})$, where n is the number of spins in the system.

7.1.3 Discussion

There are two important questions that must be answered at this point:

- Why does the number of evaluations grow faster than predicted by the BOA and hBOA scalability theory?
- How does the overall complexity compare to other known methods for the considered subclass of Ising spin glasses?

[2] http://www.informatik.uni-koeln.de/ls_juenger/projects/sgs.html

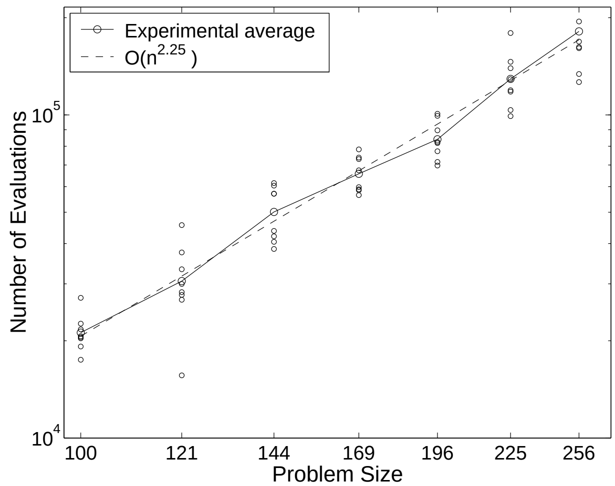


Figure 7.2: The performance of hierarchical BOA on two-dimensional Ising spin-glass systems. The size of the tested problems ranged from $n = 100$ for a 10×10 grid to $n = 256$ for a 16×16 grid. The figure shows the number of evaluations for each problem instance, the average number of evaluations over all instances of the same size, and the empirical complexity.

The answer to the first question lies in the assumptions of the BOA and hBOA scalability theory. In particular, the assumptions stated that the order of building blocks is constant and that the problem can be decomposed into separable subproblems. If either assumption is not satisfied, the complexity of hierarchical BOA could change. For instance, if the order of building blocks in any building-block decomposition grows as a logarithm of the problem size, the number of fitness evaluations is still expected to grow polynomially with the size of the problem, but the order of the polynomial is greater than that for building blocks of a constant order. Although it far from obvious whether the necessary building-block order grows with the size of spin glasses, it is clear that the second assumption is not satisfied; the building blocks in spin-glasses are interconnected.

To answer the second question, let us first compute the overall computational complexity of hierarchical BOA. Each evaluation of a spin-glass state can be bounded by $O(n)$ trivial operations. There are $O(n^{2.25})$ total evaluations, so the overall time spent in fitness evaluation grows as $O(n^{3.25})$. However, in this case the complexity of model building dominates other factors. In each generation, the construction of a model takes approximately $O(n^3 + n^2N)$ steps (Pelikan, Goldberg, & Cantú-Paz, 2000b), where N is the population size. According to the empirical results, the growth of the population size can be approximated as $O(n^{1.75})$; then, the time complexity of model building in each generation can be bounded by $O(n^{3.75})$. There are approximately $O(\sqrt{n})$

generations total, so the overall time spent in model building can be bounded by $O(n^{4.25})$. The remaining overhead—such as model sampling and bookkeeping—is overshadowed by model building and fitness evaluation. Therefore, model building dominates other complexity factors, and the overall time complexity of hierarchical BOA can be bounded by $O(n^{4.25})$. If the model was updated incrementally in each generation, the time complexity can be expected to decrease to somewhere between $O(n^{3.25})$ and $O(n^{4.25})$.

There are several algorithms that attempt to solve the above special case of two-dimensional spin glasses. Kardar and Saul (1994) can solve a subset of the considered class of spin-glass systems in $O(n^{3+\epsilon})$, where $\epsilon < 1$. De Simone, Diehl, Jünger, and Reinelt (1996) proposed a method to solve the above case of spin-glass systems in approximately $O(n^3)$ for $n \leq 50$, but they did not achieve polynomial running time for bigger problems. Most recently, Galluccio and Loebl (Galluccio & Loebl, 1999a; Galluccio & Loebl, 1999b) proposed an algorithm for solving spin glasses in $O(n^{3.5})$ for all graphs with bounded genus (two-dimensional toroids are a special case of graphs with bounded genus). So, the overall time complexity of the best currently known algorithm for the considered class of spin glasses is $O(n^{3.5})$.

The above results indicate that hBOA performs slightly worse than the best known algorithm in the field; in particular, hBOA requires $O(n^{4.25})$ steps, whereas the algorithm of Galluccio and Loebl requires $O(n^{3.5})$ steps. However, hBOA does not use any problem-specific knowledge except for the evaluation of possible states of the system, whereas the method of Galluccio and Loebl fully relies on the knowledge of the problem structure and its properties. Even without requiring any problem-specific information in advance, hBOA is capable of competing with the state-of-the-art methods in the field. Furthermore, hBOA does not explicitly restrict the interaction structure of a problem; consequently, hBOA is applicable to spin glasses in more than two dimensions and other spin glasses that fall outside the scope of the method of Galluccio and Loebl.

There are several improvements that can be incorporated into hBOA to increase its efficiency. First of all, as mentioned above, time complexity of hBOA can be reduced by using incremental updates to the model. Additionally, using local search to improve candidate solutions in hBOA can lead to a significant reduction of the required population size (Sastry, 2001a). Finally, prior information about the structure of a problem can be used to improve model building and decrease its computational complexity.

The following section implements one of the aforementioned improvements. In particular, the section presents a hybrid method that combines hBOA with a deterministic hill climber based on one-bit flips. The hill climber is used to improve every candidate solution in the population.

7.1.4 Solving Spin Glasses using hBOA + Local Search

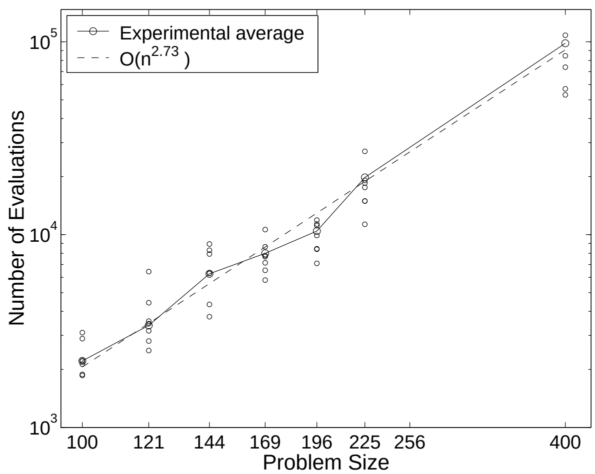
Using local search usually improves the performance of selectorecombinative search, because the search can focus on local optima, which reveal more information about the underlying structure of the problem than randomly generated solutions do. Furthermore, the selectorecombinative search can focus its exploration on basins of attraction as opposed to individual solutions.

It is fairly easy to incorporate local search in hierarchical BOA. There are several strategies that can be used. Here we apply a deterministic hill climber (DHC) with one-bit flips to each evaluated candidate solution. DHC repeatedly applies one-bit flips that improve the solution at hand the most until no more improvement is possible. In the spin-glass problem, DHC needs only one pass through the fitness function to find the best flip to perform and determine the energy of the resulting system. This holds for any constraint satisfaction problem, because the best flip can be determined by parsing the constraints, and accumulating the effects of flipping the bits involved in each constraint. Therefore, for each iteration of DHC, only one fitness evaluation is necessary.

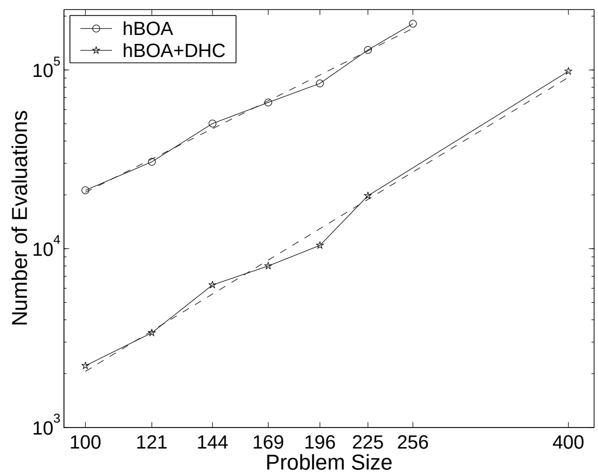
There are two ways of exploiting the results of local search. In the Baldwinian approach, the fitness of the original solution is set to the fitness of the solution after local search but the original solution is not modified; in the Lamarckian approach, both the fitness as well as the solution are modified according to the result of local search. We take the Lamarckian approach by modifying the solution as well as its fitness according to the result of local search.

A significant reduction of the required population sizes and running times was obtained when using DHC to improve each candidate solution. Figure 7.3(a) shows the performance of hierarchical BOA with DHC on the set of randomly generated spin-glass instances used in the previous section. Figure 7.3(b) compares the performance of hBOA+DHC with that of hBOA without DHC.

There are two important observations. First, hBOA+DHC is capable of solving much larger problem instances using the same number of evaluations; for example, the number of evaluations



(a) Performance of hBOA with DHC.



(b) Comparison of hBOA with and without DHC.

Figure 7.3: The number of fitness evaluations until the optimum was found with hierarchical BOA with the discrete hill climber (DHC) on two-dimensional Ising spin-glass systems. The results are compared to those of hBOA without DHC.

required by hBOA+DHC to solve problems of size $n = 400$ is *lower* than the number of evaluations required by hBOA alone to solve problems of size of only $n = 225$. Second, the running times significantly decrease, because the population sizes decrease approximately tenfold and the model building is therefore much more efficient. In fact, the running times are *better* than those reported by Galluccio and Loebl. For instance, the problem instances of $n = 400$ bits took on average 9.7 minutes per instance on a Pentium II/400MHz (the worst case took about 21 minutes, the best case took about 2.92 minutes), while Galluccio and Loebl reported times of about 25 minutes on an Athlon/500MHz. Therefore, hBOA+DHC is capable of finding the optimum significantly faster despite that hBOA+DHC does not assume any particular structure of the problem.

7.1.5 From 2D to 3D

Despite that competent methods exist for solving two-dimensional spin glasses, none of these methods is directly applicable to three-dimensional spin glasses. That is why there is an ongoing research on extending existing methods from two to three dimensions (Loebl, 2000). Nonetheless, since hBOA does not explicitly use the dimensionality of the underlying spin-glass problem, it is straightforward to apply hBOA+DHC to three-dimensional spin glasses.

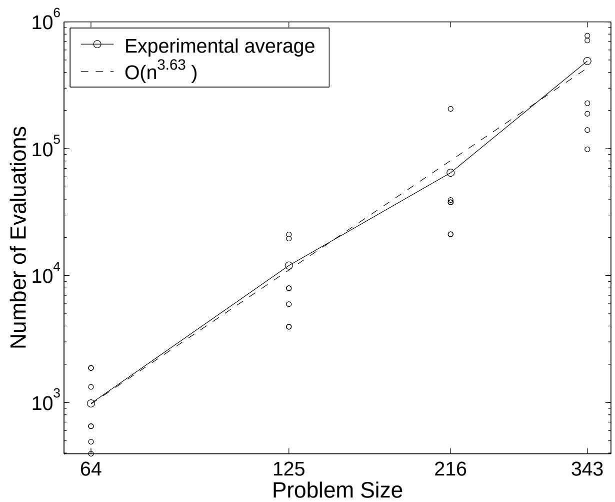


Figure 7.4: The number of fitness evaluations until the optimum was found with hierarchical BOA with the discrete hill climber (DHC) on three-dimensional Ising spin-glass systems.

To test the scalability of hBOA+DHC, eight random spin-glass systems on a three-dimensional cube with periodic boundary conditions were generated for the systems ranging from $n = 4\times4\times4 = 64$ to $n = 7 \times 7 \times 7 = 343$ spins. Since no other method exists to verify whether the found state actually represents the ground state, the hybrid hBOA+DHC was first run on each instance with an extremely large population of orders of magnitude larger than the expected one. After a number of generations, the best solution found was assumed to represent the ground state. The bisection method was then used to determine the optimal population size for each case.

Figure 7.4 shows the number of evaluations until hBOA+DHC found the ground state of the tested three-dimensional Ising spin-glass instances. The overall number of evaluations appears to grow polynomially as $O(n^{3.65})$. That means that increasing the dimensionality of spin-glass systems increases the complexity of solving these systems. However, efficient performance is retained even in three dimensions.

7.2 Maximum Satisfiability (MAXSAT)

The task of find an interpretation of predicates that maximizes the number of satisfied clauses of a given predicate logic formula expressed in conjunctive normal form, or MAXSAT, is an important problem of complexity theory and artificial intelligence. Since MAXSAT is NP-complete in its general form, there is no known algorithm that can solve MAXSAT in worst-case polynomial time.

In context of GAs, MAXSAT is usually used as an example class of problems that *cannot* be efficiently solved by the selectorecombinative bias (Rana & Whitley, 1998), although some positive results were reported with adaptive fitness (Gottlieb, Marchiori, & Rossi, 2002). The reason for poor GA performance appears to be that short-order partial solutions lead away from the optimum (sometimes in as many as 30% of predicate variables) as hypothesized by Rana and Whitley (1998). Since hierarchical BOA outperforms GAs on complex decomposable and hierarchically decomposable problems, it would be interesting to apply hierarchical BOA to MAXSAT and see whether hierarchical BOA is able to make any difference or not.

Here we consider the case where each formula is given in the conjunctive normal form with clauses of length at most k; the formulas in this form are called k-CNF. A CNF formula is a *logical and* of the clauses, where each clause is a *logical or* of k or less literals. Each literal is either a predicate or a negation of a predicate. An example 3-CNF formula over predicates (variables) X_1 to X_5 is

$$(X_5 \vee X_1 \vee \neg X_3) \wedge (X_2 \vee X_1) \wedge (\neg X_4 \vee X_1 \vee X_5).$$

An interpretation of predicates assigns each predicate either true or false; for example, $(X_1 = \text{true}, X_2 = \text{true}, X_3 = \text{false}, X_4 = \text{false}, X_4 = \text{true})$ is an interpretation of X_1 to X_5. The task is to find an interpretation that maximizes the number of satisfied clauses in the given formula. For example, the assignment $(X_1 = \text{true}, X_2 = \text{true}, X_3 = \text{true}, X_4 = \text{true}, X_5 = \text{true})$ satisfies all the clauses in the above formula, and is therefore one of the optima of the corresponding MAXSAT problem. MAXSAT is NP complete for k-CNF if $k \geq 2$. However, it is possible to check whether a 2-CNF formula is satisfiable (all clauses can be satisfied) in polynomial time.

7.2.1 Methodology

In hierarchical BOA, each candidate solution represents an interpretation of predicates in the problem. Each bit in a solution string corresponds to one predicate; true is represented by 1, false is represented by 0. The fitness of a solution is equal to the number of satisfied clauses given the interpretation encoded by the solution. The deterministic hill climber with one-bit flips is used to reduce the population-sizing requirements and improve the efficiency of the search. The hill climber flips the bit that improves the current solution the most until no more improvement is possible. DHC for MAXSAT is often called GSAT in the machine learning community (Selman, Levesque, & Mitchell, 1992). As in spin glasses, each iteration of GSAT can be performed in one pass through the formula.

For each problem instance, 30 independent runs are performed and hierarchical BOA is required to find the optimum in all the 30 runs. The performance is measured by the average number of evaluations until the optimum is found. The population size is determined empirically as the minimal population size for the algorithm to find the optimum in all the runs. Binary tournament selection with replacement is used in all experiments and the window size for RTR is set to the number of bits (predicates) in the problem. Bayesian networks with decision graphs are used and K2 metric with the term penalizing complex models is used to measure the quality of each candidate model.

7.2.2 Other Methods Included in Comparison

Three methods are included in the comparisons: (1) GSAT, (2) WalkSAT, and (3) Satz. GSAT (Selman, Levesque, & Mitchell, 1992) is a deterministic hill climber using one-bit flips. The initial solution of GSAT is generated at random. In each iteration, GSAT changes the interpretation of the predicate that leads to the largest increase in the number of satisfied clauses. If no more improvement of the current solution is possible, GSAT is restarted with a random solution.

WalkSAT extends GSAT to incorporate random changes. In each iteration, WalkSAT performs the greedy step of GSAT with the probability p; otherwise, one of the predicates that are included in some unsatisfied clause is randomly selected and its interpretation is changed. Best results have been obtained with $p = 0.5$, where both GSAT and a random perturbation are applied with the same probability. However, the optimal choice of p might change from application to application.

The last method included in the comparison is Satz (Li & Anbulagan, 1997), which is an extension of the Davis-Putnam-Logemann-Loveland algorithm (Davis, Logemann, & Loveland, 1962). Satz uses resolution (a sound and complete proof procedure for CNF) and several simple heuristics to find a satisfying interpretation for the input formula or a proof that the formula is unsatisfiable. Satz does not actually solve MAXSAT; instead of finding an interpretation that maximizes the number of satisfied clauses in the input formula, Satz only verifies whether the formula is satisfiable or not. Therefore, one must be careful when interpreting the results of the comparison, because proving whether a formula is satisfiable or not is a different task than that of finding an assignment that maximizes the number of satisfied clauses.

7.2.3 Tested Instances

Two types of MAXSAT instances are tested: (1) random satisfiable 3-CNF formulas, and (2) instances of combined-graph coloring translated into MAXSAT. All tested instances have been downloaded from the Satisfiability Library SATLIB[3].

Instances of the first type are randomly generated satisfiable 3-CNF formulas. All instances belong to the phase transition region (Cheeseman, Kanefsky, & Taylor, 1991), where the number of clauses is equal to $4.3n$ (n is the number of predicates). *Random* problems in the phase transition are known to be the most difficult ones for most MAXSAT heuristics (Cheeseman, Kanefsky, & Taylor, 1991).

There are two approaches to ensuring that random formulas are satisfiable. The first approach generates a satisfying interpretation first, and then generates only such clauses that are satisfied in the specified interpretation; these instances are called *forced satisfiable instances*. The second approach is to generate formulas at random first (with uniform distribution), and then filter out unsatisfiable instances using some of the complete algorithms such as Satz; these instances are called *unforced filtered satisfiable instances*. It has been shown forced satisfiable instances are easier than unforced filtered satisfiable instances. Here, hard instances are used, which are obtained by filtering (formulas are unforced filtered). Despite generating problem instances from the phase transition, all tested instances are rather easy for both WalkSAT and Satz.

Instances of the second type were generated by translating graph-coloring instances to MAXSAT. In graph coloring, the task is to color the vertices of a given graph so that no connected vertices share the same color. The number of colors is bounded by a constant. Every graph-coloring instance can be mapped into a MAXSAT instance by introducing one predicate for each pair (color, vertex), and creating a formula that is satisfiable if and only if exactly one color is chosen for each

[3]http://www.satlib.org/

SATLIB Archive	Description	Vars.	Clauses	Properties
uf20-91.tar.gz	Random 3-CNF instances	20	91	Easy for WalkSAT
uf50-218.tar.gz	from the phase transition	50	218	and Satz.
uf75-325.tar.gz	region ($c = 4.3n$).	75	325	
uf100-430.tar.gz		100	430	
uf125-538.tar.gz		125	538	
uf150-645.tar.gz		150	645	
sw100-8-lp1-c5.tar.gz	Instances obtained by	500	3100	Most instances
sw100-8-lp2-c5.tar.gz	translating graph	500	3100	difficult for
sw100-8-lp3-c5.tar.gz	coloring to MAXSAT.	500	3100	WalkSAT, some
sw100-8-lp4-c5.tar.gz	Graphs created by	500	3100	difficult for
sw100-8-lp5-c5.tar.gz	combining regular	500	3100	Satz.
sw100-8-lp6-c5.tar.gz	lattices with random	500	3100	
sw100-8-lp7-c5.tar.gz	graphs are considered.	500	3100	
sw100-8-lp8-c5.tar.gz	Graphs are 5-colorable.	500	3100	

Table 7.1: An overview of MAXSAT instances used in the experiments.

vertex, and the colors of the vertices corresponding to each edge are different.

Here, graph-coloring instances translated into MAXSAT instances are generated by combining regular ring lattices and random graphs with a specified number of neighbors (Gent, Hoos, Prosser, & Walsh, 1999). Combining two graphs consists of selecting (1) all edges that overlap in the two graphs, (2) a random fraction $(1-p)$ of the remaining edges from the first graph, and (3) a random fraction p of the remaining edges from the second graph. By combining regular graphs with random ones, the amount of structure in the resulting graph can be controlled; the smaller the p, the more regular the graphs are (for $p = 0$, the resulting graph is a regular ring lattice).

For small values of p (from about 0.003 to 0.03), MAXSAT instances of the second type are extremely difficult for WalkSAT and other methods based on local search. On the other hand, for higher values of p, some instances are extremely difficult for Satz and other complete methods. All instances are created from graphs of 100 vertices and 400 edges that are colorable using 5 colors, and each coloring is encoded using 500 binary variables (predicates).

Table 7.1 summarizes tested MAXSAT instances and their basic properties.

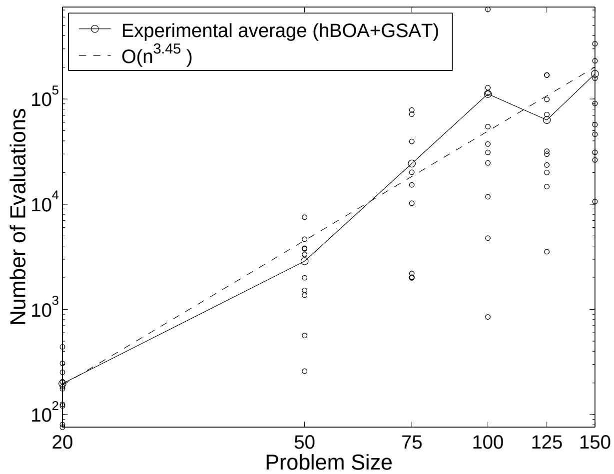


Figure 7.5: Results of hierarchical BOA with GSAT on the MAXSAT for randomly generated 3-CNF satisfiable formulas (unforced). The problem instances were downloaded from SATLIB.

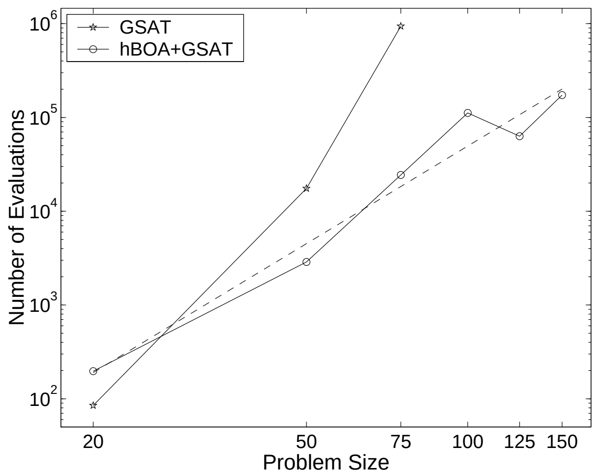
7.2.4 Results on Random 3CNF Satisfiable Instances

Figure 7.5 shows the performance of hierarchical BOA with GSAT on MAXSAT for randomly generated (unforced filtered) 3-CNFs. Ten instances are tested for each problem size[4]. The performance can be approximated by a polynomial $O(n^{3.45})$, although an exponential growth can be expected in the worst case.

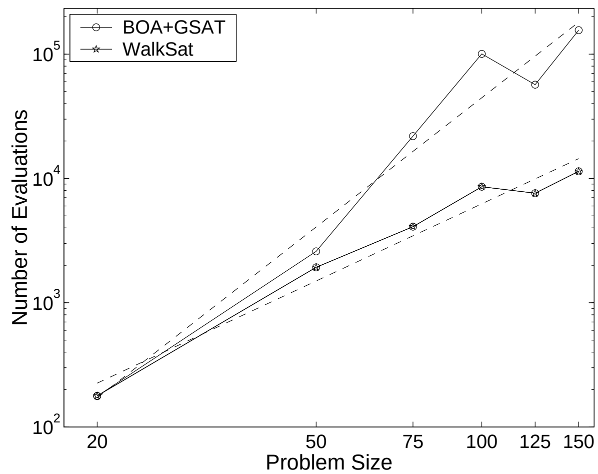
How does the performance of hBOA+GSAT compare to that of other approaches? Figure 7.6(a) compares the performance of hBOA+GSAT with that of GSAT alone. GSAT is capable of solving only the simplest instances of up to $n = 75$ variables, because the computational time requirements of GSAT grow extremely fast. Already for instances of $n = 100$ variables, GSAT could not find the optimal interpretation even after days of computation. The increasing slope of GSAT complexity in logarithmic scale indicates that the number of evaluations required by GSAT is exponential in the number of variables in the problem. Therefore, GSAT alone cannot solve the problem efficiently, although it improves the efficiency of hierarchical BOA when used in the hybrid hBOA+GSAT.

Figure 7.6(b) compares the performance of hBOA+GSAT with that of WalkSAT. The figure indicates that the performance of WalkSAT is better than that of hBOA+GSAT both in the magnitude and in the growth. Therefore, a simple randomization of GSAT performs better than the sophisticated bias of hBOA+GSAT. Nonetheless, although hBOA uses selectorecombinative

[4]In particular, the first ten instances from the archives downloaded from SATLIB are used.



(a) Comparison of hBOA+GSAT with GSAT alone.



(b) Comparison of hBOA+GSAT with WalkSAT.

Figure 7.6: Comparison of the performance of hBOA+GSAT with the performance of GSAT alone and WalkSAT on the MAXSAT for randomly generated 3-CNF satisfiable formulas (unforced). The problem instances were downloaded from SATLIB. hBOA+GSAT outperforms GSAT alone; however, hBOA+GSAT is outperformed by WalkSAT.

bias based on problem decomposition and hierarchical problem decomposition, it is capable of competing with local search on problem instances that are rather easy for local search.

Satz can verify satisfiability of all tested instances relatively fast. That indicates that the tested instances are hard only for GSAT, which appears to require exponential time to solve these instances. The performance of hBOA+GSAT is comparable to (but worse than) that of WalkSAT, and it is qualitatively better than that of GSAT.

7.2.5 Results on Combined-Graph Coloring

Randomly generated 3-CNF instances are rather easy for most tested algorithms. Nonetheless, real-world problems are not random, most real-world problems contain a considerable amount of regularities. Combined-graph coloring described in Section 7.2.3 provides an interesting class of problems, where regularity is combined with randomness. By controlling the relative amounts of structure and randomness, interesting classes of problems can be generated. This section tests the algorithms that performed relatively well on random 3-CNF, and applies these algorithms to combined-graph coloring (translated into MAXSAT).

Instance	Vars	Clauses	p	**hBOA+GSAT Evaluations**	**WalkSAT Evaluations**
`SW100-8-5/sw100-1.cnf`	500	3600	2^{-5}	1,262,018	$>$ 40,000,000
`SW100-8-5/sw100-2.cnf`	500	3600	2^{-5}	1,099,761	$>$ 40,000,000
`SW100-8-5/sw100-3.cnf`	500	3600	2^{-5}	1,123,012	$>$ 40,000,000
`SW100-8-6/sw100-1.cnf`	500	3600	2^{-6}	1,183,518	$>$ 40,000,000
`SW100-8-6/sw100-2.cnf`	500	3600	2^{-6}	1,324,857	$>$ 40,000,000
`SW100-8-6/sw100-3.cnf`	500	3600	2^{-6}	1,629,295	$>$ 40,000,000
`SW100-8-7/sw100-1.cnf`	500	3600	2^{-7}	1,732,697	$>$ 40,000,000
`SW100-8-7/sw100-2.cnf`	500	3600	2^{-7}	1,558,891	$>$ 40,000,000
`SW100-8-7/sw100-6.cnf`	500	3600	2^{-7}	1,966,648	$>$ 40,000,000
`SW100-8-7/sw100-7.cnf`	500	3600	2^{-7}	1,222,615	$>$ 40,000,000
`SW100-8-8/sw100-1.cnf`	500	3600	2^{-8}	1,219,675	$>$ 40,000,000
`SW100-8-8/sw100-2.cnf`	500	3600	2^{-8}	1,537,094	$>$ 40,000,000
`SW100-8-8/sw100-6.cnf`	500	3600	2^{-8}	1,650,568	$>$ 40,000,000
`SW100-8-8/sw100-7.cnf`	500	3600	2^{-8}	1,287,180	$>$ 40,000,000

Table 7.2: The results of hierarchical BOA with GSAT on the MAXSAT instances that are practically unsolvable by WalkSAT. The instances are generated by translating the graph coloring of 5-colorable combined graphs. All instances were downloaded from SATLIB.

Although regular ring lattices ($p = 0$) can be solved by WalkSAT efficiently (Gent, Hoos, Prosser, & Walsh, 1999), introducing even a slight perturbation to the regular graph by combining it with a random graph severely affects WalkSAT's performance. More specifically, WalkSAT is practically unable to solve any instances with $p \leq 2^{-5}$ even with very large number of restarts and trials. For these problem instances, the performance of GSAT is also poor.

On the other hand, hBOA+GSAT is capable of solving all these instances despite their large size (500 variables). Table 7.2 shows the performance of hBOA on several instances that are practically unsolvable by WalkSAT. WalkSAT is not able to solve any of these instances even when allowed to check over 40 million interpretations (when the runs are terminated).

Satisfiability of most instances of the second type is easy to verify with Satz. However, there are several instances for which Satz performs poorly. We selected some of those instances, and applied hBOA+GSAT to them. Table 7.3 shows the performance of hBOA+GSAT and Satz on those instances. The performance of Satz is measured by the number of branchings, which corresponds to the number of decisions that Satz must make until it outputs the final answer. Since Satz is

Instance	Vars	Clauses	p	hBOA+GSAT Evaluations	Satz Branchings
`SW100-8-1/sw100-1.cnf`	500	3600	2^{-1}	2,927,182	314,051,251
`SW100-8-1/sw100-15.cnf`	500	3600	2^{-1}	3,578,068	12,350,045
`SW100-8-2/sw100-44.cnf`	500	3600	2^{-2}	1,983,397	16,017,146

Table 7.3: The results of hierarchical BOA with GSAT on the MAXSAT instances that are extremely hard for Satz. The instances are generated by translating the graph coloring of 5-colorable combined graphs. All instances were downloaded from SATLIB.

deterministic, only one run of Satz suffices for each instance. For Satz-hard instances shown in the table, hBOA+GSAT proves its robustness, because it is capable of solving all these instances in time comparable to that on other instances.

7.2.6 Discussion

There are several important observations regarding the performance of the hybrid hBOA+GSAT on the tested MAXSAT instances.

hBOA+GSAT outperformed GSAT alone on all problem instances; not surprisingly, hBOA is capable of supplying much better starting points for GSAT than random restarts do. However, on the class of randomly generated 3-CNF, the hybrid hBOA+GSAT is outperformed by a randomized GSAT called WalkSAT.

On the other hand, for those problem instances that are practically unsolvable by any local search methods included in the comparison (WalkSAT and GSAT), hBOA+GSAT retains efficient performance. In particular, MAXSAT instances obtained by translating graph coloring of graphs with a large amount of structure and a little amount of randomness cannot be solved by GSAT or WalkSAT even after tens of millions of evaluations, whereas hBOA+GSAT is capable of solving all these problems in fewer than two million evaluations. Therefore, hBOA+GSAT can solve those instances that are easy for local search (random 3-CNF), but it is not limited to those instances—it can solve also problems that are practically unsolvable by local search alone.

Although Satz can verify satisfiability of most formulas efficiently, for several instances the performance of Satz is extremely poor. As shown in Table 7.3, hBOA+GSAT is capable of solving even those Satz-hard instances in time comparable to that on other instances. Nonetheless, recall that Satz does not actually solve MAXSAT. Satz can only find a satisfying interpretation of the formula or a proof that the formula is not satisfiable, but it cannot find the best interpretation for an unsatisfiable formula; hBOA, GSAT, and WalkSAT can.

To summarize, hierarchical BOA can solve an array of MAXSAT problems of different structure without requiring any information about the problem except for a measure of performance of each candidate solution and the number of bits in candidate solutions. On some MAXSAT problems, hierarchical BOA is outperformed by specialized heuristics; however, hierarchical BOA can solve all problems that it was applied to, whereas other methods are very good on some problems and very bad on other problems. Furthermore, note that the same algorithm was applied to onemax, composed traps, exponentially scaled deceptive problems, hierarchical traps, spin glasses, and MAXSAT. Hierarchical BOA is therefore robust not only with respect to specialized classes of problems such as spin glasses or MAXSAT, but it provides a scalable solution to an array of qualitatively different problem classes, as long as these problems are decomposable or hierarchically decomposable.

7.3 Summary

This chapter applied hierarchical BOA to two real-world problems: (1) two- and three-dimensional Ising spin-glass systems and (2) maximum satisfiability of predicate-logic formulas. The chapter compared the performance of BOA to that of other algorithms for solving these two classes of problems. A summary of the key points of this chapter follows:

- Finding the ground state of an Ising spin-glass system is an important problem of statistical physics. The behavior of the system at large scale can be approximated using periodic boundary conditions. Each solution in hierarchical BOA represents a possible state of the system by setting each spin to one of two values. The task is to find the state that minimizes the energy of the system.

- For 2D Ising spin glasses with periodic boundary conditions, hBOA can find the ground state in time $O(n^{4.25})$, while the best method for the problem is capable of finding the ground state in time $O(n^{3.5})$. Therefore, hBOA does not outperform the best method in the field with respect to the overall computational complexity. However, BOA does not use any problem-specific knowledge except for the evaluation of possible states of the system, whereas the specialized approaches fully rely on the knowledge of the problem structure and its properties. Even without requiring any problem-specific information in advance, hBOA provides competitive results. Furthermore, hBOA does not explicitly restrict the interaction structure of a problem; consequently, hBOA is applicable to spin glasses that fall outside the scope of specialized methods.

- After incorporating a discrete hill climber (DHC) based one-bit flips into hBOA, the performance of hBOA dramatically improves, yielding lower computational requirements than the best-known method. For example, for a system of 20×20 spins, hBOA+DHC required 9.7 minutes on average, while the best method needed about 25 minutes even on a faster computer.

- Since hBOA does not rely on the assumptions about the structure of spin-glass systems, the hybrid hBOA+DHC can also be applied to 3D spin glasses. There exists no method that solves 3D spin-glass systems in polynomial time. The results indicate that hBOA+DHC is capable of solving 3D spin glasses in $O(n^{3.65})$ evaluations.

- Maximum satisfiability of predicate logic formulas in conjunctive normal form (MAXSAT) is a well known problem of computational complexity and artificial intelligence. The task is to find an interpretation of predicates that maximizes the number of satisfied clauses in a given formula. A deterministic hill climber based on one-bit flips was used to improve the performance of hBOA. In MAXSAT, the deterministic hill climber based on one-bit flips comprises the so-called GSAT algorithm.

- On randomly generated 3-CNF instances from the phase transition, hBOA+GSAT outperforms GSAT alone, which is not capable of solving problems of even a moderate size. Therefore, hBOA is capable of supplying much better starting points for GSAT than random restarts do.

- On the same set of problem instances, hBOA+GSAT is outperformed by simple randomized hill climber (WalkSAT) that flips a random variable in an unsatisfied clause half of the time, and it performs the greedy step of GSAT otherwise. Therefore, a simple randomization of GSAT performs better than the sophisticated bias of hBOA+GSAT. Nonetheless, although hBOA uses selectorecombinative bias based on problem decomposition and hierarchical problem decomposition, it is capable of competing with local search on problem instances that are rather easy for local search.

- hBOA proves its robustness by being able to solve even those instances that are practically unsolvable by WalkSAT or Satz.

- Although there are problems for which hBOA is outperformed by specialized methods (and it is no surprise that this happens), note that the same algorithm was applied to onemax, composed traps, hierarchical traps, spin glasses, and MAXSAT. Empirical results over this wide spectrum of problems indicate that hierarchical BOA is robust not only within a specific class of problems, but it provides a scalable solution to an array of qualitatively different problem classes, as long as these problems are decomposable or hierarchically decomposable.

Chapter 8

Future Work

There are three promising avenues of future research on BOA and hBOA. The first focuses on improving the efficiency of BOA and hBOA by parallelization, hybridization, and other efficiency-enhancement techniques. The second focuses on the extensions of BOA and hBOA to problems defined outside the domain of fixed-length strings over a finite alphabet, such as vectors of variable length or program codes. The third focuses on extending the theory of BOA and hBOA. The purpose of this chapter is to outline most important topics in each of the three areas.

The chapter starts with an overview of approaches to enhance the efficiency of BOA and hBOA. Many efficiency-enhancement techniques can be adopted from GAs, but some techniques are directly related to BOA and hBOA. Section 8.2 discusses important issues for extending the applicability of BOA and hBOA to problems defined outside the domain of fixed-length finite-alphabet strings. Section 8.3 discusses important theoretical issues for developing a more complete theory of BOA and hBOA.

8.1 Enhancing the Efficiency

The results presented earlier indicate that BOA and hBOA are capable of solving difficult decomposable and hierarchically decomposable problems in a scalable manner. However, the polynomial complexity itself is sometimes insufficient for the practical application of the algorithm. There are two primary reasons for the concern about the practicality of polynomial-time optimizers (or any other algorithms). First, the constants hidden in $O(\cdot)$ notation often obstruct the practical use of computational methods for large problems. Second, even though the quadratic growth of the number of fitness evaluations seems promising, if the problem has thousands or tens of thousands variables, the quadratic growth leads to millions or hundreds of millions evaluations. Even if each evaluation could be done in one second, it would take more than 11 days to evaluate million solutions and it would take over 3 years to evaluate hundred million solutions. To successfully apply hierarchical BOA to such large and computationally intensive problems, it is necessary to enhance its efficiency as much as possible.

This section reviews the promising approaches to enhancing the efficiency of hierarchical BOA. Most of these enhancements are covered in the four-part harmony of GA efficiency enhancements studied at the Illinois Genetic Algorithms Laboratory (Goldberg, 2002; Sastry, 2001b; Albert, 2001).

8.1.1 Parallelization

The most straightforward way of making any algorithm work faster is to enable the algorithm to use multiple computers or CPUs. There are several approaches to parallelizing hierarchical BOA:

1. Distribute fitness evaluation.

2. Distribute model building.

3. Distribute model sampling.

4. Distribute the population.

Different approaches might be useful in different cases. If fitness evaluation is computationally intensive, a master-slave architecture can be used for distributing fitness evaluations and collecting the results (Cantú-Paz, 2000). If most of the computational time is spent in model building, model building should be parallelized by either (1) computing the frequencies in each step of the greedy algorithm in parallel (again, a master-slave architecture can be used), or (2) building several simpler models in parallel and combining the results afterwards. If large populations are necessary, a population might be distributed among multiple processors, each of which would process only a fraction of the entire population.

Many parallelization techniques and much of the theory can be adopted from GA research. One of the most important works in the area of GA parallelization was published by Cantú-Paz (2000), who provides useful theory and practical recommendations for ensuring that both the GAs run faster and the resources available in multiprocessor architectures are fully utilized. In the context of BOA, Ocenasek and Schwarz (2000) proposed the parallel BOA (pBOA) that distributes the model building using the MPI standard. Another approach to parallelizing the construction of Bayesian networks in the estimation of Bayesian network algorithm (EBNA) was discussed in Larranaga and Lozano (2002).

8.1.2 Hybridization

The basic reason for combining BOA or hBOA with local search—or hybridization of BOA and hBOA—is that by reducing the search space to the local optima in the problem, the structure of the problem can be easily identified and the population-sizing requirements can be significantly decreased. Furthermore, the search reduces to the space of "attractors" around each local optimum as opposed to the space of all potential solutions. The advantages of hybridization were demonstrated in the previous chapter, where the performance of hierarchical BOA was significantly improved with the addition of local search.

For the design of competent hybrid methods, it is necessary that the work is properly divided between the global and local searchers, so that the overall time complexity is minimized (Goldberg & Voessner, 1999; Goldberg, 2002; Sinha & Goldberg, 2002). When incorporating local search into BOA and hBOA, it is often advantageous to use local search more extensively in the first few generations to improve the initial probabilistic model as opposed to traditional GA practice. Once local optima are present in the current population, the probabilistic model should oftentimes be capable of sampling more local optima without the additional use of local search. Some theory considering the important issues for the design of efficient and scalable hybrids can be found in Goldberg and Voessner (1999), Goldberg (2002) and Sinha and Goldberg (2001). For survey of hybrid GAs, see Belew and Mitchell (1996) and Sinha and Goldberg (2002).

8.1.3 Time Continuation

There are two bounding ways of getting to the final solution to a given problem:

1. Use a big population for a small number of generations.

2. Use a small population and optimize the problem in multiple epochs.

In all the experiments presented in this thesis, we have taken the first approach and used one big population in a single run. However, sometimes it is advantageous to use a much smaller population and obtain only a part of the optimal solution at first, then perturb the solutions slightly, and run the algorithm for the second time (Goldberg, 1999c; Srivastava & Goldberg, 2001). Running BOA and hBOA for multiple epochs would significantly decrease the cost of model building and sampling. However, small populations might not be sufficient for discovering a satisfactory model.

Promising research directions related to time continuation should focus on answering the following questions. For what problems is it advantageous to run BOA and hBOA for multiple epochs with smaller populations? How to identify important features for making that distinction on the fly? Can a probabilistic model learned by BOA be used to make time-continuation approaches more effective in reducing the overall time complexity (e.g., by helping to design better perturbation operators)? What configuration (population size, number of epochs, perturbation methods) is the most efficient for specific classes of problems (decomposable, exponentially scaled, hierarchical)? For problems with high salience, where only a fraction of variables matters at any point (e.g., exponentially scaled deceptive problems), it can be expected that using multiple epochs is advantageous, because only partially correct models suffice. However, for problems with low salience or difficult hierarchical problems, dividing the search into multiple epochs could cause performance to deteriorate.

For more information regarding time continuation in GAs, see Goldberg (1999c), Goldberg (2002), and Srivastava and Goldberg (2001). Although the situation is different for BOA and hBOA, the results of the GA research might still suggest promising directions in approaching the aforementioned questions.

8.1.4 Prior Knowledge Utilization

In the experiments presented in this thesis, no prior knowledge about the problem was incorporated into BOA or hBOA. The reason for ignoring the prior knowledge was to provide sufficient evidence that both BOA and hBOA are capable of solving boundedly difficult problems in a scalable manner without the use of any prior information about the structure of the problem or its properties.

However, in real world, there is often much knowledge about the problem at hand, and it would be valuable to use all the information available. There are two straightforward ways of incorporating prior knowledge into BOA: (1) bias the initial population, and (2) bias model building.

The initial population of BOA and hBOA is usually generated at random with the *uniform* distribution. But in some cases it might be possible to incorporate the prior knowledge about the problem by biasing the generation of the initial population to ensure there is a better initial supply of good solutions. These approaches have proven useful in the domain of graph partitioning with BOA (Schwarz & Ocenasek, 2000) and silicon cluster optimization with ECGA (Sastry, 2001a). There are alternative ways of biasing the initial population. Local search can be used to improve the quality of candidate solutions in the initial population before the optimization starts. An expert can incorporate his or her knowledge by injecting promising partial or fully specified solutions in a fraction of the initial population. But in any case, the basic idea of biasing the initial population is to improve the quality of candidate solutions in the initial population to accelerate optimization.

Another approach to incorporating prior knowledge about the problem is to introduce additional bias in model building. There are two ways of incorporating prior knowledge into model building: (1) prior probabilities of structures, and (2) prior probabilities of partial solutions. If there is information available about the most likely interactions in the problem, the prior probability of those network structures that contain many of these interactions can be increased at the expense of other possible structures (see Section 3.3). An alternative way is to restrict the interactions in the model to those that are known to be important for solving the given problem. Prior probabilities of partial solutions can also be incorporated using Bayesian metrics; if good partial solutions are known, this information can lead to an increased quality of the model. Schwarz and Ocenasek (2000) discuss some of the interesting approaches to incorporating prior knowledge in the domain of graph partitioning. Important directions of future research should consider the aforementioned approaches to incorporating prior knowledge into BOA and hBOA and analyze the efficiency of the resulting approaches for specific classes of problems.

8.1.5 Fitness Evaluation Relaxation

On some problems—such as traps, hierarchical traps, spin glasses, and MAXSAT—the evaluation of each solution is a trivial task and thousands or tens of thousands of candidate solutions can be evaluated in a few seconds. However, to evaluate each solution in some problems, it might be necessary to simulate a stochastic process, execute an experiment in a wind tunnel, interact with the human operator, or use a finite element analysis. In that case, the fitness evaluation becomes prohibitively expensive for using population-based search with large populations, which are necessary for a successful application of BOA and hBOA.

To reduce the time complexity of fitness evaluation, an approximation of the fitness or a coarse-grained model can be used when possible (Albert, 2001; Sastry, 2001b). On one hand, using the approximation of the actual fitness decreases the computational time required to compute the fitness. On the other hand, the approximations usually lead to errors in the fitness, which can be biased or unbiased. Even if these errors are biased, the fitness approximation might still be very useful at the beginning of optimization, when the differences between the quality of different solutions in the population are much larger than the errors introduced by approximative fitness evaluation. However, later in the run the amount of error should be decreased, so that the algorithm converges to a true optimum (Albert, 2001; Sastry, 2001b). If errors introduced by an approximative model are unbiased, the overall computational complexity using each possible model must be considered to determine the best model to use (Miller & Goldberg, 1996b; Albert, 2001; Sastry, 2001b).

The approximation of the fitness can be obtained by modifying the parameters of the simulation or building an internal model (e.g., a neural network) for evaluating a specified fraction of the population (Albert, 2001; Sastry, 2001b). For example, the stochastic simulation can be run for a certain period of time, and the more time the simulation gets, the more accurate the final results are. In the finite element analysis, the mesh approximation can range from fine to coarse, affecting the accuracy of the result. Another way of approximating fitness is to build an internal model of the fitness incrementally. In that case, the GA can learn a neural network, a linear model, or a quadratic model of the fitness online, and use the model to approximate the fitness of a specified fraction of the population (Sastry, Goldberg, & Pelikan, 2001; Sastry, 2001b).

While in GAs, approximate fitness evaluation affects only the decision making, in BOA and hBOA the model building is affected as well. Important questions include the following. What are the tradeoffs between the time saved by approximate fitness evaluation and the time lost by using an "imperfect" fitness to learn a model? How does the situation change compared to GAs? How can the model learned by BOA be used to learn a better internal model of the fitness on the fly?

8.1.6 Incremental and Sporadic Model Building

In our implementation of BOA and hBOA, the model is built from scratch in each generation. However, as was suggested by Etxeberria and Larrañaga (1999), an incremental model building can reduce the time complexity of model building in subsequent generations. Furthermore, performance can be improved by learning the structure of a model sporadically at certain intervals, while updating only the parameters of the model in the remaining generations.

If the model is a traditional Bayesian network, it is fairly straightforward to build the model incrementally. However, if Bayesian networks with decision graphs are used, the task of updating the model becomes much more complicated. One important topic of future research is to provide an efficient technique for updating Bayesian networks with decision graphs on the fly. Of course, incremental learning must be more efficient than building a model from scratch.

To further reduce the time complexity of model building, structural learning can be performed only at certain intervals, while in the remaining time only the conditional probabilities in a model must be updated. The update of conditional probabilities can be obtained by one pass through the population of promising solutions. Learning the structure of a model less frequently should significantly reduce the computational cost of model building. However, if a bad structure is learned in some generation, this structure will affect subsequent generations in addition to the current one. It is important that the potential savings in model building will not endanger convergence to the optimum. Moreover, the time saved by learning the model less frequently must be lower than the time lost by using a model that is not up-to-date.

8.2 Extending the Applicability

BOA and hBOA are applicable to problems where the solutions are represented by fixed-length strings over a finite alphabet and the quality of each solution is expressed as a real number. How can we apply the two algorithms to problems where the solutions are represented by variable-length strings, vectors of real numbers, or program codes represented by tree structures? How can we extend the two algorithms to take into account multiple objectives at the same time to determine the tradeoff between the objectives? A promising area of future research on BOA and hBOA deals with the above two questions in order to extend the applicability of BOA and hBOA. This section discusses possible directions for extending BOA and hBOA to incorporate multiobjective optimization and other than fixed-length finite-alphabet strings.

8.2.1 Multiobjective Optimization

This thesis considered the problems with a single objective. However, in many engineering problems there are multiple criteria or objectives. Consider designing a radiator in the heat exchanger with two objectives: (1) minimize the size of the radiator, and (2) maximize the heat exchange. Clearly, there is a *tradeoff* between the two objectives; the smaller the radiator, the less heat is exchanged.

One way of dealing with several objectives is to weight the objectives, and compute the fitness of each solution as a weighted average of the fitnesses with respect to each objective. In that case, single-objective optimization methods can be applied to solve the weighted problem. However, it is often impossible to find proper weights without knowing what the real tradeoff is. That is why it is often necessary to take into account both the objectives. If the solution is outperformed by any other solution in both the objectives, the solution is not useful, because it is dominated by the other solution. However, if the solution outperforms any other solution in at least one objective, then the solution might be useful in the end. Of course, the scheme can be generalized to more than two objectives. Multiobjective optimization attempts to find these solutions that are better than other solutions in at least one objective, comprising the so-called *Pareto-optimal front*.

The primary task of multiobjective optimization techniques is to find a *diverse* Pareto-optimal front. In other words, the task is to find the widest tradeoff curve possible, so that it both contains good solutions and it allows the user to choose from a wide variety of solutions. Extending BOA to multiobjective problems is straightforward and a lot can be adopted from GA research for multiobjective optimization (Deb, 2001). However, hBOA has its own niching mechanism and it's not as easy to incorporate a Pareto-front related niching into the scheme. One of the important research directions is to combine solution-based niching of hBOA and Pareto-based niching of multiobjective GAs to ensure that both hard problems can be solved and a diverse Pareto front is discovered.

A thorough overview of the most recent developments in multiobjective optimization can be found in (Deb, 2001). Thierens and Bosman (2001) incorporated multiobjective optimization techniques in the IDEA framework based on mixture distributions. Khan (2002) combined the advanced methods of multiobjective optimization with BOA and proposed a set of multiobjective problems that require both the automatic discovery of a proper problem decomposition as well as the maintenance of a diverse Pareto front.

8.2.2 From Binary Strings to Computer Programs

BOA and hBOA can optimize problems defined over fixed-length strings over a finite alphabet. How to apply BOA or hBOA if the problem at hand is defined over vectors of real numbers, variable-length binary vectors, or graph structures? Section 2.4 identified the two basic approaches to extending BOA and hBOA to other domains:

1. Map the other domain to the domain of fixed-length discrete strings.

2. Extend or modify the model from the discrete fixed-length strings to other domains.

The first approach is straightforward; a mapping function must be designed that transforms every candidate solution from the other domain to the domain of fixed-length binary (or k-ary) strings. The mapping does not have to be one-to-one; groups or clusters of candidate solutions in other domains can be mapped into the same solution. However, it is necessary that the solutions that are mapped into one string are very similar and can be indeed represented as one member of the search space.

The second approach is to modify the model to the new domain. Section 2.4 discussed some of the models that can be used in other domains. Mixtures of normal distributions, products of mixtures, probabilistic trees, and other models can be used. Each of these models can be superior to other ones in some cases, and it is crucial to identify the classes of problems that the models can solve efficiently, reliably, and accurately.

One of the most promising lines of research is to extend BOA and hBOA to the domain of computer programs of genetic programming. Clearly, competent genetic programming techniques could be applied to a wide variety of domains using only little problem specific knowledge by evolving populations of computer programs. There is an ongoing discussion on the scalability of traditional genetic programming, but many experimental results suggest that extremely large populations must be used to achieve competitive results. That suggests that genetic programming might be facing the same problem as traditional genetic algorithms do, and that there is a need for some form of automatic identification of the regularities in the problem at hand. Once the regularities are identified, these can be exploited to ensure a scalable optimization within the framework of genetic programming.

8.3 Developing Additional Theory

Although the most important goal of the design of BOA and hBOA was to provide a competent adaptive black-box optimization method that can exploit decomposition and hierarchical decomposition, the development of theory is important for identifying the classes of problems that can be reliably solved by BOA and hBOA in an efficient and scalable manner and providing practical recommendations for setting the different parameters of the algorithms. The theory presented in chapters 4 and 6 suggested that on decomposable problems of bounded difficulty, the total number of fitness evaluations should grow subquadratically or quadratically with the size of the problem.

However, the theory presented in this thesis represents only the first few steps toward a complete theory that takes into account all the factors that can be encountered in real-world problems. These factors include decomposition where the subproblems interact in some way, decomposition where the order of subproblems grows with the size of the problem, the effects of using the restricted tournament selection as a replacement strategy, and others. Probably the most important theoretical issue is the one related to *overlap* in decomposable problems. Some sufficient conditions exist for determining good models with overlap (Mühlenbein, Mahnig, & Rodriguez, 1999), but there is much work to be done for a better and more practical theory. Furthermore, it is important to relate the sufficient conditions for a good problem decomposition for decomposable problems with much overlap to the theory considering model building similar to that presented in chapter 4.

Finally, an interesting area of future research on BOA and hBOA is in designing problems that *deceive* model building in some way. Similarly as composed traps helped make GAs better by making them fail, designing problems that are currently beyond the class of problems solvable by BOA and hBOA can help make BOA and hBOA more powerful.

Chapter 9

Conclusions

The purpose of this chapter is to provide the summary and main conclusions of this dissertation. First, the contributions of the dissertation are summarized. Next, the main conclusions of the dissertation are provided.

9.1 What Has Been Done

A summary of the major results of this thesis follows:

Bayesian optimization algorithm (BOA) The thesis proposed the Bayesian optimization algorithm (BOA), which replaces traditional variation operators of GAs by (1) building a Bayesian network for promising solutions and (2) sampling new solutions according to the built network. Building the network allows for automatic discovery of regularities in the problem at hand. Sampling the constructed network allows for effective exploitation of encoded regularities. BOA can solve hard problems of bounded difficulty in subquadratic or quadratic time with respect to the number of candidate solutions that must be evaluated until the algorithm converges to the optimum.

Scalability theory of BOA A theory was developed that estimates the number of fitness evaluations until convergence of BOA on problems of bounded difficulty. The number of fitness evaluations was computed by (1) approximating an adequate population size for reliable convergence to the optimum, (2) estimating the number of generations until convergence, and (3) making a product of these two quantities. The theory confirmed that the number of evaluations until convergence to the optimum for problems of bounded difficulty grows subquadratically or quadratically with the problem size, depending on the type of the problem.

Hierarchy for complexity reduction Hierarchical decomposition allows for a scalable solution of those problems that are not decomposable on a single level but that can be solved by decomposition over multiple levels of difficulty. The thesis identified three important features that must be incorporated into optimizers that exploit problem decomposition to ensure that the algorithms can solve difficult hierarchical problems in a scalable manner. The issues comprise the three keys to hierarchy success: (1) proper decomposition, (2) chunking, and (3) preservation of alternative solutions. On each level, the problem must be decomposed properly so that the algorithm is not misled by salient nonlinearities on the current level. Best partial solutions of the subproblems on each level must be represented in a compact way and a mechanism for chunking partial solutions must be introduced so that these can be juxtaposed effectively on higher levels. Finally, alternative partial solutions to the subproblems on each level of decomposition must be preserved until it becomes possible to eliminate some of the alternatives.

Hierarchical traps In the development of competent optimization methods, it is important to design a class of problems that can be used to test developed techniques on the boundary of their design envelope. The design of difficult hierarchical problems can be guided by the three keys to hierarchy success, so that any algorithm that is incapable of tackling the three keys to hierarchy success will fail at solving the proposed class of problems efficiently. The thesis proposed hierarchical traps to provide a class of difficult hierarchical problems that can be used to test optimization methods that attempt to exploit hierarchical decomposition. Hierarchical traps are practically unsolvable if the three keys to hierarchy success are not dealt with properly.

Hierarchical BOA Hierarchical BOA extends BOA by (1) using local structures to represent local probability distributions in Bayesian networks more compactly and (2) using restricted tournament replacement to incorporate new candidate solutions into the original population of candidate solutions. Hierarchical BOA was shown to solve hierarchical traps and other hard problems in a subquadratic or quadratic number of evaluations.

Experiments on artificial and real-world problems A number of experiments to verify the scalability of BOA and hBOA were performed. Three fundamentally different sets of problems were used: (1) boundedly difficult decomposable problems, (2) difficult hierarchical problems, and (3) real-world problems. Empirical results on boundedly-difficult problems—onemax, composed traps, and composed deceptive functions—indicated that BOA can solve the class of problems decomposable into subproblems of bounded order in a scalable manner. Empirical results on hierarchically difficult problems—hierarchical traps and HIFF—indicated that hierarchical BOA is capable of solving difficult hierarchical problems in a scalable manner. Empirical results on two real-world problems—2D and 3D Ising spin-glass systems and MAXSAT—showed that hierarchical BOA can not only compete with state-of-the-art methods that are designed to solve a specific class of problems, but that it can oftentimes outperform those methods. A hybrid algorithm comprising of hierarchical BOA and a simple local searcher has proven to be advantageous in real-world applications.

Future work on BOA, hBOA, and PMBGAs The thesis gave a number of suggestions for future research on BOA, hBOA, and other PMBGAs. There are three important research directions. The first focuses on enhancing the efficiency; the second considers improving the applicability of BOA and hBOA; the last deals with developing additional theory and test problems.

9.2 Main Conclusions

A scalable black-box optimization algorithm capable of automatic discovery and effective exploitation of single-level and hierarchical problem decomposition exists and is ready for application. The Bayesian optimization algorithm (BOA) and the hierarchical Bayesian optimization algorithm (hBOA) can solve a broad class of problems as long as these problems are decomposable on a single level or multiple levels of difficulty. For a successful application of BOA and hBOA, no prior knowledge about the problem is required except for the (1) number of decision variables and (2) a measure for evaluating candidate solutions.

There are only a few parameters that must be set in BOA and hBOA. In fact, the only parameter that really matters is the population size, because all the remaining parameters change the performance only marginally. The population size can also be eliminated by using the parameter-less population-sizing scheme of Harik and Lobo (1999). Therefore, the developed algorithms can be applied to the problem at hand without knowing much about the problem itself and without worrying about thresholds or any other parameters.

Although BOA and hBOA do not rely on problem-specific knowledge, it is fairly straightforward to incorporate prior knowledge about the problem into the algorithms to further improve their efficiency. Prior knowledge of various forms can be incorporated; the initial population can be biased according to the expert knowledge or using other search, optimization, and machine learning techniques; the structure of the problem can be incorporated into the model building by favoring models similar to a suggested structure or set of structures; finally, promising partial solutions can be incorporated into the measure for discriminating competing probabilistic models.

For practitioners seeking robust, efficient, and scalable optimization techniques, BOA and hBOA represent a significant step forward; BOA and hBOA do not require the user to be an expert in the field of optimization, nor do they require the user to provide complete information about the structure of the problem and its properties. Despite that, BOA and hBOA can achieve competitive or better performance compared to specialized methods that fully rely on problem-specific knowledge. Furthermore, prior problem-specific knowledge can be incorporated at no extra cost, and the algorithms can be easily combined with specialized local searchers.

BOA and hBOA should have a large impact on the research in genetic and evolutionary computation and computational optimization in general. First of all, BOA provides a powerful solution to linkage learning in genetic and evolutionary algorithms. Second, hierarchical BOA extends the basic approach to linkage learning to solve difficult hierarchical problems, which are practically unsolvable by other known optimization methods. Finally, BOA and hBOA replace specialized heuristics that are often incorporated into genetic and evolutionary optimization techniques to improve their performance on difficult problems by rigorous methods of statistics and probability theory.

Bibliography

Ackley, D. H. (1987). An empirical study of bit vector function optimization. *Genetic Algorithms and Simulated Annealing*, 170–204.

Albert, L. A. (2001). *Efficient genetic algorithms using discretization scheduling*. Master's thesis, University of Illinois at Urbana-Champaign, Department of General Engineering, Urbana, IL.

Asoh, H., & Mühlenbein, H. (1994). On the mean convergence time of evolutionary algorithms without selection and mutation. *Parallel Problem Solving from Nature*, 88–97.

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. *Proceedings of the International Conference on Genetic Algorithms (ICGA-85)*, 101–111.

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

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

Baluja, S., & Davies, S. (1998). Fast probabilistic modeling for combinatorial optimization. *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 469–476.

Belew, R. K., & Mitchell, M. (Eds.) (1996). *Adaptive individuals in evolving populations: Models and algorithms*. Reading, MA: Addison Wesley.

Booker, L. B. (1982). *Intelligent behavior as an adaptation to the task environment*. Doctoral dissertation, The University of Michigan. (University Microfilms No. 8214966).

Bosman, P., & Thierens, D. (2001). Exploiting gradient information in continuous iterated density estimation evolutionary algorithms. *Proceedings of the Belgium-Netherlands Conference on Artificial Intelligence (BNAIC-2001)*, 69–76.

Bosman, P. A., & Thierens, D. (2000a). Continuous iterated density estimation evolutionary algorithms within the IDEA framework. *Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, 197–200.

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

Bosman, P. A. N., & Thierens, D. (1999). Linkage information processing in distribution estimation algorithms. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, *I*, 60–67.

Cantú-Paz, E. (2000). *Efficient and accurate parallel genetic algorithms*. Boston, MA: Kluwer.

Cantú-Paz, E. (2001). Supervised and unsupervised discretization methods for evolutionary algorithms. *Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 213–216.

Cavicchio, Jr., D. J. (1970). *Adaptive search using simulated evolution*. Unpublished doctoral dissertation, University of Michigan, Ann Arbor, MI. (University Microfilms No. 25-0199).

Ceroni, A., Pelikan, M., & Goldberg, D. E. (2001). *Convergence-time models for the simple genetic algorithm with finite population* (IlliGAL Report No. 2001028). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Cheeseman, P., Kanefsky, B., & Taylor, W. M. (1991). Where the really hard problems are. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-91)*, 331–337.

Chickering, D. M., Geiger, D., & Heckerman, D. (1994). *Learning Bayesian networks is NP-hard* (Technical Report MSR-TR-94-17). Redmond, WA: Microsoft Research.

Chickering, D. M., Heckerman, D., & Meek, C. (1997). *A Bayesian approach to learning Bayesian networks with local structure* (Technical Report MSR-TR-97-07). Redmond, WA: Microsoft Research.

Chow, C., & Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, *14*, 462–467.

Cohoon, J. P., Hegde, S. U., Martin, W. N., & Richards, D. (1987). Punctuated equilibria: A parallel genetic algorithm. *Proceedings of the International Conference on Genetic Algorithms (ICGA-87)*, 148–154.

Collins, R. J., & Jefferson, D. R. (1991). Selection in massively parallel genetic algorithms. *Proceedings of the International Conference on Genetic Algorithms (ICGA-91)*, 249–256.

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

Culberson, J. C. (1992). *Genetic invariance: A new paradigm for genetic algorithm design.* Unpublished manuscript.

Davidor, Y. (1991). A naturally occuring niche and species phenomenon: The model and first results. *Proceedings of the International Conference on Genetic Algorithms (ICGA-91)*, 257–263.

Davies, S., & Moore, A. (1999). Using Bayesian networks for lossless compression in data mining. *Proceedings of the International Conference on Knowledge Discovery & Data Mining (KDD-99)*, 387–391.

Davis, M., Logemann, G., & Loveland, D. (1962). A machine program for theorem proving. *Communications of the ACM*, *5*(7), 394–397.

De Bonet, J. S., Isbell, C. L., & Viola, P. (1997). MIMIC: Finding optima by estimating probability densities. *Advances in neural information processing systems (NIPS-97)*, *9*, 424–431.

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

De Simone, C., Diehl, M., Jünger, M., & Reinelt, G. (1996). Exact ground states of two-dimensional $+-$J Ising spin glasses. *Journal of Statistical Physics*, *84*, 1363–1371.

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

Deb, K., & Goldberg, D. E. (1989). An investigation of niche and species formation in genetic function optimization. *Proceedings of the International Conference on Genetic Algorithms (ICGA-89)*, 42–50.

Deb, K., & Goldberg, D. E. (1991). *Analyzing deception in trap functions* (IlliGAL Report No. 91009). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Edmonds, J. (1967). Optimum branching. *J. Res. Nat. Bur. Standards*, *71B*, 233–240.

Eldredge, N., & Gould, S. J. (1972). Punctuated equilibria: An alternative to phyletic gradualism. In Schopf, T. (Ed.), *Paleobiology* (pp. 82–115). San Francisco, CA: Freeman & Company.

Etxeberria, R., & Larrañaga, P. (1999). Global optimization using Bayesian networks. In Rodriguez, A. A. O., Ortiz, M. R. S., & Hermida, R. S. (Eds.), *Second Symposium on Artificial Intelligence (CIMAF-99)* (pp. 332–339). Habana, Cuba: Institude of Cybernetics, Mathematics, and Physics and Ministry of Science, Technology and Environment.

Feller, W. (1970). *An introduction to probability theory and its applications*. New York, NY: Wiley.

Fonseca, C. M., & Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. *Proceedings of the International Conference on Genetic Algorithms (ICGA-93)*, 416–423.

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

Friedman, N., & Yakhini, Z. (1996). On the sample complexity of learning Bayesian networks. *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI-96)*, 274–282.

Gallagher, M., Frean, M., & Downs, T. (1999, 13-17 July). Real-valued evolutionary optimization using a flexible probability density estimator. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, *1*, 840–846.

Galluccio, A., & Loebl, M. (1999a). A theory of Pfaffian orientations. I. Perfect matchings and permanents. *Electronic Journal of Combinatorics*, *6*(1). Research Paper 6.

Galluccio, A., & Loebl, M. (1999b). A theory of Pfaffian orientations. II. T-joins, k-cuts, and duality of enumeration. *Electronic Journal of Combinatorics*, *6*(1). Research Paper 7.

Geiger, D., & Heckerman, D. (1996). Beyond Bayesian networks: Similarity networks and Bayesian multinets. *Artificial Intelligence*, *82*, 45–74.

Geiger, D., Heckerman, D., & Meek, C. (1996). Asymptotic model selection for directed networks with hidden variables. *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI-96)*, 158–168.

Gent, I., Hoos, H. H., Prosser, P., & Walsh, T. (1999). Morphing: Combining structure and randomness. *Proceedings of the American Association of Artificial Intelligence (AAAI-99)*, 654–660.

Goldberg, D. E. (1983). Computer-aided gas pipeline operation using genetic algorithms and rule learning. *Dissertation Abstracts International*, *44*(10), 3174B. Doctoral dissertation, University of Michigan.

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

Goldberg, D. E. (1989b). Sizing populations for serial and parallel genetic algorithms. *Proceedings of the International Conference on Genetic Algorithms (ICGA-89)*, 70–79. Also IlliGAL Report No. 88004.

Goldberg, D. E. (1994). *First flights at genetic-algorithm Kitty Hawk* (IlliGAL Report No. 94008). Urbana, IL: University of Illinois at Urbana-Champaign.

Goldberg, D. E. (1998, June 15). Four keys to understanding building-block difficulty. Presented in Projet FRACTALES Seminar at I.N.R.I.A. Rocquencourt, Le Chesnay, Cedex.

Goldberg, D. E. (1999a). *Genetic and evolutionary algorithms in the real world* (IlliGAL Report No. 99013). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

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

Goldberg, D. E. (1999c). Using time efficiently: Genetic-evolutionary algorithms and the continuation problem. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, 212–219. Also IlliGAL Report No. 99002.

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

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., & Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. *Proceedings of the International Conference on Genetic Algorithms (ICGA-87)*, 41–49.

Goldberg, D. E., & Rudnick, M. (1991). Genetic algorithms and the variance of fitness. *Complex Systems*, *5*(3), 265–278. Also IlliGAL Report No. 91001.

Goldberg, D. E., Sastry, K., & Latoza, T. (2001). On the supply of building blocks. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 336–342. Also IlliGAL Report No. 2001015.

Goldberg, D. E., & Segrest, P. (1987). Finite Markov chain analysis of genetic algorithms. *Proceedings of the International Conference on Genetic Algorithms (ICGA-87)*, 1–8.

Goldberg, D. E., & Voessner, S. (1999). Optimizing global-local search hybrids. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, 220–228. Also IlliGAL Report No. 99001.

Gonzalez, C., Lozano, J., & Larranaga, P. (2001). Analyzing the PBIL algorithm by means of discrete dynamical systems. *Complex Systems*, *4*(12), 465–479.

Gorges-Schleuter, M. (1989). ASPARAGOS: An asynchronous parallel genetic optimization strategy. *Proceedings of the International Conference on Genetic Algorithms (ICGA-89)*, 422–428.

Gottlieb, J., Marchiori, E., & Rossi, C. (2002). Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, *10*(1), 35–50.

Grosso, P. B. (1985). *Computer simulations of genetic adaptation: Parallel subcomponent interaction in a multilocus model.* Unpublished doctoral dissertation, The University of Michigan. (University Microfilms No. 8520908).

Grünwald, P. (1998). *The minimum description length principle and reasoning under uncertainty.* Doctoral dissertation, University of Amsterdam, Amsterdam, Netherlands.

Handley, S. (1994). On the use of a directed acyclic graph to represent a population of computer programs. *Proceedings of the International Conference on Evolutionary Computation (ICEC-94)*, 154–159.

Hansen, N., Ostermeier, A., & Gawelczyk, A. (1995). On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. *Proceedings of the International Conference on Genetic Algorithms (ICGA-95)*, 57–64.

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

Harik, G., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1999). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, *7*(3), 231–253.

Harik, G., & Lobo, F. (1999). A parameter-less genetic algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, *I*, 258–265.

Harik, G. R. (1994). *Finding multiple solutions in problems of bounded difficulty* (IlliGAL Report No. 94002). Urbana, IL: University of Illinois at Urbana-Champaign.

Harik, G. R. (1997). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms.* Doctoral dissertation, University of Michigan, Ann Arbor. Also IlliGAL Report No. 97005.

Harik, G. R., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Proceedings of the International Conference on Evolutionary Computation (ICEC-97)*, 7–12. Also IlliGAL Report No. 96004.

Harik, G. R., Lobo, F. G., & Goldberg, D. E. (1997). *The compact genetic algorithm* (IlliGAL Report No. 97006). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

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

Henrion, M. (1988). Propagation of uncertainty in Bayesian networks by logic sampling. In Lemmer, J. F., & Kanal, L. N. (Eds.), *Uncertainty in Artificial Intelligence* (pp. 149–163). Amsterdam, London, New York: Elsevier.

Höhfeld, M., & Rudolph, G. (1997). Towards a theory of population-based incremental learning. *Proceedings of the International Conference on Evolutionary Computation (ICEC-97)*, 1–6.

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

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

Holland, J. H. (2000). Building blocks, cohort genetic algorithms, and hyperplane-defined functions. *Evolutionary Computation*, *8*(4), 373–391.

Hollstein, R. B. (1971). *Artificial genetic adaptation in computer control systems*. Doctoral dissertation, University of Michigan. (University Microfilms No. 71-23,773).

Horn, J. (1993). Finite Markov chain analysis of genetic algorithms with niching. *Proceedings of the International Conference on Genetic Algorithms (ICGA-93)*, 110–117. Also IlliGAL Report No. 93002.

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

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

Kardar, M., & Saul, L. (1994). The 2D +/−J Ising spin glass: Exact partition functions in polynomial time. *Nucl. Phys. B*, *432*, 641–667.

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

Khan, N. (2002). *Multiobjective Bayesian optimization algorithm* (Technical Report). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory. Unpublished technical report.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection.* Cambridge, MA: The MIT Press.

Koza, J. R. (1994). *Genetic programming II: Automatic discovery of reusable programs.* Cambridge, MA: Massachusetts Institute of Technology.

Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *Annals of Math. Stats.*, *22*, 79–86.

Kvasnicka, V., Pelikan, M., & Pospichal, J. (1996). Hill climbing with learning (An abstraction of genetic algorithm). *Neural Network World*, *6*, 773–796.

Larranaga, P., Etxeberria, R., Lozano, J., & Pena, J. (2000a). Combinatorial optimization by learning and simulation of bayesian networks. *Proceedings of the Uncertainty in Artificial Intelligence (UAI-2000)*, 343–352.

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

Larranaga, P., & Lozano, J. A. (Eds.) (2002). *Estimation of distribution algorithms: A new tool for evolutionary computation.* Boston, MA: Kluwer.

Li, C. M., & Anbulagan (1997). Heuristics based on unit propagation for satisfiability problems. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-97)*, 366–371.

Lobo, F. G., Goldberg, D. E., & Pelikan, M. (2000). Time complexity of genetic algorithms on exponentially scaled problems. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, 151–158.

Loebl, M. (2000). *On the dimer problem in 3-dimensional lattices* (Technical Report). Prague, Czech Republic: Department of Applied Mathematics, Charles University.

Mahfoud, S. W. (1992). Crowding and preselection revisited. *Parallel Problem Solving from Nature*, 27–36.

Marascuilo, L. A., & McSweeney, M. (1977). *Nonparametric and distribution-free methods for the social sciences*. CA: Brooks/Cole Publishing Company.

Mauldin, M. L. (1984). Maintaining diversity in genetic search. In Brachman, R. J. (Ed.), *Proceedings of the National Conference on Artificial Intelligence* (pp. 247–250). Austin, TX: William Kaufmann.

Mengshoel, O. J., & Goldberg, D. E. (1999). Probabilistic crowding: Deterministic crowding with probabilisitic replacement. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, *I*, 409–416.

Miller, B. L., & Goldberg, D. E. (1996a). Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, *4*(2), 113–131.

Miller, B. L., & Goldberg, D. E. (1996b). Optimal sampling for genetic algorithms. *Intelligent Engineering Systems through Artificial Neural Networks*, *6*, 291–297.

Mitchell, M. (1996). *An introduction to genetic algorithms*. Cambridge, MA: MIT Press.

Mitchell, M., Forrest, S., & Holland, J. H. (1992). The royal road for genetic algorithms: Fitness landscapes and GA performance. *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, 245–254.

Monien, B., & Sudborough, I. H. (1988). Min cut is np-complete for edge weighted trees. *Theoretical Computer Science*, *58*(1–3), 209–229.

Mühlenbein, H. (1991). Evolution in time and space-The parallel genetic algorithm. *Foundations of Genetic Algorithms*, 316–337.

Mühlenbein, H. (1992). How genetic algorithms really work: I.Mutation and Hillclimbing. *Parallel Problem Solving from Nature*, 15–25.

Mühlenbein, H. (1997). The equation for response to selection and its use for prediction. *Evolutionary Computation*, *5*(3), 303–346.

Mühlenbein, H., & Mahnig, T. (1998). Convergence theory and applications of the factorized distribution algorithm. *Journal of Computing and Information Technology*, *7*(1), 19–32.

Mühlenbein, H., & Mahnig, T. (1999). FDA – A scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, *7*(4), 353–376.

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

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

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

Naudts, B., & Naudts, J. (1998). The effect of spin-flip symmetry on the performance of the simple GA. *Parallel Problem Solving from Nature*, 67–76.

Ocenasek, J., & Schwarz, J. (2000). The parallel Bayesian optimization algorithm. In *Proceedings of the European Symposium on Computational Inteligence* (pp. 61–67). Physica-Verlag.

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

Pelikan, M., & Goldberg, D. E. (2000a). Genetic algorithms, clustering, and the breaking of symmetry. *Parallel Problem Solving from Nature*, 385–394. Also IlliGAL Report No. 2000013.

Pelikan, M., & Goldberg, D. E. (2000b). Hierarchical problem solving by the Bayesian optimization algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, 275–282. Also IlliGAL Report No. 2000002.

Pelikan, M., & Goldberg, D. E. (2001). Escaping hierarchical traps with competent genetic algorithms. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 511–518. Also IlliGAL Report No. 2000020.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1998). *Linkage problem, distribution estimation, and Bayesian networks* (IlliGAL Report No. 98013). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

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

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (2000a). Bayesian optimization algorithm, population sizing, and time to convergence. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, 275–282. Also IlliGAL Report No. 2000001.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (2000b). Linkage problem, distribution estimation, and Bayesian networks. *Evolutionary Computation*, *8*(3), 311–341. Also IlliGAL Report No. 98013.

Pelikan, M., Goldberg, D. E., & Lobo, F. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, *21*(1), 5–20. Also IlliGAL Report No. 99018.

Pelikan, M., Goldberg, D. E., & Sastry, K. (2001). Bayesian optimization algorithm, decision graphs, and Occam's razor. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 519–526.

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

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

Pelikan, M., Sastry, K., & Goldberg, D. E. (2001). *Evolutionary algorithms + graphical models = scalable black-box optimization* (IlliGAL Report No. 2001029). Urbana, IL: Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

Perry, Z. A. (1984). Experimental study of speciation in ecological niche theory using genetic algorithms. *Dissertation Abstracts International*, *45*(12), 3870B. (University Microfilms No. 8502912).

Poli, R., Langdon, W., & O'Reilly, U.-M. (1998). Analysis of schema variance and short term extinction likelihoods. *Proceedings of the Genetic Programming Conference (GP-98)*, 284–292.

Prim, R. (1957). Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, *36*, 1389–1401.

Rana, S., & Whitley, D. L. (1998). Genetic algorithm behavior in the MAXSAT domain. *Parallel Problem Solving from Nature*, 785–794.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Frommann-Holzboog.

Rechenberg, I. (1994). *Evolutionsstrategie '94*. Stuttgart: Frommann-Holzboog Verlag.

Reeves, C. (1993). Using genetic algorithms with small populations. *Proceedings of the International Conference on Genetic Algorithms (ICGA-93)*, 92–99.

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

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

Rissanen, J. J. (1996). Fisher information and stochastic complexity. *IEEE Transactions on Information Theory*, *42*(1), 40–47.

Rothlauf, F. (2001). *Towards a theory of representations for genetic and evolutionary algorithms: Development of basic concepts and their application to binary and tree representations.* Doctoral dissertation, University of Bayreuth, Beyreuth, Germany.

Rothlauf, F., Goldberg, D. E., & Heinzl, A. (2000). *Bad codings and the utility of well-designed genetic algorithms* (IlliGAL Report No. 200007). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Rudlof, S., & Köppen, M. (1996). Stochastic hill climbing with learning by vectors of normal distributions. Nagoya, Japan.

Rudnick, M. W. (1992). *Genetic algorithms and fitness variance with application to automated design of artificial neural networks.* Doctoral dissertation, Oregon Graduate Institute of Science & Technology, Beaverton, OR.

Salustowicz, R., & Schmidhuber, J. (1998). *H-PIPE: Facilitating hierarchical program evolution through skip nodes* (Technical Report IDSIA-08-98). Lugano, Switzerland: Instituto Dalle Molle di Studi sull' Intelligenza Artificiale (IDSIA).

Salustowicz, R. P., & Schmidhuber, J. (1997a). Probabilistic incremental program evolution. *Evolutionary Computation*, *5*(2), 123–141.

Salustowicz, R. P., & Schmidhuber, J. (1997b). Probabilistic incremental program evolution: Stochastic search through program space. *Proceedings of the European Conference of Machine Learning (ECML-97)*, *1224*, 213–220.

Sastry, K. (2001a). *Efficient atomic cluster optimization using a hybrid extended compact genetic algorithm with seeded population* (IlliGAL Report No. 2001018). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Sastry, K. (2001b). *Evaluation-relaxation schemes for genetic and evolutionary algorithms.* Master's thesis, University of Illinois at Urbana-Champaign, Department of General Engineering, Urbana, IL. Also IlliGAL Report No. 2002004.

Sastry, K., & Goldberg, D. E. (2000). *On extended compact genetic algorithm* (IlliGAL Report No. 2000026). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Sastry, K., Goldberg, D. E., & Pelikan, M. (2001). Don't evaluate, inherit. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 551–558.

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

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, *6*, 461–464.

Schwarz, J., & Ocenasek, J. (1999). Experimental study: Hypergraph partitioning based on the simple and advanced algorithms BMDA and BOA. *Proceedings of the International Conference on Soft Computing*, 124–130.

Schwarz, J., & Ocenasek, J. (2000). A problem-knowledge based evolutionary algorithm KBOA for hypergraph partitioning. In *Proceedings of the Fourth Joint Conference on Knowledge-Based Software Engineering* (pp. 51–58). Brno, Czech Republic: IO Press.

Schwefel, H.-P. (1977). *Numerische Optimierung von Computer–Modellen mittels der Evolutionsstrategie*, Volume 26 of *Interdisciplinary Systems Research.* Basle, Switzerland: Birkhäuser.

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

Selman, B., Levesque, H. J., & Mitchell, D. (1992). A new method for solving hard satisfiability problems. *Proceedings of the National Conference on Artificial Intelligence (AAAI-92)*, 440–446.

Servet, I., Trave-Massuyes, L., & Stern, D. (1997). Telephone network traffic overloading diagnosis and evolutionary computation techniques. *Proceedings of the European Conference on Artificial Evolution (AE-97)*, 137–144.

Simon, H. A. (1968). *The sciences of the artificial.* Cambridge, MA: The MIT Press.

Sinha, A., & Goldberg, D. E. (2001). Verification and extension of the theory of global-local hybrids. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 591–597. Also IlliGAL Report No. 2001010.

Sinha, A., & Goldberg, D. E. (2002). *A survey of hybrid genetic and evolutionary algorithms* (IlliGAL Report No. 2002XXX). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Srivastava, R., & Goldberg, D. E. (2001). Verification of the theory of genetic and evolutionary continuation. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 551–558. Also IlliGAL Report No. 2001007.

Thierens, D. (1995). *Analysis and design of genetic algorithms*. Doctoral dissertation, Katholieke Universiteit Leuven, Leuven, Belgium.

Thierens, D., & Bosman, P. A. N. (2001). Multi-objective mixture-based iterated density estimation evolutionary algorithms. *Morgan Kaufmann*, 663–670.

Thierens, D., & Goldberg, D. (1994). Convergence models of genetic algorithm selection schemes. *Parallel Problem Solving from Nature*, 116–121.

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. *Proceedings of the International Conference on Genetic Algorithms (ICGA-93)*, 38–45.

Thierens, D., Goldberg, D. E., & Pereira, A. G. (1998). Domino convergence, drift, and the temporal-salience structure of problems. *Proceedings of the International Conference on Evolutionary Computation (ICEC-98)*, 535–540.

Tsutsui, S., Pelikan, M., & Goldberg, D. E. (2001). Evolutionary algorithm using marginal histogram models in continuous domain. *Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 230–233.

Van Hoyweghen, C. (2001). Detecting spin-flip symmetry in optimization problems. In Kallel, L., Naudts, B., & Rogers, A. (Eds.), *Theoretical Aspects of Evolutionary Computing* (pp. 423–437). Berlin: Springer.

Van Hoyweghen, C., Goldberg, D. E., & Naudts, B. (2001). *From TwoMax to the Ising model: Easy and hard symmetrical problems* (IlliGAL Report No. 2001030). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

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

Wright, S. (1968). *Evolution and the genetics of populations: A treatise*. University of Chicago Press.

Index

Vita

Martin Pelikan was born in Martin, Slovakia on December 8, 1974. He graduated from the Comenius University in Bratislava, Slovakia in 1998 with a degree in Computer Science. Prior to arriving in Illinois to pursue graduate study in Computer Science, he visited several laboratories as a visiting researcher, including the Department of Mathematics of the Slovak Technical University in Bratislava (Slovakia), the Adaptive Systems Group at the German National Center for Information Technology in Sankt Augustin (Germany), and the Illinois Genetic Algorithms Laboratory at the University of Illinois at Urbana-Champaign. Following the completion of his Ph.D., Pelikan will engage in postdoctoral research.