Extending the Scalability of Linkage Learning Genetic Algorithms: Theory and Practice

Ying-ping Chen

IlliGAL Report No. 2004018
April 2004

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

EXTENDING THE SCALABILITY OF LINKAGE LEARNING GENETIC ALGORITHMS: THEORY AND PRACTICE

BY

YING-PING CHEN

B.S., National Taiwan University, 1995
M.S., National Taiwan University, 1997

DISSERTATION

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, 2004

Urbana, Illinois

Abstract

There are two primary objectives of this dissertation. The first goal is to identify certain limits of genetic algorithms that use only fitness for learning genetic linkage. Both an explanatory theory and experimental results to support the theory are provided. The other goal is to propose a better design of the linkage learning genetic algorithm. After understanding the cause of the performance barrier, the design of the linkage learning genetic algorithm is modified accordingly to improve its performance on uniformly scaled problems.

This dissertation starts with presenting the background of the linkage learning genetic algorithm. Then, it introduces the use of promoters on the chromosome to improve the performance of the linkage learning genetic algorithm on uniformly scaled problems. The convergence time model is constructed by identifying the sequential behavior, developing the tightness time model, and establishing the connection in between. The use of subchromosome representations is to avoid the limit implied by the convergence time model. The experimental results demonstrate that the use of subchromosome representations may be a promising way to design a better linkage learning genetic algorithm.

The study finds that using promoters on the chromosome can improve nucleation potential and promote correct building-block formation. It also observes that the linkage learning genetic algorithm has a consistent, sequential behavior instead of different behaviors on different problems as was previously believed. Moreover, the competition among building blocks of equal salience is the main cause of the exponential growth of convergence time. Finally, adopting subchromosome representations can reduce the competition among building blocks, and therefore, scalable genetic linkage learning for a unimetric approach is possible.

To my family

Acknowledgments

First of all, I would like to thank my parents and sister for their love which supports me all the way finishing up my dissertation. Without them and their love, what has happened, what is happening, and what is going to happen would all be impossible. For this, I am forever in their debt.

If it was not for my thesis advisor, David E. Goldberg, I would never have a chance to start my research and finish my dissertation. Dave gave me the great opportunity to work so close with him and other members of the Illinois Genetic Algorithms Laboratory. In my four and a half years in the laboratory, I learned so much from him in every aspect that I do not even know where to start. Conducting research, writing papers, giving lectures, and resolving business situations are only a small fraction of it. It is not only a pleasure but also definitely an honor and privilege to acknowledge the profound influence that Dave has had on my research, my dissertation, and myself.

Furthermore, as Martin Pelikan mentioned, because of Dave, our laboratory is always full of great, brilliant people. I would like to take this opportunity to thank every one of the lab members and visitors that I worked with: Hussein Aly Abbass, Chang-Wook Ahn, Laura A. Albert, Jacob Borgerson, Martin Butz, Alessio Ceroni, Chen-Ju Chao, Jian-Hung Chen, Karen Czarnecki, Nazan Khan, Dimitri Knjazew, Alex Kosorukoff, Pier Luca Lanzi, Jeffrey Leesman, Fiepeng Li, Xavier Llorà, Jonathan Loveall, Naohiro Matsumura, Kei Onishi, Prasanna V. Parthasarathy, Gerulf Pedersen, Martin Pelikan, Jaume Bacardit, Franz Rothlauf, Kumara Sastry, Abhishek Shinha, Ryan Spraetz, Ravi Srivastava, Shigeyoshi Tsutsui, Clarissa Van Hoyweghen, Andy Vaugn, and Tian-Li Yu.

In particular, I would specially like to thank Kumara Sastry, Tian-Li Yu, Martin Pelikan, and Martin Butz for many useful, inspiring discussions. Chen-Ju Chao also deserves my special thank for lending me her notebook during the terrible sixty-hour power loss in my apartment on March 23, 24, and 25, 2004, due to the replacement of the meter banks. It was indeed the most critical time revising and rewriting my dissertation. In addition to the friends from the lab, I would like to thank Po-Hao Chang, Ruei-Sung Lin, Che-Bin Liu, Yu-Ping Tseng, Ting-Hao Yang, Wen-Tau Yih, and Dav Zimak for helping me get a lot of things done and for having fun together.

Moreover, I would like to thank my dissertation committee, David E. Goldberg, Sylvian Ray, Dan Roth, and Jay E. Mittenthal for being willing to serve on my committee. I am very grateful for their intriguing comments, questions, and suggestions.

I would also like to thank my remote friends for their support and encouragement, including Ying-Chieh Chen, Chih-Hsien Chien, Jorng-Tzong Horng, Jane Yung-Jen Hsu, Chang-Ya Hu, Colleen Chia-Li Huang, Wei-Chih Kan, Cheng-Yan Kao, and Yu-Yin Tsai.

Finally, for my lovely girlfriend, Chia-Chi Lin, I really cannot find appropriate words to fully express my gratitude. Even I myself know that just being together with me is not an easy, trivial task, let alone enduring my bad temper and irrationality for ten years. I thank Chia-Chi with all my heart for everything she did for me.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants F49620-00-1-0163 and F49620-03-1-0129. Research funding for this work was also provided by a grant from the National Science Foundation under grant DMI-9908252. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

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

Table of Contents

List of Tables

List of Figures

List of Abbreviations

BB	Building Block.
EDA	Estimation of Distribution Algorithm.
EPE	Extended Probabilistic Expression.
FBB	First-Building-Block.
fmGA	Fast Messy Genetic Algorithm.
GA	Genetic Algorithm.
LL	Linkage Learning.
LLGA	Linkage Learning Genetic Algorithm.
mGA	Messy Genetic Algorithm.
mgf	moment generating function.
PE	Probabilistic Expression.
PMBGA	Probabilistic Model-Building Genetic Algorithm.
PMX	Partially Mapped Crossover.
POI	Point of Interpretation.

Introduction

Genetic algorithms (GAs) are powerful search techniques based on principles of evolution. They are now widely applied to solve problems in many different fields. However, most genetic algorithms employed in practice nowadays are simple genetic algorithms with fixed genetic operators and chromosome representations. Unable to learn linkage among genes, these traditional genetic algorithms suffer from the linkage problem, which refers to the need of appropriately arranging or adaptively ordering the genes on chromosomes during the evolutionary process. They require their users to possess prior domain knowledge of the problem such that the genes on chromosomes can be correctly arranged in advance. One way to alleviate this burden of genetic algorithm users is to make the algorithm capable of adapting and learning genetic linkage by itself.

Harik (1997) took Holland (1975)'s call for the evolution of tight linkage quite literally and proposed the linkage learning genetic algorithm (LLGA). The linkage learning genetic algorithm uses a unique combination of the (*gene number*, *allele*) coding scheme and an exchange crossover operator to permit genetic algorithms to learn tight linkage of building blocks through a special probabilistic expression. While the linkage learning genetic algorithm performs much better on badly scaled problems than simple genetic algorithms, it does not work well on uniformly scaled problems as other competent genetic algorithms, which are a class of genetic algorithms that can solve problems quickly, accurately, and reliably (Goldberg, 2002). Therefore, we need to understand why it is so and need to know how to design a better linkage learning genetic algorithm or whether there are certain limits of such a genetic linkage learning process.

As suggested by Goldberg and Bridges (1990), there is a *race* or *time-scale comparison* in the genetic linkage learning process. If we call the characteristic time of *allelic convergence* t_α and the characteristic time of *linkage convergence* t_λ, it is easy to see that sets of alleles converge more quickly than linkage does, i.e. $t_\alpha < t_\lambda$. Because selection works on the fitness to promote good alleles and demote bad ones, allele convergence receives a stronger and more direct signal from the selection force than linkage convergence does. The force for linkage convergence only comes from the *differential selection of linkage* (Goldberg, 2002), which is generated indirectly from the schema theorem (Holland, 1975; Goldberg, 1989c; Goldberg & Sastry, 2001). Such a condition leads to the failure of genetic algorithms because loose linkage prevents genetic algorithms from getting correct alleles, and once the alleles converge to wrong combinations, the result cannot be reversed or rectified. In short, to have a working algorithm capable of learning genetic linkage, we have to make linkage convergence not slower than allele convergence, i.e. $t_\lambda \leq t_\alpha$, to ensure the success of genetic algorithms.

In order to tackle the linkage problem and handle the time-scale comparison, a variety of genetic linkage learning techniques are employed in existing competent genetic algorithms. Most of the current, successful genetic algorithms that are capable of learning genetic linkage separate the linkage learning process from the evolutionary process to avoid the time-scale comparison and utilize certain add-on criteria to guide linkage learning instead of using only the fitness given by the problem. Genetic algorithms that incorporate such add-on criteria which are not directly related to the problem at hand for learning linkage are called *multimetric* approaches. On the other hand, the algorithms that use only fitness to guide the search in both linkage learning and the evolutionary process are called *unimetric* approaches.

While multimetric approaches oftentimes yield better performance, we are particularly interested in the unimetric approach not only because it is usually easier to parallelize a unimetric approach to speed up the evolutionary process but also because the unimetric approach is more biologically plausible and closer to the observation that we can make in nature. Empirically, multimetric approaches usually perform better than unimetric approaches, and a question to ask is whether or not the unimetric approach has some upper limit on the number of building blocks up to which it can handle and process. Here, using the linkage learning genetic algorithm as the study subject, we try to understand the genetic linkage learning process and to improve the linkage learning genetic algorithm such that the insights and ramifications from this research project might be useful in the design genetic algorithms as well as in related fields of biology.

Thesis Objectives

This dissertation presents a research project that aims to gain better understanding of the linkage learning genetic algorithm in theory and to improve its performance on uniformly scaled problems in practice. It describes the steps and approaches taken to tackle the research topics, including using promoters on the chromosome, developing the convergence time model, and adopting the subchromosome representation. It also provides the experimental results for observation of the genetic linkage learning process and for verification of the theoretical models as well as the proposed new designs. Given the nature and development of this research project, there are two primary objectives of the dissertation:

1. Identify certain limits of genetic algorithms that use fitness alone, so-called *unimetric* approaches, for learning genetic linkage. The dissertation provides both an explanatory theory and experimental results to support the theory.

2. Propose a better design of the linkage learning genetic algorithm. After understanding the cause of the difficulty and performance barrier, the design of the linkage learning genetic algorithm is modified to improve its performance.

These two objectives may advance our understanding of the linkage learning genetic algorithm as well as demonstrate potential research directions.

Road Map

This dissertation is divided into eight chapters. It starts with an introduction of genetic algorithms, genetic linkage, and the linkage learning genetic algorithm. Chapter 1 presents the terminology of genetic algorithms, the pseudo-code of a simple genetic algorithm, the design-decomposition theory, and the gambler's ruin model for population sizing, followed by a discussion of genetic linkage as well as the linkage problem. The importance of learning genetic linkage in genetic algorithms is also discussed. Chapter 2 provides a set of classifications of the existing genetic linkage learning techniques such that different views from several facets of these techniques are revealed and depicted. The chapter also presents the lineage of the linkage learning genetic algorithm to demonstrate how it was developed and constructed from its precursors and ancestors. Moreover, the position of the linkage learning genetic algorithm among the existing genetic linkage learning techniques is identified. Chapter 3 describes in detail the linkage learning genetic algorithm, including (1) the chromosome representation, (2) the exchange crossover operator, (3) two mechanisms that enable the linkage learning genetic algorithm, (4) accomplishments of the linkage learning genetic algorithm, and (5) difficulties encountered by the linkage learning genetic algorithm.

After introducing the background, importance, and motivations, the approaches, results, and conclusions of this research project are presented in the remainder of this dissertation. Chapter 4 presents the assumptions regarding the framework based on which we develop the theoretical models as well as those regarding the genetic algorithm structure we adopt in this work. Then, it describes in detail the definition of the elementary test problem and the construction of the larger test problems. Chapter 4 provides a background establishment for the following chapters. Chapter 5 introduces the use of promoters and a modified exchange crossover operator to improve the performance of the linkage learning genetic algorithm.

Chapter 6 develops the convergence time model for the linkage learning genetic algorithm. It identifies the sequential behavior of the linkage learning genetic algorithm, extends the linkage-skew and linkage-shift models to develop the tightness time model, and establishes the connection between the sequential behavior and the tightness time model to construct a convergence time model for the linkage learning genetic algorithm. According to this convergence time model, chapter 7 proposes the use of subchromosome representations to avoid the limit implied by the convergence time model. The experimental results demonstrating that the use of subchromosome representations may be a promising way to design a better linkage learning genetic algorithm are also presented. Finally, chapter 8 concludes this dissertation by summarizing its contents, discussing important directions for its extension, drawing significant conclusions, and offering a number of recommendations.

Chapter 1

Genetic Algorithms and Genetic Linkage

This chapter provides a summary of fundamental material on genetic algorithms. It presents definitions of genetic algorithm terms and briefly describes how a simple genetic algorithm works. Then, it introduces the term *genetic linkage* and the so-called *linkage problem* that exists in common genetic algorithm practice. The importance of genetic linkage is often overlooked, and this chapter helps explain why linkage learning is an essential topic in the field of genetic and evolutionary algorithms. More detailed information and comprehensive background can be found elsewhere (Goldberg, 1989c; Goldberg, 2002; Holland, 1975).

Specifically, this chapter introduces the following topics:

- An overview of genetic algorithms: Gives a skeleton of genetic algorithms and briefly describes the roles of the key components.

- Goldberg's design-decomposition theory: Lays down the framework for developing facetwise models of genetic algorithms and for designing component genetic algorithms.

- The gambler's ruin model for population sizing: Governs the requirement of the population size of the genetic algorithm based on both the building-block supply and decision making. This model is employed throughout the study.

- The definition of genetic linkage and importance of linkage learning: Explains what

```
set generation $t \leftarrow 0$
randomly generate the initial population $P(0)$
evaluate all individuals in $P(0)$
repeat
  select a set of promising individuals from $P(t)$ for mating
  apply crossover to generate offspring individuals
  apply mutation to perturb offspring individuals
  replace $P(t)$ with the new population
  set generation $t \leftarrow t + 1$
  evaluate all individuals in $P(t)$
until certain termination criteria are met
```

Figure 1.1: Pseudo-code of a simple genetic algorithm.

genetic linkage is in both biological systems and genetic algorithms as well as gives the reason why genetic linkage learning is an essential topic in the field of genetic and evolutionary algorithms.

In the following sections, we will start with the overview of genetic algorithms, followed by the design-decomposition theory for genetic algorithms, the gambler's ruin model for population sizing, and the introduction of genetic linkage learning.

1.1 Overview of Genetic Algorithms

Genetic algorithms are stochastic, population-based search and optimization algorithms loosely modeled after the paradigms of evolution. Genetic algorithms guide the search through the solution space by using natural selection and genetic operators, such as crossover, mutation, and the like. In this section, the mechanisms of a genetic algorithm are briefly introduced as a background of this research project. The pseudo-code of a simple genetic algorithm is shown in Figure 1.1.

1.1.1 Representation, Fitness, and Population

Based on the principles of natural selection and genetics, genetic algorithms encode the decision variables or input parameters of the underlying problem into solution strings of a finite length over an alphabet of certain cardinality. Characters in the solution string are called *genes*. The value and the position in the string of a gene are called *locus* and *allele*, respectively. Each solution string is called an *individual* or a *chromosome*. While traditional optimization techniques work directly with the decision variables or input parameters, genetic algorithms usually work with the codings of them. The codings of the variables are called *genotypes*, and the variables themselves are called *phenotypes*.

For example, if the decision variable to a problem at hand is an integer $x \in [0, 63]$, we can encode x in the specified range as a 6-bit string over a binary alphabet $\{0,\ 1\}$ and define the mapping between an individual $A \in \{0, 1\}^6$ and the value $x(A)$ represented by A as

$$x(A) = \sum_{i=0}^{5} 2^i A(i) \ ,$$

where i is the position of the character in the string, and $A(i)$ is the ith character (either 0 or 1). A, the binary string, is the genotype of x, and $x(A)$, the integer value, is the phenotype of x. For instance, $x(A_1 = 010001) = 2^0 + 2^4 = 17$, and $x(A_2 = 100110) = 2^1 + 2^2 + 2^5 = 38$, where the positions are read from right to left.

After having the solutions to the problem encoded, a method or procedure for distinguishing good solutions from bad solutions is in order. The procedure can be a laboratory experiment, a field test, a complex computer simulation, or a human who decides the solution quality. Genetic algorithms work with these different modes or measurements as long as better solutions are assigned higher *fitness* when compared to worse solutions. Fitness in nature is to provide a differential signal according to which genetic algorithms guide the evolution of solutions to the problem.

Equipped with the coding scheme to represent the solutions to the problem and the fitness function to distinguish good solutions from bad solutions, genetic algorithms start to search from a *population* of encoded solutions instead of from a single point in the solution space. The initial population of individuals can be created at random or with some problem-specific knowledge. In order to evolve new solutions, genetic algorithms use genetic operators to create promising solutions based on the solutions in the current population. The most popular genetic operators are (1) *selection*, (2) *crossover*, and (3) *mutation*, which will be described in what follows. The newly generated individuals replace the old population, and the evolution process proceeds until certain termination criteria are satisfied.

1.1.2 Selection, Crossover, and Mutation

As previously mentioned, genetic algorithms evolve the population of solutions with genetic operators, including selection, crossover, and mutation. The selection procedure implements the natural selection force or the survival-of-the-fittest principle and selects good individuals out of the current population for generating the next population according to the assigned fitness. The existing selection operators can be broadly classified into two classes: (1) *proportionate* schemes, such as roulette-wheel selection (Goldberg, 1989c) and stochastic universal selection (Baker, 1985; Grefenstette & Baker, 1989), and (2) *ordinal* schemes, such as tournament selection (Goldberg, Korb, & Deb, 1989) and truncation selection (Mühlenbein & Schlierkamp-Voosen, 1993).

Ordinal schemes have grown more and more popular over the recent years, and one of the most popular ordinal selection operators is tournament selection. Tournament selection selects s, called the *tournament size*, individuals from the current population of size n. The best individual among the s individuals gets one copy in the mating pool. The selection of s individuals can be done with or without replacement. The procedure is repeated until the mating pool is filled with n individuals. Hence, in tournament selection, the best individual on average gets s copies in the mating pool, and the worst one gets none.

After selection, *crossover* and *mutation* recombine and alter parts of the individuals to generate new solutions. Crossover, also called the *recombination* operator, exchanges parts of solutions from two or more individuals, called *parents*, and combines these parts to generate new individuals, called *children*, with a *crossover probability*, p_c. There are a lot of ways to implement a recombination operator. As an example, consider two individuals A_1 and A_2:

$$A_1 = \texttt{1 1 | 1 1 1 | 1}$$

$$A_2 = \texttt{0 0 | 0 0 0 | 0}$$

The symbol | indicates the positions of the crossover points which are chosen at random. The widely used two-point crossover generates the following two children A'_1 and A'_2:

$$A'_1 = \texttt{1 1 0 0 0 1}$$

$$A'_2 = \texttt{0 0 1 1 1 0}$$

Other well known crossover operators include one-point crossover and uniform crossover. When using one-point crossover, only one crossover point is chosen at random, such as

$$A_1 = \texttt{1 1 1 1 | 1 1}$$

$$A_2 = \texttt{0 0 0 0 | 0 0}$$

Then, one-point crossover recombines A_1 and A_2 and yields:

$$A'_1 = \texttt{1 1 1 1 0 0}$$

$$A'_2 = \texttt{0 0 0 0 1 1}$$

While one-point crossover and two-point crossover choose some crossover points at random for cutting the individual into pieces, uniform crossover exchanges each gene with a probability 0.5. Uniform crossover, on average, swaps half of genes of the individuals and therefore achieves the maximum allele-wise mixing rate.

Mutation usually alters some pieces of individuals to form perturbed solutions. In contrast to crossover, which operates on two or more individuals, mutation operates on a single individual. One of the most popular mutation operators is the bitwise mutation, in which each bit in a binary string is complemented with a *mutation probability*, p_m. For example,

$$A = \texttt{1 1 1 1 1 1}$$

might become

$$A' = \texttt{1 1 1 1 0 1}$$

after mutation. Mutation performs a random-walk in the vicinity of the individual in the solution space, which may keep the solution diversity of the current population.

It has been shown that although these genetic operators are ineffectual when analyzed individually, they can work well when cooperating with one another (Goldberg, 1999; Goldberg, 2002). This aspect has been explained with the concepts of the *fundamental intuition* and *innovation intuition* by Goldberg (2002). In that study, the combination of selection and mutation, called *selectomutative* genetic algorithms, is compared to *continual improvement* (a form of hillclimbing), and the combination of selection and crossover, called *selectorecombinative* genetic algorithms, is compared to the *cross-fertilizing* type of *innovation*. These analogies have been used to develop *competent* genetic algorithms, which are a class of genetic algorithms that solve hard problems quickly, accurately, and reliably. Competent genetic algorithms successfully solve problems of *bounded difficulty* and *scale* polynomially (oftentimes subquadratically) with the problem size (Goldberg, 2002). In order to design competent genetic algorithms, a design decomposition has been proposed (Goldberg, 1991; Goldberg & Liepens, 1991; Goldberg, Deb, & Clark, 1992; Goldberg, 1993; Goldberg, 2002), and a note on the design-decomposition theory is presented in the following section.

1.2 Goldberg's Design Decomposition

As discussed in the previous section, we are interested in developing competent genetic algorithms that can solve problems of *bounded difficulty* and *scale* polynomially (oftentimes subquadratically) with the problem size. Based on Holland (1975)'s notion of a building block (BB), Goldberg proposed a design-decomposition theory for designing a selectorecombinative genetic algorithm that can solve hard problems quickly, accurately, and reliably (Goldberg, 1991; Goldberg & Liepens, 1991; Goldberg, Deb, & Clark, 1992; Goldberg, 1993; Goldberg, 2002). As of its current development, the design decomposition consists of seven steps (Goldberg, 2002), which lead us step by step toward designing competent genetic algorithms. These steps are reproduced and briefly described in the following paragraphs:

1. **Know what GAs process—building blocks (BBs).** This step emphasizes that genetic algorithms work through quasi-invariant, highly fit structures—components of good solutions—identified as building blocks by Holland (1975). The key idea is that genetic algorithms implicitly, virtually decompose the problem into subproblems, find subsolutions, and combine different subsolutions to form solutions of high quality.

2. **Solve problems of bounded BB difficulty.** Competent genetic algorithms are capable of solving problems reliably if the building blocks involved are of low order. According to Goldberg's cross-fertilizing innovation, problems are hard because they have deep or complex building blocks, or because they have building blocks that are hard to separate, or because they have low-order building blocks that are *misleading* or *deceptive* (Goldberg, 1987; Goldberg, 1989b; Goldberg, Deb, & Horn, 1992; Deb & Goldberg, 1993; Deb & Goldberg, 1994). Hence, we know that there exist very hard problems that are too difficult to solve with reasonable computational resources. However, here we are interested in solving nearly decomposable problems of bounded difficulty, which include a wide range of practical problems with a low difficulty level. The primary interest is to develop competent genetic algorithms to efficiently solve problems of certain level of difficulty through building-block processing.

3. **Ensure an adequate supply of raw BBs.** This step states that for a genetic algorithm to successfully solve a problem, all the necessary raw building blocks should be available in a sufficient manner. In a selectorecombinative genetic algorithm, building blocks are mixed and recombined to create complete solutions. Therefore, this con-

dition suggests that the initial population should be large enough to ensure that the supply of raw building blocks is adequate from the beginning.

4. **Ensure increased market share for superior BBs.** The proportion of good building blocks in the population should continue to grow over the generations to ensure the continuous supply of good building blocks. Cooperating with the previous step, which ensures the supply of raw building blocks from the beginning, good building blocks can have more and more copies in the population such that a genetic algorithm can combine them to create promising solutions. Otherwise, because of the lack of good building blocks, there is little chance to mix and combine appropriate building blocks to yield the global optimum.

5. **Know BB takeover and convergence times.** Although it is stated in the previous step that the market share for superior building blocks has to grow and cannot grow too slowly over the generations, the growth rate of the portion of superior building blocks in the population should not be too fast, either. If the proportion of certain building blocks grows too fast, a genetic algorithm will not have enough time to work on the exchange and recombination of raw building blocks before there is no alternative building blocks left in the population for conducting effective recombination. Such condition may result in premature convergence of genetic algorithms or lead to the domination of deceptive building blocks if they exist.

6. **Ensure the BB decisions are well made.** Because genetic algorithms work on the fitness of an individual containing multiple building blocks, they face a statistical decision problem when deciding among competing building blocks. For example, a bad building block might be selected over its competitors because it comes with other good building blocks in the same individual. Given a particular building block, fitness contribution of the building blocks from other partitions in the individual can be considered as noise to the fitness. Therefore, this condition also suggests that the

population size should be sufficiently large for a genetic algorithm to make statistically correct decisions among competing building blocks.

7. **Ensure a good mixing of BBs.** After having the steps that can ensure the supply of raw building blocks, the appropriate growth rate of market share for good building blocks, and the correct decisions among competing building blocks, we need to efficiently and effectively mix and reassemble these building blocks in order to create high quality solutions. In other words, the supply and growth of good building blocks continuously maintains the presence of individual building blocks. A good mixing of building blocks is the other key component to ensure a GA success by combining and assembling these building blocks.

This design-decomposition theory explains the way genetic algorithms work, indicates the direction to design competent genetic algorithms, and helps the development of theoretical models for predicting the scalability of genetic algorithms. To ensure the supply of raw building blocks, several population-sizing models were proposed for estimating reasonable population sizes (Holland, 1973; Holland, 1975; Goldberg, 1989c; Reeves, 1993; Goldberg, Sastry, & Latoza, 2001) based on the building-block supply. To ensure the building-block growth, we can set the selection pressure and the crossover probability appropriately (Goldberg & Sastry, 2001; Goldberg, 2002) according to the schema theorem (De Jong, 1975; Holland, 1975; Goldberg, 1989c), which describes the market share growth of building blocks in terms of selection pressure and crossover probability.

The convergence time discussed in the design-decomposition theory dictates the time requirement of genetic algorithms. Studies were conducted through facetwise models (Bäck, 1995; Blickle & Thiele, 1995; Blickle & Thiele, 1996; Miller & Goldberg, 1995; Miller & Goldberg, 1996; Thierens & Goldberg, 1994a; Thierens & Goldberg, 1994b; Thierens & Goldberg, 1998) and found that the convergence time for genetic algorithms scales up with the order of the square root of the problem size or scales up linearly to the problem size on different types of problems examined in the those studies when the ordinal selection schemes are employed. To ensure good building-block decision making, a population-sizing model considering the fitness contribution from other building blocks as the noise of fitness of the building block in question were also developed to achieve statistically correct decisions among competing building blocks (Goldberg, Deb, & Clark, 1992). Finally, by integrating the requirements of the building-block supply and decision making, a refined population-sizing model considering the evolutionary process as the gambler's ruin problem (Feller, 1970) was proposed (Harik, Cantú-Paz, Goldberg, & Miller, 1997; Harik, Cantuú-Paz, Goldberg, & Miller, 1999) to provide a tighter bound on the population size required by selectorecombinative genetic algorithms. Because the gambler's ruin model for population sizing is used to determine the population sizes in the experiments throughout this study, we will describe it in detail next.

1.3 Population-Sizing Model

As discussed in the previous section, the gambler's ruin model for population sizing provides a tight bound on the population size required by selectorecombinative genetic algorithms and is used to determine the population size throughout this study. Therefore, in this section, we reproduce the gambler's ruin model proposed by Harik, Cantú-Paz, Goldberg, and Miller (1997) and summarized by Sastry (2002) to introduce its idea, derivation, and implication.

Harik, Cantú-Paz, Goldberg, and Miller (1997) incorporated both the initial building-block supply model and the decision-making model in the population-sizing relation to construct the model which can provide a tight bound of the population size. They removed the requirement used by Goldberg, Deb, and Clark (1992) that only a successful decision making in the first generation results in the convergence to the optimum. In order to remove this criterion, they modeled the decision making in subsequent generations with the well known gambler's ruin model (Feller, 1970). The proposed population-sizing equation is directly proportional to the square root of the problem size, inversely proportional to the square root of the signal to the noise ratio, and is proportional to 2^k, where k is the building-block size. Furthermore, Miller (1997) extended this population-sizing model for noisy environments, and Cantú-Paz (1999) applied it for parallel genetic algorithms.

For deriving the gambler's ruin model, we need to make the following assumptions: (1) the building blocks are of same salience, (2) the building blocks are of the same size, k, (3) the total fitness of an individual is an additive function of building blocks in all schema partitions, (4) the fitness function is stationary, time independent, but can contain external noise, and (5) the initial population has equal proportion of building blocks in all schema partitions. Now, we can start to construct the model with calculating the initial building-block supply. Based on these assumptions, the number of copies which a building block receives in an initial population of size n is

$$x_0 = \frac{n}{\chi^k} \, , \tag{1.1}$$

where χ is the cardinality of the alphabet used in the chromosome-coding scheme, k is the building-block size, and χ^k are the number of possible schemata.

In addition to the building-block supply, we also need to model the decision making between building blocks in genetic algorithms. The selection procedure is the principal operator that performs decision making in the evolutionary process. However, it discriminates the individuals on a chromosomal level instead of a building-block level. Therefore, a bad building block might be selected over its competitors because it comes with good building blocks from other schema partitions in the same individual. Given a particular building block, fitness contribution of the building blocks from other schema partitions can be considered as noise to the fitness. In order to derive a relation for the probability of deciding between building blocks correctly, we consider two individuals, one with the best building block in a partition, and the other with the second best building block in the same partition (Goldberg, Deb, & Clark, 1992).

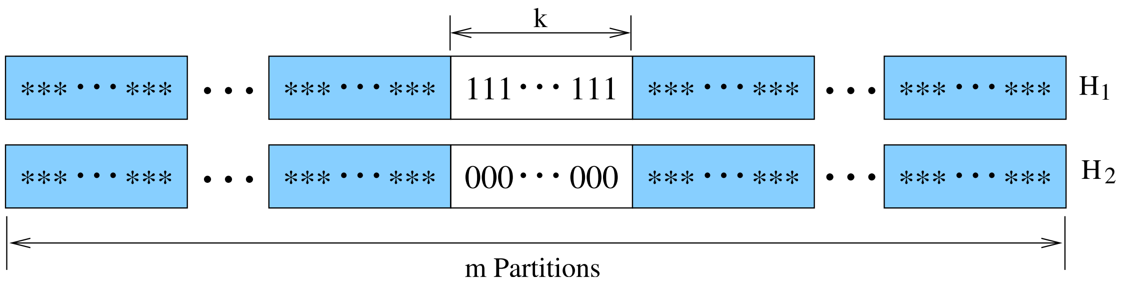


Figure 1.2: Two competing building blocks of size k, one is the best building block, H_1, and the other is the second best building block, H_2 (Sastry, 2002). Reprinted by permission.

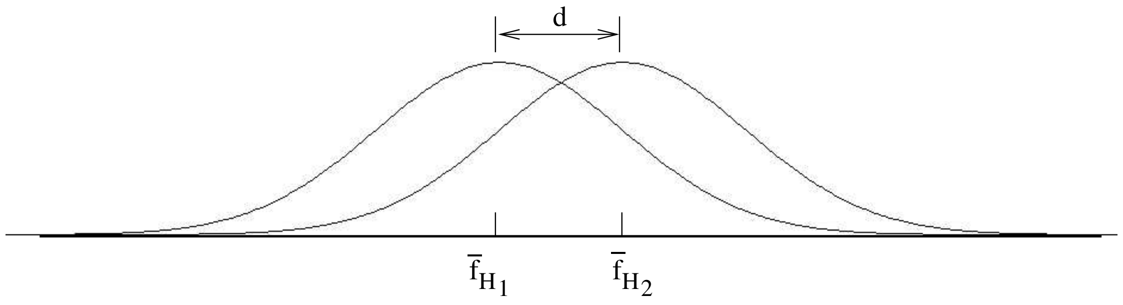


Figure 1.3: Fitness distribution of individuals in the population containing the two competing building blocks, the best building block H_1, and the second best building block H_2 (Sastry, 2002). Reprinted by permission.

Let A_1 and A_2 be two individuals with m non-overlapping building blocks of size k as shown in Figure 1.2. Individual A_1 has the best building block, H_1 ($111\cdots111$ in Figure 1.2) and individual A_2 has the second best building block, H_2 ($000\cdots000$ in Figure 1.2). The fitness values of A_1 and A_2 are f_{H_1} and f_{H_2}, respectively. To derive the probability of correct decision making, the fitness distribution of the chromosomes containing H_1 and H_2 can be modeled with a Gaussian distribution, which following from the third assumption, the additive fitness function assumption, and from the central limit theorem. The fitness distributions of individuals containing building blocks H_1 and H_2 are illustrated in Figure 1.3. The distance between the mean fitness of individuals containing H_1, $\overline{f}_{H_1}$, and the mean fitness of individuals containing H_2, $\overline{f}_{H_2}$, is the *signal*, d, where

$$d = \overline{f}_{H_1} - \overline{f}_{H_2}\ . \tag{1.2}$$

Since f_{H_1} and f_{H_2} are normally distributed, $f_{H_1} - f_{H_2}$ is also normally distributed with mean d and variance $\sigma^2_{H_1} + \sigma^2_{H_2}$, where $\sigma^2_{H_1}$ and $\sigma^2_{H_2}$ are the fitness variances of individuals containing H_1 and H_2, respectively. Because the probability to correctly decide between H_1 and H_2 is equivalent to that of $f_{H_1} - f_{H_2} > 0$, we obtain

$$f_{H_1} - f_{H_2} \sim \mathcal{N}(d, \sigma^2_{H_1} + \sigma^2_{H_2}) \ , \tag{1.3}$$

and the probability to make statistically correct decisions, p_{dm}, is given by

$$p_{dm} = P(f_{H_1} - f_{H_2} > 0) = P\left(\frac{f_{H_1} - f_{H_2} - d}{\sqrt{\sigma^2_{H_1} + \sigma^2_{H_2}}} > -\frac{d}{\sqrt{\sigma^2_{H_1} + \sigma^2_{H_2}}}\right) \ .$$

Because $(f_{H_1} - f_{H_2} - d)/\sqrt{\sigma^2_{H_1} + \sigma^2_{H_2}}$ follows a unit Gaussian distribution, by using the symmetry of the Gaussian distribution, we can obtain p_{dm} as

$$p_{dm} = \Phi\left(\frac{d}{\sqrt{\sigma^2_{H_1} + \sigma^2_{H_2}}}\right) \ . \tag{1.4}$$

Based on the assumptions that the fitness function is the sum of m independent subfunctions of size k, $\sigma^2_{H_1}$ is the sum of the variances of the $m-1$ subfunctions (Goldberg, Deb, & Clark, 1992). This calculation is also valid for $\sigma^2_{H_2}$. Moreover, since the m schema partitions are uniformly scaled (or of the same salience), the variance of each subfunction is equal to the average building-block variance, σ^2_{bb}. Hence,

$$\sigma^2_{H_1} = \sigma^2_{H_2} = (m-1)\sigma^2_{bb} \ . \tag{1.5}$$

Substituting Equation (1.5) in Equation (1.4), we obtain

$$p_{dm} = \Phi\left(\frac{d}{\sqrt{2(m-1)}\sigma_{bb}}\right) \ , \tag{1.6}$$

where $\Phi(z)$ is the cumulative density function of a unit normal variate. Now we have the initial building-block supply, x_0, from Equation (1.1) and the probability to make correct decisions between building blocks, p_{dm}, in Equation (1.6). We are ready to integrate these results with the gambler's ruin model to derive the population-sizing equation.

In the gambler's ruin problem, as shown in Figure 1.4, the gambler starts with some initial capital, c_0, and competes with an opponent with initial capital, $c_t - c_0$. The gambler can reach one of the two absorbing states, one in which he loses all the money and the other in which he wins all the money from his opponent. For each run of the game, the gambler may win one unit of money from his opponent with a probability p, or lose one unit to his opponent with a probability $1 - p$. The objective in this game is to win all the money from his opponent. Harik, Cantú-Paz, Goldberg, and Miller (1997) drew an analogy between the gambler's ruin problem and selection of building blocks in a single partition by recognizing that the capital of the gambler is the number of individuals containing the best building block, H_1. That is, the gambler starts with an initial capital of x_0 (Equation (1.1)) and competes with an opponent with an initial capital of $n - x_0$, where n is the population size. The probability of increasing the capital by an unit is the probability of correct decision making p_{dm} (Equation (1.6)). Genetic algorithms succeed when all the individuals in the population have the best building block. From the gambler's ruin model (Feller, 1970), the probability that the best building block takes over the whole population is given by:

$$P_{bb} = \frac{1 - \left(\frac{1-p_{dm}}{p_{dm}}\right)^{x_0}}{1 - \left(\frac{1-p_{dm}}{p_{dm}}\right)^{n}} \, . \tag{1.7}$$

Since $\overline{f}_{H_1} > \overline{f}_{H_2}$, $p_{dm} > 1 - p_{dm}$, and for moderate-to-large values of n the denominator approaches 1 and can be neglected. After neglecting the denominator and writing x_0 in terms of n, Equation (1.7) can be approximately reduced to

$$P_{bb} = 1 - \left(\frac{1 - p_{dm}}{p_{dm}}\right)^{\frac{n}{\chi^k}} \, . \tag{1.8}$$

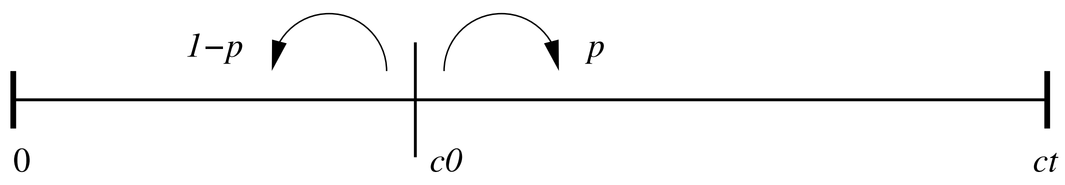


Figure 1.4: In the gambler's ruin problem, the gambler starts with some initial capital, c_0, and competes with an opponent with initial capital, $c_t - c_0$. The gambler can reach one of the two absorbing states, one in which he loses all the money and the other in which he wins all the money from his opponent. For each run of the game, the gambler may win one unit of money from his opponent with a probability p, or lose one unit to his opponent with a probability $1 - p$. The objective in this game is to win all the money from his opponent.

Because P_{bb} is the probability for the best building block to reside in every individual, $1-P_{bb}$ is therefore the probability that the best building block fails to take over the population. If we let $1 - P_{bb}$ the failure rate, α, we can derive an expression for the population size, n, to ensure a specified level of success as

$$n = \frac{\chi^k \log \alpha}{\log\left(\frac{1-p_{dm}}{p_{dm}}\right)} \,. \tag{1.9}$$

In order to simply Equation (1.9) in terms of the building-block size, k, the signal to noise ratio, d/σ_{bb}^2, and the number of building blocks, m, several approximations are applied. First, we expand p_{dm} by using the first two terms of the power series (Abramowitz & Stegun, 1972):

$$p_{dm} \approx \frac{1}{2}\left(1 + \frac{d}{\sigma_{bb}\sqrt{\pi(m-1)}}\right) \,. \tag{1.10}$$

Substituting the above approximation in Equation (1.9), we obtain

$$n = \frac{\chi^k \log \alpha}{\log\left(1 - d/(\sigma_{bb}\sqrt{\pi(m-1)})\right) - \log\left(1 + d/(\sigma_{bb}\sqrt{\pi(m-1)})\right)} \,. \tag{1.11}$$

For problems of moderate-to-large sizes or for problems with a low signal to noise ratio,

$d/(\sigma_{bb}\sqrt{\pi(m-1)})$ tends to be small, and the following approximation,

$$\log\left(1 \pm \frac{d}{\sqrt{\pi(m-1)}}\right) \approx \pm\frac{d}{\sqrt{\pi(m-1)}} \ , \tag{1.12}$$

can be applied to Equation (1.11). Thus, we get

$$n = -\frac{1}{2}\chi^k \log(\alpha)\,\frac{\sigma_{bb}\sqrt{\pi(m-1)}}{d} \ . \tag{1.13}$$

Because in this work, we use a binary alphabet containing $0, 1$ to encode the solutions, the cardinality, χ, is therefore 2. By substituting $\chi = 2$ in Equation (1.13), we obtain the approximate population-sizing model which determines the population size used in this work:

$$n = -2^{k-1} \log(\alpha)\,\frac{\sigma_{bb}\sqrt{\pi(m-1)}}{d} \ . \tag{1.14}$$

Furthermore, if the tournament size used in a genetic algorithm is not 2, which was assumed in the derivation of the gambler's ruin model, competitions among building blocks will be slightly different. For this condition, Goldberg, Deb, and Clark (1992) made the gambler's ruin model valid for genetic algorithm configurations in which a tournament size other than 2 is employed by proposing the following signal adjustment formula:

$$d' = d + \Phi^{-1}\left(\frac{1}{s}\right)\sigma_{bb} \ , \tag{1.15}$$

where s is the tournament size, and $\Phi^{-1}(1/s)$ is the ordinate of a unit normal distribution where the CDF equals $1/s$.

Followed the guidelines for designing selectorecombinative genetic algorithms expressed by Goldberg's design-decomposition theory, numerous studies and researches in the field of genetic and evolutionary computation have been conducted for tackling the issues in design decomposition piece by piece. These studies include those on the time, such as the takeover time and convergence time, as well as those on the space, such as the population-sizing models based on several different assumptions. They not only improve our understandings of the genetic algorithm operations in theory but also enhance the design of genetic algorithms in practice, which leads to the design of competent genetic algorithms. In the next section, we will briefly discuss the competent genetic algorithms and point out one of the most important mechanisms for genetic algorithms to succeed—genetic linkage learning.

1.4 Competent Genetic Algorithms

As mentioned previously, *competent* genetic algorithms are a class of genetic algorithms that solve hard problems quickly, accurately, and reliably. Hard problems are roughly defined as those problems that have large subsolutions which cannot be decomposed into simpler ones and must be discovered whole, or have badly scaled subsolutions, or have a lot of local optima, or have a very high degree of interaction among subsolutions, or are subject to a high level of external noise. While designing a competent genetic algorithms, the objective is to develop genetic algorithms that can successfully solve problems of *bounded difficulty* and *scale* polynomially (oftentimes subquadratically) with the problem size (Goldberg, 2002).

Following the steps presented in the design-decomposition theory, a number of genetic algorithm designs have been proposed to achieve the criteria of competent genetic algorithms. According to the current research on selectorecombinative genetic algorithms, it is found that effective and efficient building-block identification and exchange is critical to the success of genetic algorithms. Many of these competent genetic algorithms address the issue of building-block identification and exchange by utilizing a variety of techniques. Although both building-block identification and exchange are critical and of our interests, based on the nature of this work, we will focus on the issue of building-block identification.

In the context of this study, we consider building-block identification in the form of genetic linkage learning and concentrate on the understanding and improvement of one of the existing competent genetic algorithms, called the *linkage learning genetic algorithm* (LLGA), which will be introduced in a separate chapter. But before that, we first introduce the definition of genetic linkage and emphasize the importance of linkage learning in the following section.

1.5 Genetic Linkage and the Linkage Problem

As discussed in the previous section, we consider building-block identification as genetic linkage learning in this work. In this section, we present the definition of genetic linkage and discuss the importance of linkage learning. Particularly, the following topics are presented in this section:

- The definition of genetic linkage: Describes what genetic linkage is in biological systems and genetic algorithms.
- The ordering problem: Is a way to consider the linkage problem in terms of the chromosome representation in the literature.
- The importance of linkage learning: Explains the reason why linkage learning is an essential topic in the field of genetic and evolutionary algorithms.

We will start from introducing the definition of genetic linkage and then discuss why genetic linkage and the linkage problem are important in the context of genetic algorithms.

1.5.1 What Is Genetic Linkage?

Since genetic linkage is one of the central topics in this study, the definition of genetic linkage in both biological systems and genetic algorithms is introduced in this section. First, in biological systems, genetic linkage refers to the greater association in inheritance of two or more nonallelic genes than is to be expected from independent assortment (Hartl & Jones, 1998). When a crossover event occurs during *meiosis*, which refers to the process of nuclear division in gametogenesis or sporogenesis in which one replication of the chromosomes is followed by two successive divisions of the nucleus to produce four haploid nuclei (Hartl & Jones, 1998), the genetic material is recombined as shown in Figure 1.5. Based on this meiosis-crossover process, if two genes are closer to each other on a chromosome, there is a higher probability that they will be transferred to the offspring together. Figure 1.6 gives an illustrative example. Therefore, genes are said to be *linked* when they reside on the same chromosome in biological systems, and their distance between each other determines the level of their linkage. The closer together a set of genes is on a chromosome, the more probable it will not be split during the meiosis-crossover process.

When using genetic algorithms, we use strings of characters as chromosomes and genetic operators to manipulate these chromosomes. In Holland (1975)'s seminal publication, *Adaptation in Natural and Artificial Systems*, he indicated that crossover in genetic algorithms induces a *linkage* phenomenon. In this study, we loosely define the term *genetic linkage* for a set of genes as follows:

> If the genetic linkage between these genes is tight, the crossover operator disrupts them with a low probability and transfers them all together to the child individual with a high probability. On the other hand, if the genetic linkage between these genes is loose, the crossover operator disrupts them with a high probability and transfers them all together to the child individual with a low probability.

Note that the above definition of genetic linkage is identical to its counterpart in biological systems. Furthermore, this definition implies that the genetic linkage of a set of genes depend on the chromosome representation and the crossover operator.

Given the definition of genetic linkage, many well known and widely applied crossover operators, including one-point crossover and two-point crossover mentioned previously, induce the linkage phenomenon on a fixed chromosome representation as their counterparts do in biological systems. For example, if we have a 6-bit function consisting of two independent

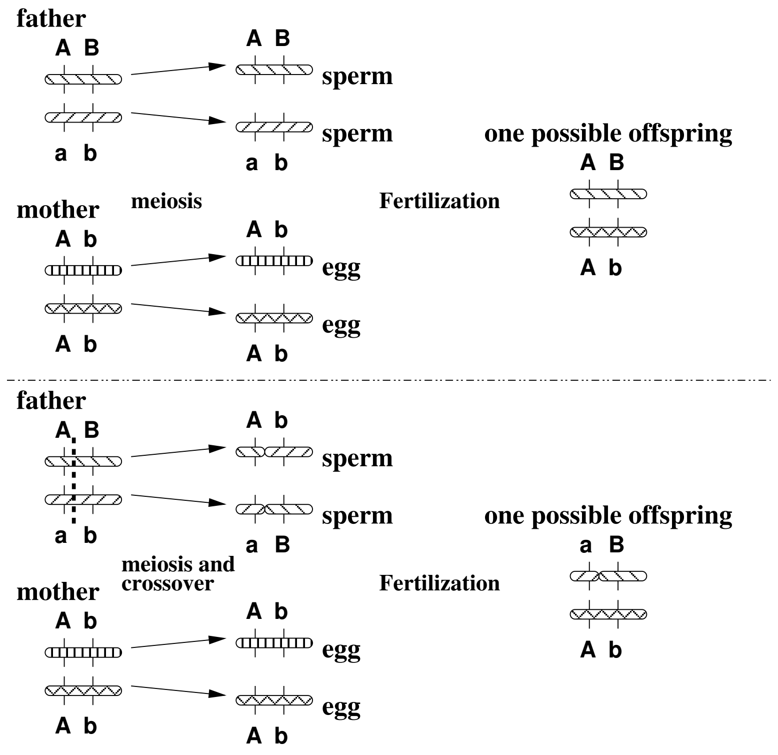


Figure 1.5: Crossover and meiosis. The upper part shows meiosis without crossover, and the lower part shows a crossover event occurs during meiosis.

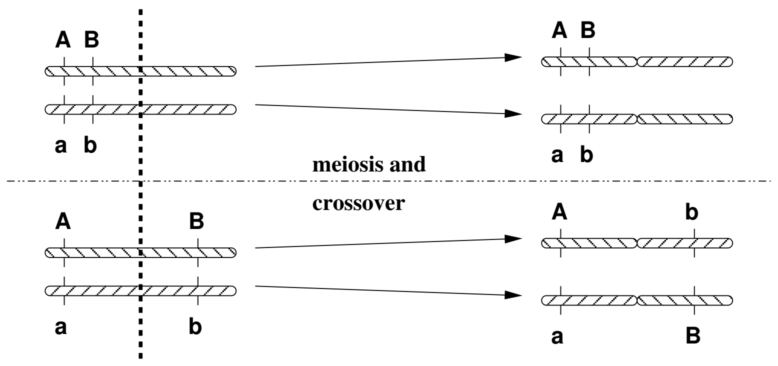


Figure 1.6: The genetic linkage between two genes. The upper part shows that if the genes are closer, they are likely to maintain the allele configuration. The lower part shows that if the genes are far away from each other, it is easier for a crossover event to separate them.

3-bit subfunctions, three possible coding schemes for the 6-bit chromosome are

$$C_1(A) = \mathtt{a}_0^0\ \mathtt{a}_1^0\ \mathtt{a}_2^0\ \mathtt{a}_3^1\ \mathtt{a}_4^1\ \mathtt{a}_5^1;$$

$$C_2(A) = \mathtt{a}_0^0\ \mathtt{a}_1^1\ \mathtt{a}_2^0\ \mathtt{a}_3^1\ \mathtt{a}_4^0\ \mathtt{a}_5^1;$$

$$C_3(A) = \mathtt{a}_0^0\ \mathtt{a}_1^0\ \mathtt{a}_2^1\ \mathtt{a}_3^1\ \mathtt{a}_4^1\ \mathtt{a}_5^0,$$

where $C_n(A)$ is the coding scheme n for an individual A, and a_i^j is the ith gene of A and belongs to the jth subfunction.

Taking one-point crossover as an example, it is easy to see that genes belonging to the same subfunction of individuals encoded with C_1 are unlikely to be separated by crossover events. However, if the individuals are encoded with C_2, genes of the same subfunction are split almost every time when a crossover event occurs. For C_3, genes of subfunction 0 are easy to be disconnected, while genes of subfunction 1 are likely to stay or to be transferred together. From the viewpoint of genetic algorithms, genetic linkage can be used to describe and measure how close those genes belonging a building block are on a chromosome. Moreover, Holland (1975) suggested that the chromosome representation should adapt during the evolutionary process to avoid the potential difficulty directly caused by coding schemes.

1.5.2 Linkage Learning as an Ordering Problem

Because encoding the solutions as fixed strings of characters is common in genetic algorithm practice, it is easy to see that genetic linkage can be identified as the ordering of the loci of genes as the examples given in the previous section. Furthermore, early genetic algorithm researchers used to consider the linkage problem as an ordering problem of the chromosome representation and addressed to the same issue of building-block identification or genetic linkage learning. That is, if a genetic algorithm is capable of rearranging the positions of genes on the fly during the evolutionary process, the responsibility of the user to choose a good coding scheme can be alleviated or even eliminated. In order to achieve this goal, Bagley (1967) used the (*gene number*, *allele*) coding scheme to study the *inversion* operator for genetic linkage learning by reversing the order of a chromosome segment but did not conclude in favor of the use of inversion. Frantz (1972) further investigated the utility of inversion and reported that inversion was too slow and not very effective.

Goldberg and Bridges (1990) analyzed the performance of a genetic algorithm with an idealized reordering operator. They showed that with an idealized reordering operator, the *coding traps*—the combination of loose linkage and deception among lower order schemata (Goldberg, 1987)—of a fixed chromosome representation can be overcome, and therefore, genetic linkage learning can be achieved by an idealized reordering operator. This analysis was extended to the tournament selection family, including pairwise tournament selection, S-ary tournament selection, and probabilistic tournament selection (Chen & Goldberg, 2003a). The upper bound of the probability to apply an idealized reordering operator found in the previous analysis on proportional selection did not exist when tournament selection was used.

1.5.3 Why Is Genetic Linkage Learning Important?

Although according to Goldberg's design-decomposition theory, building-block identification or genetic linkage learning is critical to the success of genetic algorithms, except for competent genetic algorithms, most genetic algorithms in practice today use fixed genetic operators and chromosome representations. These genetic algorithms either explicitly or implicitly act on an assumption of a good coding scheme which can provide tight linkage for genes of a building block on the chromosome. Goldberg, Korb, and Deb (1989) used an experiment to demonstrate how genetic linkage dictate the success of a simple genetic algorithm. They used an objective function composed of 10 uniformly scaled copies of an order-3 fully deceptive function (Goldberg, 1989a; Goldberg, 1989b). Three types of codings schemes were tested: tightly ordering, loosely ordering, and randomly ordering. The tightly ordering coding scheme is similar to C_1 described in section 1.5.1. Genes of the same subfunction are arranged adjacent to one another on the chromosome. The loosely ordering coding scheme is like C_2, all genes are distributed evenly so that an overall loosest linkage can be achieved. The randomly ordering coding scheme arranges the genes according to an arbitrary order.

The results obtained by Goldberg, Korb, and Deb (1989) are shown in Figures 1.7 and 1.8. We can observe in these figures that the success of a simple genetic algorithm depends very much on the degree of genetic linkage. If the chromosome representation provides tight linkage, a simple genetic algorithm can solve difficult problems. Otherwise, simple genetic algorithms can easily fail. Therefore, for simple genetic algorithms, tight genetic linkage, or a good coding scheme, is indeed far more important than it is usually considered.

In addition to the experiments done by Goldberg, Korb, and Deb (1989), some other studies (Thierens, 1995; Goldberg, Deb, & Thierens, 1993; Goldberg, 1989c) also showed that genetic algorithms work very well if the genes belonging to the same building block are tightly linked together on the chromosome. Otherwise, if these genes spread all over the chromosome, building blocks are very hard to be created and easy to be destroyed by the recombination operator. Genetic algorithms cannot perform well under such circumstances. In practice, without prior knowledge of the problem and linkage information, it is difficult to guarantee that the coding scheme defined by the user always provides tight building blocks, although it is a key to the success of genetic algorithms.

It is clear that for simple genetic algorithms with fixed genetic operators and chromosome representations, one of the essential keys to success is a good coding scheme that puts genes belonging to the same building blocks together on the chromosome to provide tight linkage of building blocks. The genetic linkage of building blocks dominates all kinds of building-block processing, including creation, identification, separation, preservation, and mixing. However, in the real world, it is usually impossible to know such information a priori. As a consequence, handling genetic linkage for genetic algorithms to succeed is extremely important.

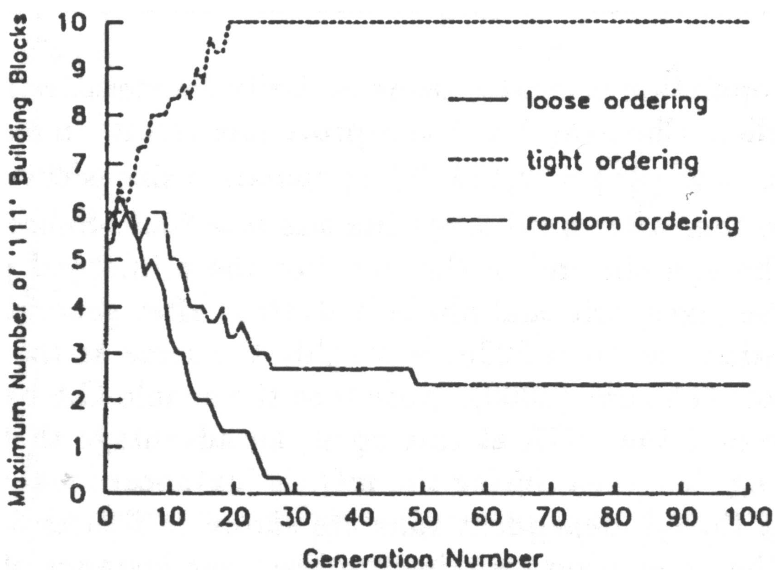


Figure 1.7: The generation maximum number of subfunctions optimized correctly by a simple GA graphed versus generation using tight, loose, and random orderings. The tight runs are able to optimize the function and the loose runs are not. The random runs get roughly 25% of the subfunctions correct (Goldberg, Korb, & Deb, 1989). Reprinted by permission.

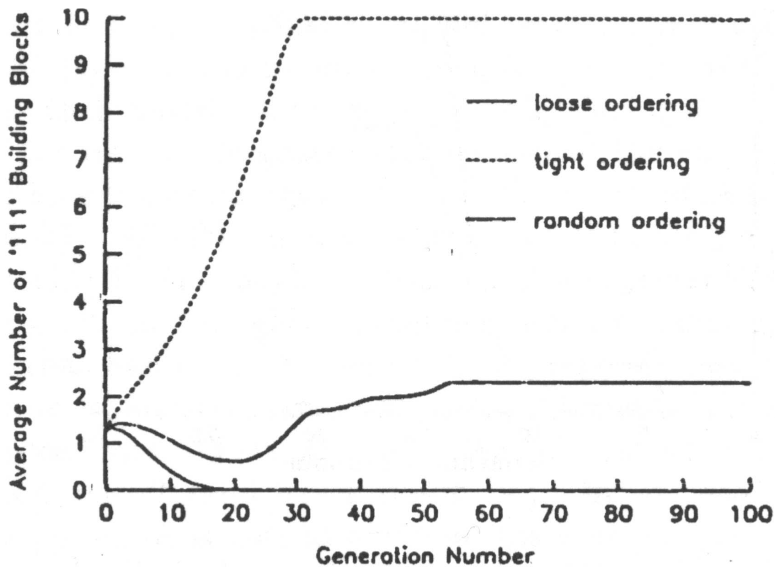


Figure 1.8: The generation average number of subfunctions optimized correctly by a simple GA graphed versus generation using tight, loose, and random orderings (Goldberg, Korb, & Deb, 1989). Reprinted by permission.

1.6 Summary

In this chapter, genetic algorithms were briefly introduced, including the terminology, working principles, major components, and algorithmic flow controls. The design-decomposition theory for developing competent selectorecombinative genetic algorithms were described, followed by the gambler's ruin model for population sizing which is employed in this study. Genetic linkage in both of the biological system and the genetic algorithm was defined. Because of the way genetic algorithms work as well as fixed genetic operators and chromosome representations that are widely used, the genetic linkage problem was also considered as an ordering problem in the literature, which refers to the need of appropriate arrangements of genes on the chromosome to ensure the success of genetic algorithms. Finally, the importance of learning genetic linkage in a genetic and evolutionary approach was discussed.

Chapter 2

Genetic Linkage Learning Techniques

The importance of learning genetic linkage has been discussed in the previous chapter and recognized in the field of genetic and evolutionary algorithms (Goldberg, 1989c; Goldberg, 2002; Holland, 1975). A design-decomposition methodology for successful design of genetic and evolutionary algorithms was proposed in the literature (Goldberg, 1991; Goldberg & Liepens, 1991; Goldberg, Deb, & Clark, 1992; Goldberg, 1993; Goldberg, 2002) and introduced previously. One of the key elements of the design-decomposition theory is genetic linkage learning. Research in the past decade have shown that genetic algorithms that are capable of learning genetic linkage and exploiting good building-block linkage can solve boundedly hard problems quickly, accurately, and reliably. Such *competent* genetic and evolutionary algorithms take the problems that were intractable for the first-generation genetic algorithms and render them practical in polynomial time (oftentimes, in subquadratic time) (Pelikan, Goldberg, & Cantú-Paz, 1999; Pelikan, Goldberg, & Cantú-Paz, 2000; Pelikan & Goldberg, 2001; Goldberg, 2002).

Because it is hard to guarantee that the user-designed chromosome representation provides tightly linked building blocks when the problem domain knowledge is unavailable, a variety of genetic linkage learning techniques have been proposed and developed to handle the linkage problem, which refers to the need of good building-block linkage. These genetic linkage learning techniques are so diverse, sophisticated, and highly integrated with the genetic algorithm that it is a difficult task to review all of them from a simple, unified, and straightforward point of view. Therefore, the purpose of this chapter is to provide different facetwise views of the existing genetic linkage learning techniques proposed in the literature, followed by a discussion of the lineage of the linkage learning genetic algorithm and its position among the existing genetic linkage learning techniques. Particularly, the following main topics are presented in this chapter:

- Review the existing genetic linkage learning techniques according to different facets of genetic algorithms, including
  - the means to distinguish good individuals from bad individuals;
  - the method to express or represent genetic linkage;
  - the way to store genetic linkage information.
- Present the lineage of the linkage learning genetic algorithm to demonstrate how it was developed and constructed from its precursors and ancestors;
- Identify the position of the linkage learning genetic algorithm among the existing genetic linkage learning techniques to establish its connection to other methodologies in general and to emphasize on its importance in particular.

This chapter starts with reviewing the existing genetic linkage learning techniques according to different facets, followed by the relations between the linkage learning genetic algorithm and its precursors or ancestors. Finally, the chapter discusses the position of the linkage learning genetic algorithm among the existing genetic linkage learning techniques based on the proposed aspects in this chapter.

2.1 Unimetric Approach vs. Multimetric Approach

In this section and the following two sections, we will review the existing genetic linkage learning techniques according to different facets and aspects, including the means to distinguish good individuals from bad individuals, the method to express or represent genetic linkage, and the way to store genetic linkage information. First, we start with classifying the genetic linkage learning techniques based on the means employed in the algorithm to distinguish good individuals from bad individuals in this section.

According to the means to distinguish good individuals in the population or a good model for generating individuals from bad ones, we can roughly classify existing genetic and evolutionary approaches into the following two categories:

Unimetric approach. A *unimetric* approach acts solely on the fitness value given by the fitness function. No extra criteria or measurements are involved for deciding whether an individual or a model is better.

Multimetric approach. In contrast to unimetric approaches, a *multimetric* approach employs extra criteria or measurements other than the fitness function given by the problem for judging the quality of individuals or models.

Unimetric approaches, loosely modeled after natural environments, are believed to be more biologically plausible, while multimetric approaches are of artificial design and employ certain bias which does not come from the problem at hand to guide the search. Specifically, the reasons and motivation to propose this classification to discriminate unimetric approaches and multimetric approaches are two-fold:

1. **Biological plausibility:** One of the most important reasons to propose this classification is that we believe nature appears unimetric. Because the "fitness" of an individual in nature depends on whether or not it can adapt to its environment and survive in its environment, there is obviously no other extra measurement or criterion to enforce or guide the evolution of the species to go to certain direction, such as becoming as simple as it can be. However, given the current research results in this field that most good evolutionary approaches are multimetric ones, which utilize one or more user-defined measurements to determine the solution quality, such as preference for simpler models,

we would like to separate unimetric approaches from multimetric ones and to know if there are limits to performance of unimetric methods. The theoretical results obtained on unimetric approaches might be of some significance or interests in biology, although the computational models are highly simplified.

2. **Technological motivations:** In addition to the biological viewpoints, there are also technological motivations to classify the existing genetic linkage learning techniques into unimetric approaches and multimetric approaches. For most multimetric methods, the algorithmic operations are serial in design, while unimetric methods are oftentimes easy to parallelize. The multimetric algorithms usually require access to all or a large part of the individuals in the population at the same time. This kind of requirement removes potential parallel advantages because it either incurs a high communication cost due to the necessary information exchange or demands a completely connected network topology to lower the communication latency. Therefore, it may be a foreseeable bottleneck when handling problems of a large number of variables. On the other hand, although many unimetric methods, such as the linkage learning genetic algorithm, do not perform as well as multimetric ones, they oftentimes use pairwise operators or operators that operate on only a few individuals. Hence, they are relatively easy to parallelize, and a wide range of parallelization methods are applicable.

According to these motivations, the means to distinguish good individuals in the population or a good model for generating individuals from bad ones is adopted to classify the existing genetic linkage learning techniques.

For example, because all the simple genetic algorithms and the linkage learning genetic algorithm use only fitness values to operate, they are definitely considered as unimetric approaches. Moreover, the simple genetic algorithms with *inversion* (Bagley, 1967), *punctuation marks* (Schaffer & Morishima, 1987), or the *linkage evolving genetic operator* (LEGO) (Smith & Fogarty, 1995; Smith & Fogarty, 1996), are also included in unimetric approaches because no extra measurements are utilized in these algorithms for comparing the solution or model quality.

On the other hand, most advanced genetic algorithms today, including the *gene expression genetic algorithm* (gemGA) (Kargupta, 1996), the *estimation of distribution algorithms* (EDAs) (Mühlenbein & Paaß, 1996), the *Bayesian optimization algorithm* (BOA) (Pelikan, Goldberg, & Cantú-Paz, 1999), the *extended compact genetic algorithm* (ECGA) (Harik, 1999; Lobo & Harik, 1999), and the like, are classified as multimetric approaches because they explicitly employ extra mechanisms or measurements for discriminating good individuals or models from bad ones. In addition to the obvious classification, approaches such as the *messy genetic algorithm* (mGA) (Goldberg, Korb, & Deb, 1989) and the *fast messy genetic algorithm* (fmGA) (Goldberg, Deb, Kargupta, & Harik, 1993) are in between the two classes. The messy genetic algorithm and the fast messy genetic algorithm compare individuals with the fitness value, but the use of building-block filtering indeed builds an implicit extra mechanism that prefers shorter building blocks into these algorithms.

Based on the classification of unimetric and multimetric approaches, the linkage learning genetic algorithm is especially of our interests because the difficulties encountered by the linkage learning genetic algorithm might indicate certain limits to genetic linkage learning for the unimetric approach. As mentioned previously, the most successful genetic and evolutionary approaches are all multimetric approaches. Even the fast messy genetic algorithm, which is in between the unimetric approach and the multimetric approach, is less successful and appears to have limits as well (Kargupta, 1995; Merkle, 1996). Thus, as a unimetric approach, we would like to know if the difficulties that the linkage learning genetic algorithm encountered are a solvable problem or intrinsic properties existing in the unimetric approach.

2.2 Physical Linkage vs. Virtual Linkage

After classifying the genetic linkage learning techniques according to the facet of how they distinguish good individuals or models from bad ones, in this section, we discuss the aspect of the method these algorithms use to express or represent genetic linkage. According to the method to represent genetic linkage, we can broadly classify existing genetic and evolutionary approaches into the following two categories:

Physical linkage. A genetic and evolutionary algorithm is said to use *physical linkage* if in this algorithm, genetic linkage emerges from physical locations of two or more genes on the chromosome.

Virtual linkage. On the other hand, if a genetic and evolutionary algorithm uses graphs, groupings, matrices, pointers, or other data structures that control the subsequent pairing or clustering organization of decision variables, it is said to use *virtual linkage*.

Physical linkage is closer to biological plausibility and inspired directly by it, while virtual linkage is an engineering or computer science approach to achieve the desired effect most expeditely. In particular, similar to the reasons that were discussed in the previous section, the motivations to look into this classification are also two-fold:

1. **Biological plausibility:** Because genetic and evolutionary algorithms are search techniques based on principles of evolution, it is one of our main interests to learn from nature and to borrow useful insights, inspirations, or mechanisms from genetics or biology. Given that the natural evolution apparently proceeds via genetic operations on the genotypic structures of all creatures, genetic and evolutionary algorithms that employ the mechanisms which are close to that in nature should be recognized and emphasized. By pointing out this feature of characteristic of the genetic and evolutionary algorithms that use the mechanisms existing in biological systems, we might be able to theorize certain genetic operations in biological systems with those genetic

algorithms using physical linkage, such as the fast messy genetic algorithm and the linkage learning genetic algorithm.

2. **Algorithmic improvement:** From a standpoint of efficient or effective computation, genetic and evolutionary algorithms using virtual linkage usually yield better performance than those using physical linkage. Together with the biological point of view, this might imply two possible situations:

   (a) Using virtual linkage in genetic algorithms can achieve a better performance. This kind of artificial systems can do better than their biological counterparts on conducting search and optimization;

   (b) The power of natural systems has not been fully understood and utilized yet. More critical and essential mechanisms existing in genetics and biology should be further examined and integrated into the algorithms to improve the performance.

   Hence, for the purpose of search and optimization, in the first situation, we should focus on developing better algorithms that employ virtual linkage, such as the *probabilistic model-building genetic algorithms* (PMBGAs) or EDAs (Mühlenbein & Paaß, 1996). In the other situation, we should appropriately choose useful genetic mechanisms and integrate these mechanisms into the artificial algorithms. The steps we will take in this study to improve the linkage learning genetic algorithm assume the second situation.

According to these motivations, the method to express or represent genetic linkage is used to classify the existing genetic linkage learning techniques in this section.

For example, all the genetic algorithms use fixed chromosome representations without any extra graph, grouping, matrix, pointer, or data structure to describe genetic linkage fall into the category of physical linkage. These algorithms include the ones using binary strings, integer strings, or real-variable strings as chromosomes as long as they use the chromosome alone for operations and evolution. Another important set of algorithms belonging to the category of physical linkage is the genetic algorithms that use the (*gene number*, *allele*) coding scheme (Bagley, 1967; Rosenberg, 1967). This set of genetic algorithms includes inversion (Bagley, 1967), the messy genetic algorithm (Goldberg, Korb, & Deb, 1989), the fast messy genetic algorithm (Goldberg, Deb, Kargupta, & Harik, 1993), and the linkage learning genetic algorithm.

Moreover, the category of virtual linkage includes all PMBGAs or EDAs (Larrañaga & Lozano, 2001; Pelikan, Goldberg, & Lobo, 2002), such as the Bayesian optimization algorithm (Pelikan, Goldberg, & Cantú-Paz, 1999), the extended compact genetic algorithm (Harik, 1999; Lobo & Harik, 1999), the *factorized distribution algorithm* (FDA) (Mühlenbein & Mahnig, 1999), and so on. It also contains the probabilistic inference framework for modeling crossover operators (Salman, Mehrotra, & Mohan, 1998; Salman, Mehrotra, & Mohan, 1999; Salman, Mehrotra, & Mohan, 2000), such as the *general linkage crossover* (GLinX) and the *adaptive linkage crossover* (ALinX).

The classification of physical linkage and virtual linkage based on the method to express or represent genetic linkage emphasizes on the mechanism of genetic operations and the representation that the algorithm uses for search or optimization. In this study, we are interested in understanding and improving the linkage learning genetic algorithm because the linkage learning genetic algorithm uses physical linkage, in which the genetic linkage of a building block is represented by the gaps on the chromosome between the genes of that building block. As discussed in this section, we assume that the power of natural systems has not been fully understood and utilized. After gaining better understanding of the linkage learning genetic algorithm, certain critical mechanisms existing in genetics will be integrated into the linkage learning genetic algorithms to improve its performance in this dissertation.

2.3 Distributed Model vs. Centralized Model

The last facet of the genetic and evolutionary algorithm we explore in this work for classifying the genetic linkage learning techniques is the way for these approaches to store genetic linkage information. Based on the way to store genetic linkage information, we can divide the existing genetic and evolutionary approaches into the following two categories:

Distributed Model. If a genetic and evolutionary algorithm has no centralized storage of genetic linkage information and maintains the genetic-linkage model in a distributed manner, we call such a genetic algorithm a *distributed-model* approach.

Centralized Model. In contrast to distributed-model approaches, a *centralized-model* approach utilizes a centralized storage of genetic linkage information, such as a global probabilistic vector or dependency table, to handle and process genetic linkage.

Similar to the unimetric approach, distributed-model approaches are also loosely modeled after evolutionary conditions in nature and more biologically plausible, while centralized-model approaches are developed to achieve the maximum information exchange and to obtain the desired results. The reasons to propose this classification to show the difference between distributed-model approaches and centralized-mode approaches are presented as follows:

1. **Biological plausibility:** Once more, we propose this classification in order to put an emphasis on the similarities as well as the dissimilarities between the genetic algorithms and the biological systems. Apparently, there exists no centralized genetic-linkage model in nature. Genotypes are distributed on all creatures or individuals. As described in the previous sections, genetic algorithms fall in the category of distributed model might serve as highly simplified computation models which can give insight of the way nature or evolution works.

2. **Computational motivations:** On the other hand, based on the classification, centralized-model approaches should be expected to have better performance when executing

computation, such as search or optimization, because by centralizing the genetic-linkage model, genetic-linkage information existing in the population gets well mixed and exchanged in very little time compared to that in a distributed-model approach. Therefore, centralized-model approaches have such an edge to outperform distributed-model. However, this advantage might also be a disadvantage for centralized-model approaches. Centralized-model approaches are serial in nature, and they are very hard to parallelize. Distributed-model approaches are parallel by design. Thus, distributed-model approaches might have better *scalability* when handling large-scale problems.

According to the above reasons, the way to store genetic linkage information is adopted to classify the existing genetic linkage learning techniques.

For example, simple genetic algorithms are distributed-model approaches because any information existing in the population is stored in a distributed manner over the individuals. The linkage learning genetic algorithm, the messy genetic algorithm (Goldberg, Korb, & Deb, 1989), the fast messy genetic algorithm (Goldberg, Deb, Kargupta, & Harik, 1993), and the *gene expression messy genetic algorithm* (gemGA) (Kargupta, 1996) also belong to this category for the same reason. Moreover, the linkage identification procedures, including the *linkage identification by nonlinearity check* (LINC) (Munetomo & Goldberg, 1998), the *linkage identification by non-monotonicity detection* (LIMD) (Munetomo & Goldberg, 1999), the *linkage identification based on epistasis measures* (LIEM) (Munetomo, 2002a), and the *linkage identification with epistasis measure considering monotonicity conditions* (LIEM^2) (Munetomo, 2002b), as well as the *collective learning genetic algorithm* (CLGA) (Riopka & Bock, 2000; Riopka, 2002) are in this class.

Furthermore, similar to the category of virtual linkage, the centralized-model approaches include most PMBGAs or EDAs (Larrañaga & Lozano, 2001; Pelikan, Goldberg, & Lobo, 2002), such as the Bayesian optimization algorithm (Pelikan, Goldberg, & Cantú-Paz, 1999), the extended compact genetic algorithm (Harik, 1999; Lobo & Harik, 1999), the factorized distribution algorithm (Mühlenbein & Mahnig, 1999), and so on. The probabilistic inference framework for modeling crossover operators (Salman, Mehrotra, & Mohan, 1998; Salman, Mehrotra, & Mohan, 1999; Salman, Mehrotra, & Mohan, 2000), such as the general linkage crossover and the adaptive linkage crossover, and the *dependency structure matrix driven genetic algorithm* (DSMGA) (Yu, Goldberg, Yassine, & Chen, 2003) are also considered as centralized-model approaches.

According to this classification, as a distributed-model approach, the linkage learning genetic algorithm is of our interests not only because it might serve as a computational model for the evolutionary process in nature but also because it has the potential to *scale up* well, compared to the centralized-model approaches. In the following section, the lineage of the linkage learning genetic algorithm is presented to demonstrate how it was developed and constructed from those algorithms and techniques proposed in the literature.

2.4 LLGA: Precursors and Ancestors

Although we will introduce in detail the linkage learning genetic algorithm in the next chapter, in this section, we present its precursors and ancestors proposed in the literature as a historical background. In particular, we describe the following topics in this section:

- Chromosome representation: The tag-based approach, or called messy coding, and the use of non-coding segments;

- Genetic operator: The inversion operator and the partially mapped crossover for reordering, the cut and splice operators for manipulating genetic material in the messy genetic algorithm and the fast messy genetic algorithm.

We start with the direct bloodline of the linkage learning genetic algorithm, which goes from the inversion operator to the messy genetic algorithm and the fast messy genetic algorithm. Then, combining with the use of non-coding segments and the partially mapped crossover, the lineage of the linkage learning genetic algorithm is complete.

First of all, the linkage learning genetic algorithm uses the (*gene number*, *allele*) coding scheme, which dates back to 1967 and was called the *tag-based* approach (Bagley, 1967) or later, *messy coding* (Goldberg, Korb, & Deb, 1989). As discussed in section 1.5.2, the linkage problem was considered as the ordering problem, and Bagley (1967) used the *inversion* operator to work with messy coding in order to appropriately arrange the genes on the chromosome such that the building blocks can survive the crossover disruption and transfer to the offspring by reversing the order of a segment of a chromosome encoded as a (*gene number*, *allele*) string. Although studies (Bagley, 1967; Rosenberg, 1967; Frantz, 1972) showed that the inversion operator was too slow for learning genetic linkage, Holland (1975) still recognized the importance of genetic linkage learning and suggested the use of it.

By using messy coding, the *messy genetic algorithm* (mGA) (Goldberg, Korb, & Deb, 1989; Goldberg, Deb, & Korb, 1990; Deb, 1991; Deb & Goldberg, 1991) attempted to achieve the competent genetic algorithm. Due to the use of messy coding, mGA handled *under-specified* and *over-specified* chromosomes and employed two messy operators, *cut* and *splice*, to process the individuals encoded as (*gene number*, *allele*) strings of variable lengths. The cut operator separated a string into two segments with a probability which increased with the string length, while the splice operator concatenated two strings to form one string with a fixed probability. A unique feature of mGA was that mGA utilized three heterogeneous phases: the *initialization* phase, the *primordial* phase, and the *juxtapositional* phase. Moreover, for the flow-control, mGA utilized the epoch-wise iteration, which was quite different from traditional genetic algorithms, to improve the solution quality in discrete steps.

Although mGA was able to solve hard problems accurately and reliably, it failed to be a competent genetic algorithm because $\mathcal{O}(\ell^k)$ computations were required in the initialization phase (Goldberg, Deb, & Korb, 1990), where ℓ was the problem length, and k was the order of building blocks. In order to satisfy the computational speed criterion, Goldberg, Deb, Kargupta, and Harik (1993) modified mGA and developed the *fast messy genetic algorithm* (fmGA), which can accurately and reliably solve difficult problem in subquadratic computational time. The first modification was to use the *probabilistically complete initialization* to replace the partial enumeration of building blocks so that the number of computations required in the initialization phase can be reduced to $\mathcal{O}(\ell)$. With the probabilistically complete initialization, the *building-block filtering* was designed to make the population composed of tightly linked building blocks by using selection and gene deletion repeatedly in the primordial phase. The third modification was to probabilistically increase the threshold number of genes employed for avoiding the competitions among irrelevant individuals during selection.

For the chromosome representation of the linkage learning genetic algorithm, the (*gene number*, *allele*) coding scheme is one of the key components. Another essential component proposed in the literature is the use of non-coding segments which was previously called *introns* in the chromosome representation. Levenick (1991) inserted non-coding segments into the chromosome representation and studied the effect. Forrest and Mitchell (1993) investigated into the effect of non-coding segments on the Royal Road functions. Thorough empirical studies on genetic algorithms with non-coding segments have been conducted (Wu, Lindsay, & Smith, 1994; Wu & Lindsay, 1995), and a survey on non-coding segments in genetics was given as well (Wu & Lindsay, 1996). By integrating the (*gene number*, *allele*) coding scheme and the non-coding segment, the chromosome representation of the linkage learning genetic algorithm was constructed after making the chromosome a circle and enabling the genes and non-coding elements to move freely around the chromosome.

After constructing the chromosome representation, a genetic operator which is capable of handling this type of chromosomes is in order for the linkage learning genetic algorithm. Goldberg and Lingle (1985) argued that the inversion operator was not sufficiently powerful to solve the ordering problem and proposed the *partially mapped crossover* (PMX) (Goldberg & Lingle, 1985) which allowed the exchange of both ordering and allele information between chromosomes during crossover events so that genetic linkage can be changed along with the allele configuration. In that study, they showed that PMX was able to solve a medium-sized traveling salesman problem to its optimum solution and the binary operator was more capable than the earlier unary operator, the inversion operator. Based on the idea of the cut and splice operators in mGA and fmGA which dissect and reassemble the genetic material, the exchange crossover operator was developed with the hint from PMX that genetic linkage and allele configuration should evolve along with each other.

Two key components—the chromosome representation and the exchange crossover operator—of the linkage learning genetic algorithm were constructed and developed based on the historical work presented in this section. For the chromosome representation, Harik (1997)'s contribution was to design the chromosome as a circle, to permit the genes and non-coding elements to reside anywhere on the chromosome, and to propose the probabilistic expression mechanism for interpreting the chromosome. For the exchange crossover operator, Harik (1997) combined these ideas from different directions and integrated them together to work on the chromosome representation. As a consequence, keys components of the linkage learning genetic algorithm were completed.

2.5 LLGA: Unimetric, Physical Linkage, and Distributed Model

After reviewing the genetic linkage learning techniques from different aspects of genetic algorithms and introducing the historical work based on that the linkage learning genetic algorithm was developed, in this section, we recap three classifications and point out the position of the linkage learning genetic algorithm—unimetric, physical linkage, and distributed model—among the existing genetic linkage learning techniques.

First of all, because the linkage learning genetic algorithm use only fitness alone for distinguishing solutions of good quality from that of bad quality, it is classified as a *unimetric* approach. As a unimetric approach, the linkage learning genetic algorithm is of our interests because the difficulties encountered by the linkage learning genetic algorithm might indicate certain limits to genetic linkage learning for the unimetric approach. Currently, the most successful genetic and evolutionary approaches are multimetric approaches. Even the fast messy genetic algorithm, which is in between the unimetric approach and the multimetric approach, as described in section 2.1, is less successful and appears to have limits. Thus, as a unimetric approach, we would like to know if the difficulties that the linkage learning genetic algorithm faced are a solvable problem or intrinsic properties of the unimetric approach.

Second, according to the second classification, the linkage learning genetic algorithm uses *physical linkage*. In this study, we are interested in understanding and improving the linkage learning genetic algorithm because the linkage learning genetic algorithm uses physical linkage, in which the genetic linkage of a building block is represented by the gaps on the chromosome between the genes of that building block. As discussed in section 2.2, we assume that the power of natural systems has not been fully understood and utilized. After gaining better understanding of the linkage learning genetic algorithm, certain critical mechanisms existing in genetics should be integrated into the linkage learning genetic algorithms to improve its performance.

Finally, the linkage learning genetic algorithm is obviously a *distributed-model* approach because there exists no centralized genetic-linkage model or global linkage collecting and processing in the linkage learning genetic algorithm. According to this classification, as a distributed-model approach, the linkage learning genetic algorithm is of our interests not only because it might serve as a computational model for the evolution in nature but also because it has the potential to *scale up* well, compared to the centralized-model approaches.

2.6 Summary

As pointed out by Holland (1975), learning genetic linkage is essential to the success of genetic and evolutionary algorithms if the prior knowledge of the problem is unavailable. Recognizing the importance of solving the linkage problem, many genetic linkage learning techniques have been proposed to tackle the linkage problem. These genetic linkage learning techniques are so diverse, sophisticated, and highly integrated with the genetic algorithm that it is difficult to review all of them from a simple, unified, and straightforward viewpoint.

This chapter provided different facetwise views of the existing genetic linkage learning techniques, including the means to distinguish good individuals from bad ones, the method to represent genetic linkage, and the way to store genetic linkage information. Then, it presented a discussion of the lineage of the linkage learning genetic algorithm and its position among the existing genetic linkage learning techniques according to three different classifications.

Chapter 3

Linkage Learning Genetic Algorithm

In order to handle linkage evolution and to tackle the ordering problem, Harik (1997) took Holland's call for the evolution of tight linkage quite literally and proposed the *linkage learning genetic algorithm* (LLGA), which is capable of learning genetic linkage in the evolutionary process. The linkage learning genetic algorithm used a special probabilistic expression mechanism and a unique combination of the (*gene number*, *allele*) coding scheme and an exchange crossover operator to create an evolvable genotypic structure that made genetic linkage learning natural and viable for genetic algorithms. As the subject of this study, the design, works, accomplishments, and limitations of the linkage learning genetic algorithm are presented and discussed in this chapter. Detailed background and comprehensive description can also be found elsewhere (Harik & Goldberg, 1996; Harik, 1997; Harik & Goldberg, 2000).

This chapter presents the following topics of the linkage learning genetic algorithm:

- the chromosome representation;
- the exchange crossover operator;
- two mechanisms that enable the LLGA;
- accomplishments of the LLGA;
- difficulties encountered by the LLGA.

We start with the design of the linkage learning genetic algorithm, including the chromosome representation and the exchange crossover operator. Then, the two identified existing linkage learning mechanisms are described, followed the accomplishments and limitations of the linkage learning genetic algorithms.

3.1 Chromosome Representation

The linkage learning genetic algorithm adopts a unique chromosome representation, which is an integration of several important components that makes the linkage learning genetic algorithm capable of expressing genetic linkage via the chromosomal structure and gene arrangement. The LLGA chromosome representation is composed of

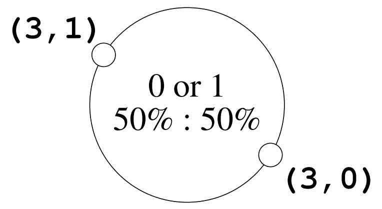
- moveable genes;
- non-coding segments;
- probabilistic expression.

Each of these elements is described in what follows.

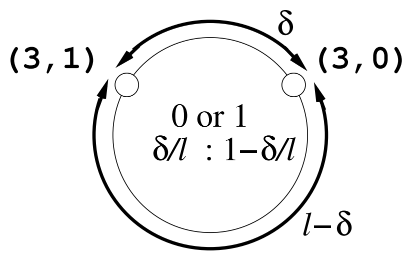
The LLGA chromosome consists of moveable genes encoded as (*gene number*, *allele*) pairs as shown in Figure 3.1(a) and is considered as a circle of circumference 1.0. The genes in the linkage learning genetic algorithm are allowed to reside anywhere on the chromosome, while those in a traditional genetic algorithm are unmoveable and fixed at their own loci.

In order to create a genotypic structure capable of expressing genetic linkage, non-coding segments are included on the LLGA chromosome. Non-coding segments act as non-functional genes, which have no effect on fitness at all. By using non-coding segments, genes no longer have to connect to each other, and genetic linkage can be expressed more accurately.

Furthermore, the method of *probabilistic expression* (PE) was proposed to preserve diversity at the building-block level. A PE chromosome contains all possible alleles for each gene.



(a) Gene 3 is expressed as 0 with probability 0.5 and 1 with probability 0.5.



(b) Gene 3 is expressed as 0 with probability δ/l and 1 with probability $1 - \delta/l$.

Figure 3.1: Probability distributions of gene 3's alleles represented by PE chromosomes.

For the purpose of evaluation, a chromosome is *interpreted* by selecting a *point of interpretation* (POI) and choosing for each gene the allele occurring first in a clockwise traversal of the circular LLGA chromosome. After interpretation, a PE chromosome is expressed as a complete string and evaluated to obtain the fitness. Besides, the point of interpretation is determined on the offspring individual when crossover occurs.

As a consequence, a PE chromosome represents not a single solution but a probability distribution over the range of possible solutions which might be expressed when different points of interpretation are chosen. Figure 3.1 illustrates the probability distribution over possible alleles of gene 3 of the chromosome. As shown in the figure, when different points of interpretation are selected, a PE chromosome might be interpreted as different solutions. Figure 3.2 shows 3 genes of a PE chromosome composed of 6 genes. If point A is the point of interpretation, the chromosome will be considered as `((5,1) (4,0) (4,1) (3,0) (3,1) (5,0))` and interpreted as `((5,1) (4,0)` ~~`(4,1)`~~ `(3,0)` ~~`(3,1)`~~ ~~`(5,0)`~~`)` $\Rightarrow$ ***001, where the struck genes are *shadowed* by their complement genes. And, if point B is the point of interpretation, the chromosome will be considered as `((4,0) (4,1) (3,0) (3,1) (5,0) (5,1))` and interpreted as `((4,0)` ~~`(4,1)`~~ `(3,0)` ~~`(3,1)`~~ `(5,0)` ~~`(5,1)`~~`)` $\Rightarrow$ ***000.

If we consider a PE chromosome as containing exactly one copy of each shadowed gene, the probabilistic expression can be generalized to let a chromosome contain more than one

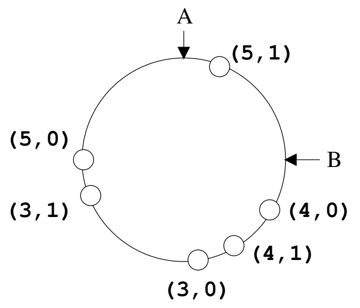


Figure 3.2: Different points of interpretation might interpret a PE chromosome as different solutions.

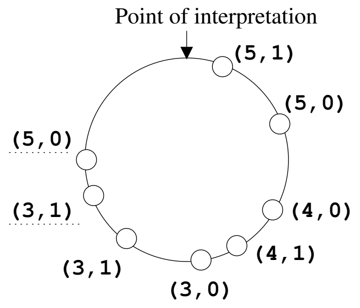


Figure 3.3: Example of an EPE-2 chromosome. Each gene can have *up to* 2 copies with the complement allele.

copy of a shadowed gene (Harik, 1997), which is called *extended probabilistic expression* (EPE). EPE is defined with a parameter k, which indicates that there can be *up to* k copies of a shadowed gene for any given gene. That is, an EPE-k chromosome can contain at most k copies of a shadowed gene. For example, Figure 3.3 shows an illustrative example of EPE-2 chromosomes, which is `((5,1) (5,0) (4,0) (4,1) (3,0) (3,1) (3,1) (5,0))`. Given the point of interpretation, the chromosome is interpreted as `((5,1)` ~~`(5,0)`~~ `(4,0)` ~~`(4,1)`~~ `(3,0)` ~~`(3,1)`~~ ~~`(3,1)`~~ ~~`(5,0)`~~`)` $\Rightarrow$ ***001. In this example, gene 4 has only one copy of the shadowed gene, `(4,1)`, as it can have in a PE chromosome, but both genes 3 and 5 have two copies of the shadowed genes. Therefore, when using EPE-k, a shadowed gene can have from 1 to k copies on the chromosome.

3.2 Exchange Crossover

In addition to the probabilistic expression and extended probabilistic expression mechanisms, the *exchange crossover* operator is another key mechanism for the linkage learning genetic algorithm to learn genetic linkage. Exchange crossover is defined on a pair of LLGA chromosomes. One of the chromosomes is the *donor*, and the other is the *recipient*. The operator cuts a *random* segment of the donor, selects a grafting point at random on the recipient, and grafts the segment cut from the donor onto the recipient. The grafting point is the point of interpretation of the generated offspring. Starting from the point of interpretation, the redundant genetic material caused by injection is removed right after crossover to ensure the validity of the offspring.

3.3 Linkage Definition and Two Linkage Learning Mechanisms

With the integration of probabilistic expression and exchange crossover, the linkage learning genetic algorithm is capable of solving difficult problems without prior knowledge of good linkage. Traditional genetic algorithms have been shown to perform poorly on difficult problems (Thierens & Goldberg, 1993; Goldberg, Deb, & Thierens, 1993) without such knowledge. To better decompose and understand the working behavior of the linkage learning genetic algorithm, two key mechanisms of linkage learning: *linkage skew* and *linkage shift* has been identified and analyzed (Harik & Goldberg, 1996; Harik, 1997). Therefore, in order to introduce the current theoretical framework of the linkage learning genetic algorithm, we present the following topics in this section:

- The method to quantify the genetic linkage of a building block which enables the development of quantitative models of the linkage learning mechanisms;
- Linkage skew, which occurs when an optimal building block is transferred from the donor to the recipient;
- Linkage shift, which occurs when an optimal building block resides in the recipient and survives an injection.

In this section, we will introduce Harik's definition for quantifying genetic linkage (Harik & Goldberg, 1996), followed by presenting the two linkage learning mechanisms identified by Harik and Goldberg (1996).

3.3.1 Quantifying Linkage

For studying the linkage learning process, Harik's definition for quantifying linkage (Harik & Goldberg, 1996) is adopted in this study. Linkage is defined as the sum of the squares of the inter-gene distances of a building block, considering the chromosome as a circle of circumference 1.0. Assuming that there is an order-k building block, k genes on the circular chromosome split the circle into k segments, or gaps. Number these k gaps 1, 2, …, k, and let y_i represent the length of gap i for all $i = 1, 2, \ldots, k$. Since the circumference of the circle is 1.0, we have the relationship among the lengths of these k gaps as

$$\sum_1^k y_i = 1 \; . \tag{3.1}$$

Then, Harik's definition for quantifying linkage, λ, can be expressed as

$$\lambda = \sum_1^k y_i^2 \; . \tag{3.2}$$

Figure 3.4 shows an example for calculating the linkage of a three-gene building block. The definition is appropriate in that genetic linkage in such definition specifies a measure directly proportional to the probability for a building block to be preserved under exchange crossover. It was theoretically justified by Harik and Goldberg (1996) that for any linkage learning operator working on the same form of chromosome representation, the expected linkage of a randomly spaced order-k building block is

$$\frac{2}{k+1} \; , \tag{3.3}$$

which also represents the average random linkage in the initial population of the linkage learning genetic algorithm.

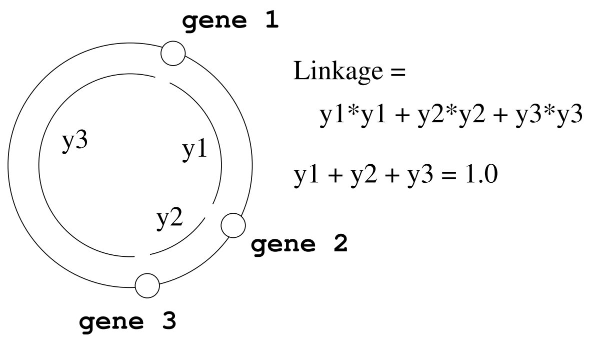


Figure 3.4: Calculation of the genetic linkage for a three-gene building block.

3.3.2 Linkage Skew

After quantifying the genetic linkage of a building block, we are now ready to develop the quantitative models for the linkage learning mechanisms. We present the linkage-skew model in this section, and then linkage-shift model in the following section.

Linkage skew, the first linkage learning mechanism (Harik & Goldberg, 1996), occurs when an optimal building block is successfully transferred from the donor into the recipient. The conditions for an optimal building block to be transferred are (1) the optimal building block resides in the cut segment and (2) the optimal building block gets expressed before an inferior one does. The effect of linkage skew was found to make linkage distributions move toward higher linkages by propagating tightly linked building blocks all over the population. Linkage skew does not make the linkage of a building block of any particular individual tighter because it occurs when an optimal building block is transferred and expressed. The transferred building block has the same linkage as it has before crossover. However, the tighter the building block is, the higher probability it gets transferred as a whole. Therefore, linkage skew helps genetic linkage learning by propagating tightly linkage building blocks.

Let $\Lambda_t(\lambda)$ be the probability density function of the random variable λ, which represents the genetic linkage of the optimal building block at generation t. The following linkage-skew model to describe the evolution of genetic linkage under linkage skew only has been proposed by Harik and Goldberg (1996):

$$\overline{\Lambda_{t+1}} = \overline{\Lambda_t} + \frac{\sigma^2 \left(\Lambda_t\right)}{\overline{\Lambda_t}} \ , \tag{3.4}$$

where $\sigma^2 \left(\Lambda_t\right)$ is the variance in the linkage distribution at time t.

3.3.3 Linkage Shift

Linkage shift is the second linkage learning mechanism (Harik & Goldberg, 1996). It occurs when an optimal building block resides in the recipient and survives a crossover event. For the optimal building block to survive, there cannot be any gene contributing to a deceptive building block transferred. If some genes of a deceptive building block get transferred to the recipient, the optimal building block on the recipient will be disrupted because the transferred genes get expressed before the genes of the optimal building block do. Therefore, linkage shift gets the linkage of a building block in an individual tighter by deleting duplicate genetic material which is unrelated to the optimal building block and caused by injection of exchange crossover. Compared to linkage skew, linkage shift gets genetic linkage of building blocks in each individual tighter by removing gaps consisting of non-coding elements or unrelated functional genes between the genes contributing to the optimal building block.

In addition to the model for linkage skew, Harik and Goldberg (1996) also proposed the following recurrence equation to describe and understand linkage shift:

$$\overline{\lambda_0(t+1)} = \lambda_0(t) + (1 - \lambda_0(t)) \frac{2}{(k+2)(k+3)} \ , \tag{3.5}$$

for an order-k building block.

3.4 Accomplishments of the LLGA

As presented in the previous sections, by integrating the evolvable genotypic structure consisting of moveable genes, non-coding segments, and probabilistic expression, the linkage learning genetic algorithm successfully achieves genetic linkage learning and handles the linkage problem. More specifically, the major accomplishments of the linkage learning genetic algorithm include:

- Proposed probabilistic expression to level the battlefield of the race between allelic convergence and linkage convergence;
- Identified the two linkage learning mechanisms to understand the linkage learning process in a bottom-up manner;
- Achieved successful genetic linkage learning on problems composed of exponentially scaled building blocks.

Each of these accomplishments is described in what follows.

As mentioned previously, there is a *race* or *time-scale comparison* in the genetic linkage learning process, which was suggested by Goldberg and Bridges (1990). Denote the characteristic time of allelic convergence t_α and the characteristic time of linkage convergence t_λ, because selection works on the fitness to promote good alleles and demote bad ones, allele convergence receives stronger and more direct signal from the selection force than linkage convergence does, sets of alleles converge more quickly than linkage does. That is, $t_\alpha < t_\lambda$. Such a condition leads to the failure of genetic algorithms because loose linkage prevents genetic algorithms from getting correct alleles, and once the alleles converge to wrong combinations, the result cannot be reversed or rectified.

One of the most important accomplishments of the linkage learning genetic algorithm is the use of probabilistic expression. By employing probabilistic expression, allelic convergence is effectively slowed down because the shadowed genes can exist on the chromosome throughout the evolutionary process and can get expressed again with certain probability even at the late stage of the run. This delay of allelic convergence gives a chance for genetic linkage to converge. Therefore, the linkage learning genetic algorithm makes linkage convergence not slower than allele convergence ($t_\lambda \leq t_\alpha$) by using probabilistic expression.

Moreover, Harik identified the two linkage learning mechanisms, linkage skew and linkage shift, of the linkage learning process by theorizing how the genetic linkage of a building block gets tight under the exchange crossover operator. By doing so, Harik not only provided an explanatory theory to describe the way the genetic linkage is learned and to predict the evolution of the genetic linkage of a single building block under these two mechanisms but also started a fundamental framework based on which we develop more comprehensive theoretical models, including the tightness time and the convergence time models, in this study to advance our understandings of the linkage learning genetic algorithm in theory and to propose a better design of the linkage learning genetic algorithm.

Overall, the linkage learning genetic algorithm successfully achieved genetic linkage learning on problems composed of badly scaled building blocks. With its sophisticated, unique design, the linkage learning genetic algorithm can keep allelic diversity until linkage convergence and solve exponentially scaled problems quickly, reliably, and accurately. Figure 3.5 shows a successful run of applying the linkage learning genetic algorithm to solve a 76-bit problem composed of 19 exponentially scaled building blocks. Each building block is an order-4 trap as described in section 4.2. The scaling factor to construct the test problem is 7. As shown in the figure, the linkage learning genetic algorithm successfully identified each building block in equal increments of time and solved the whole problem. The overall convergence time for the linkage learning genetic algorithm on exponentially scaled problems is linear to the problem size in terms of the number of building blocks.

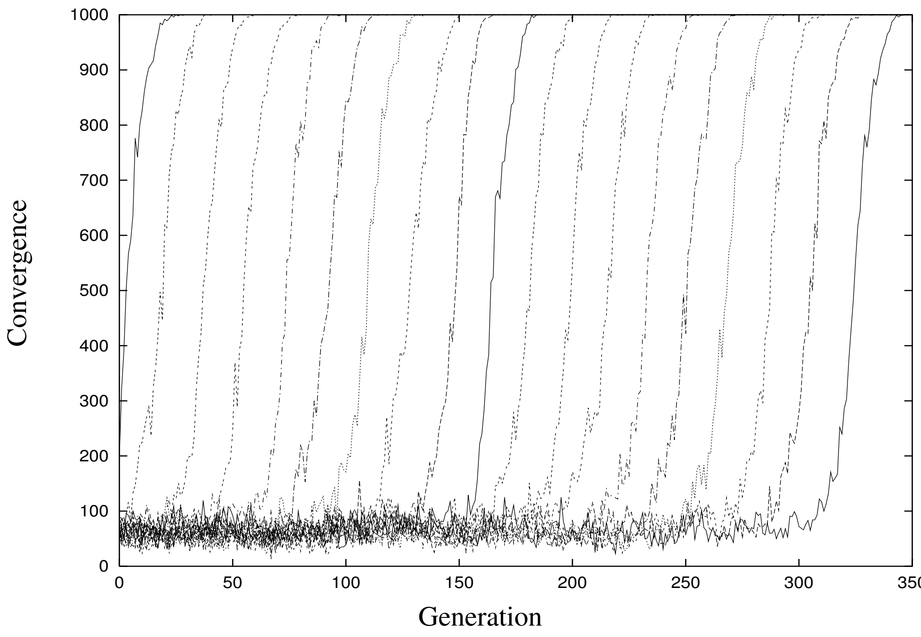


Figure 3.5: The linkage learning genetic algorithm is able to solve a 76-bit problem composed of 19 exponentially scaled order-4 traps with a scaling factor 7. Each building block is identified and gets tightly linked in equal increments of time. The overall convergence time is linear to the problem size in terms of the number of building blocks (Harik, 1997).

3.5 Difficulties Faced by the LLGA

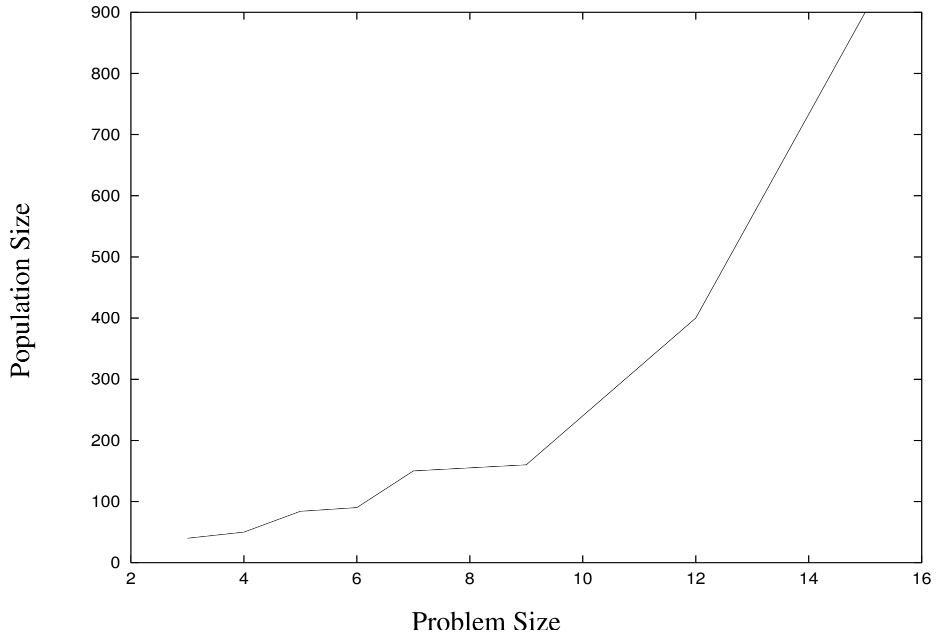
As described in the previous section, the linkage learning genetic algorithm can successfully learn genetic linkage and tackle the linkage problem to solve problems consisting of exponentially scaled building blocks by slowing down the allelic convergence with probabilistic expression. some extent. However, the linkage learning genetic algorithm does not perform as well as Harik and Goldberg originally expected on the uniformly scaled problem. As mentioned previously and reported by Harik (1997), when the building blocks of the problem are exponentially scaled, the linkage learning genetic algorithm can solve it in linear time in terms of the number of building blocks, but when the building blocks are uniformly scaled, the linkage learning genetic algorithm needs a population size growing exponentially with the problem size. Figure 3.6 shows that the linkage learning genetic algorithm requires an exponentially growing population size in terms of the number of building blocks to solve the uniformly scaled problem. Moreover, in our recent study, if we do not use a population size growing exponentially, the time required to solve a uniformly scaled problem grows exponentially as shown in Figure 6.4 in section 6.3.1.

Based on the performance reported on exponentially scaled building blocks and uniformly scaled building blocks, the linkage learning genetic algorithm seems to behave differently when handling different types of problems. Why the behavior of the linkage learning genetic algorithm seems inconsistent when solving multiple building blocks of different scalings was previously unknown, and it is one of the important questions that is answered in this study.

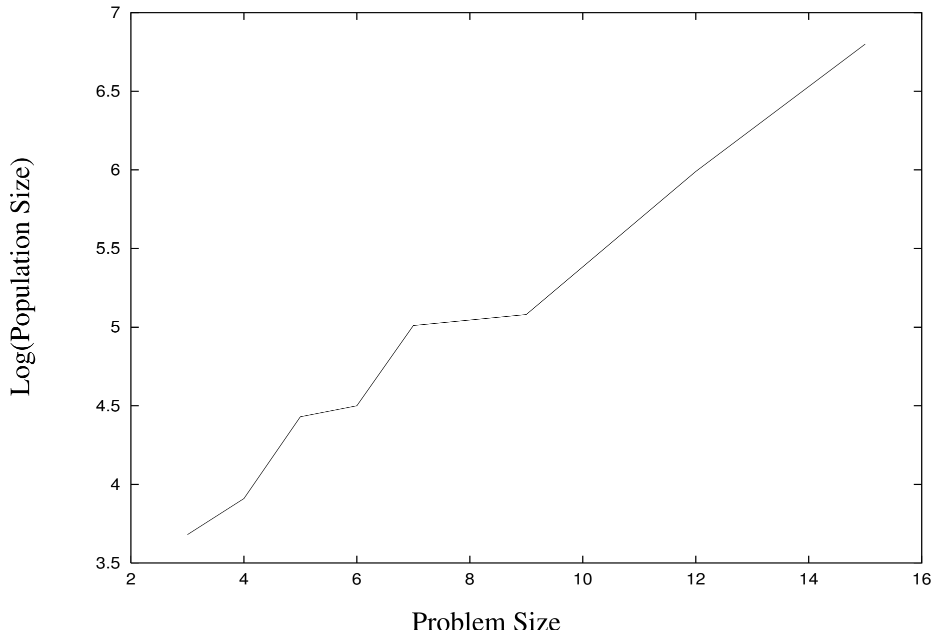
After resolving the issue of the seemingly inconsistent behavior and identifying the consistent sequential behavior, another important objective of this work is to propose effective modifications to eliminate the performance barrier of the linkage learning genetic algorithm on the uniformly scaled problem. A good number of efforts and attempts have been made to improve the performance of the linkage learning genetic algorithm in the past few years, including utilizing niching techniques (Harik, 1997), compressing the non-coding segments (Lobo, Deb, Goldberg, Harik, & Wang, 1998), employing multiple points of interpretation (Lobo, 2001), investigating the properties of building blocks of different scalings (Lobo, 2001), applying various selection pressure, crossover probability, and population sizes (Harik, 1997; Chen, 2002). These attempts either did not achieve any improvement or provided only marginal improvements. Therefore, in a few words, there are undiscovered mysteries, mechanisms, or limits of the linkage learning genetic algorithm. This study seeks better understanding of these facets of the linkage learning genetic algorithm and tries to provide solutions or explanations to the difficulties faced by the linkage learning genetic algorithm.

3.6 Summary

In this chapter, key elements of the linkage learning genetic algorithm, such as circular chromosomes, exchange crossover, and the two previously identified linkage learning mechanisms were introduced. Then, the accomplishments of the linkage learning genetic algorithm were



(a)



(b) semi-log scale

Figure 3.6: The linkage learning genetic algorithm requires an exponentially growing population size to solve a problem composed of uniformly scaled building blocks, which are order-4 tmmps. Straight lines on a semi-log scale are indicative of exponential growth (Harik, 1997).

presented and described. Finally, the difficulties faced by the linkage learning genetic algorithm were discussed, followed by the motivations to conduct this research. In what follows, a chapter presents the assumptions regarding the framework based on which we develop the theoretical models and regarding the genetic algorithm structure we adopt in this work. It provides a background establishment for the remainder of this dissertation.

Chapter 4

Preliminaries: Assumptions and the Test Problem

After introducing the background and motivation of the linkage learning genetic algorithm, we will start to improve and understand the linkage learning genetic algorithm by proposing practical mechanisms as well as developing theoretical models. However, before going further, we need to establish a ground based on which the models will be constructed and the experiments will be conducted in the following chapters. In particular, this chapter focuses the following topics:

- Describe the framework based on which we develop the models and present the genetic algorithm structure we used in this study;
- Introduce the test problem, including the elementary problem and the way to construct larger problems, that we employ throughout this work.

This chapter starts with introducing the key assumptions that we use throughout this study, followed by the definition and construction of the test problem.

4.1 Assumptions

The primary assumptions used in this study include:

- Selectorecombinative genetic algorithms;

- Generational genetic algorithms;
- Non-overlapping population;
- Fixed population size;
- Stationary fitness function;
- Uniformly or exponentially scaled building blocks.

Each of these assumptions is described and discussed in detail as follows:

- **Selectorecombinative genetic algorithms.** In this work, a selectorecombinative genetic algorithm is assumed. Selectorecombinative genetic algorithms are those genetic algorithms that use only selection and crossover as the genetic operators. According to the concept of the innovation intuition proposed by Goldberg (2002), the combination of selection and crossover is compared to the cross-fertilizing innovation, which performs global search and is more difficult to understand, whereas the combination of selection and mutation is a form of hillclimbing, which performs local search and easier to understand. Furthermore, in traditional genetic algorithms, crossover, or the recombination operator, is the main operator, and mutation is considered as a background operator applied with a low probability. Therefore, the combination of selection and crossover largely determines the behavior and performance of the search procedure, and we concentrate on selectorecombinative genetic algorithms in this work.

- **Generational genetic algorithms.** The flow-control of the genetic algorithm we consider in this work is generationwise. In generationwise genetic algorithms, evolution occurs through discrete steps. At one generation, all the individuals in the population are evaluated. Highly fit individuals get selected by the selection procedure and are recombined by the crossover operator to generate new individuals. The procedure is repeated until the new population is full, and then, the next generation starts.

- **Non-overlapping population.** We assume a non-overlapping population in this study. The new individuals generated by the combination of selection and crossover replace all of the old individuals. For example, if we start the genetic algorithm with a population of size n, the genetic operators, which are selection and crossover, create n new individuals, and then, these n new individuals replace the old n individuals. It is possible the some of the generated new individuals are identical to some of the old individuals, but the non-overlapping population means no old individual directly goes into the next population regardless of what the individual is.

- **Fixed population size.** The population size is assumed constant. The constant population size assumption indicates that the number of individuals created by the genetic operators is kept equal to the size of the initial population. Therefore, the population size does not change over generations.

- **Stationary fitness function.** The fitness function we consider in this work is assumed stationary. That is, the fitness function is not time-dependent or time-variant. Although we are using a fitness function without noise to test the linkage learning genetic algorithm in this study, the fitness function can be noisy or stochastic.

- **Uniformly or exponentially scaled building blocks.** We assume that the contribution to the overall fitness of the building blocks are either uniformly scaled or exponentially scaled. These two scalings are employed not only because of their prevalence in the literature but also because they are abstract versions of many decomposable problems (Goldberg, 2002). Uniformly scaled problems resemble those with subproblems of equal importance, while exponentially scaled problems represent those with subproblems of distinguishable importance. We will describe these two scaling methods in detail in the following section on the construction of the test problem.

Based on these assumptions, we develop the theoretical modes to describe the behavior of the linkage learning genetic algorithms and implement the algorithm to conduct the experiments for observation and verification. In the next section, we will describe the test problem employed throughout this work for studying the linkage learning genetic algorithm.

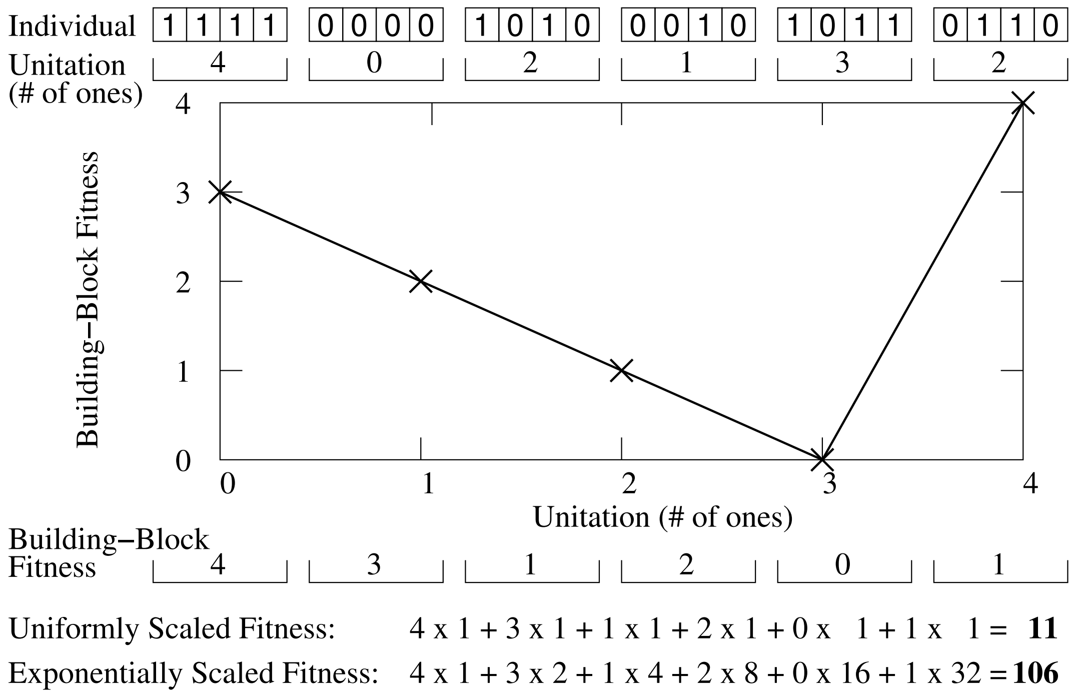
4.2 Test Problem

In this research project, a family of trap functions (Ackley, 1987; Deb & Goldberg, 1993; Deb, Horn, & Goldberg, 1993; Deb & Goldberg, 1994) is adopted as the elementary test problem. We use the trap function as the subproblem to construct all of the larger test problems used in the experiments in this study. The most important reason to choose trap functions as our test problems is that trap functions provide decent linkage structures among variables, and good genetic linkage is necessary for solving the problems consisting of traps. Since we concentrate on studying genetic linkage of building blocks and the linkage learning process, instead of other arbitrary functions, which might provide no linkage structure among variables or of which the properties are unknown to us, trap functions are very appropriate for our purpose of study.

A trap function is a piecewise-linear function defined on *unitation*, which is the number of ones in a binary input string. The function divides the variable domain into two sets. One of them leads to a global optimum, and the other leads to a local optimum. Specifically, an order-k trap function can be expressed as

$$\mathrm{trap}_k(u) = \begin{cases} u & u = k \\ k - 1 - u & \text{otherwise} \end{cases} . \tag{4.1}$$

Figure 4.1 shows the order-4 trap function used in most of the experiments in the present work. Note that for this particular order-4 trap function, the ratio r of the local optimum and the global optimum is $r = 3/4 = 0.75$. As indicated by Deb and Goldberg (1993), this



Individual
1 1 1 1
0 0 0 0
1 0 1 0
0 0 1 0
1 0 1 1
0 1 1 0
Unitation (# of ones)
4
0
2
1
3
2
Building−Block Fitness
0
1
2
3
4
Unitation (# of ones)
Building−Block Fitness
4
3
1
2
0
1
Uniformly Scaled Fitness: 4 x 1 + 3 x 1 + 1 x 1 + 2 x 1 + 0 x 1 + 1 x 1 = 11
Exponentially Scaled Fitness: 4 x 1 + 3 x 2 + 1 x 4 + 2 x 8 + 0 x 16 + 1 x 32 = 106

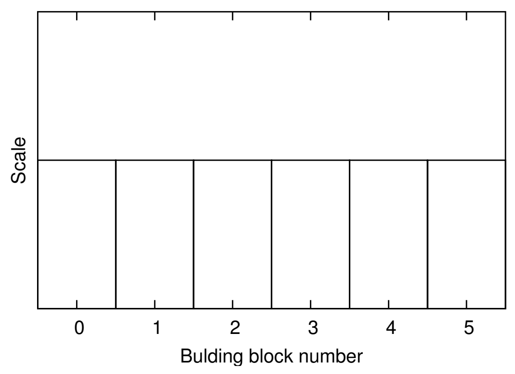
Figure 4.1: The order-4 trap function used in this study and two examples for concatenating six, order-4 trap functions—one is uniformly scaled and the other is exponentially scaled—to form larger test problems. The scaling factor is 2 in the exponential scaling example.

order-4 trap is fully deceptive because

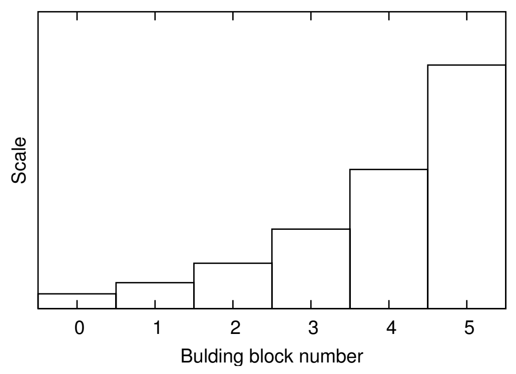
$$r = 0.75 \geq r_{min} = \frac{k-1}{2k-3} = \frac{4-1}{8-3} = 0.6 \ ,$$

where $k = 4$ for order-4 trap functions. This order-4 trap function is chosen as the elementary test problem because it is fully deceptive, and we are interested in solving the problems of bounded difficulty as discussed in section 1.2.

Moreover, in order to construct larger test problems, we mainly consider two types of scaling methods to combine these traps, which are considered as elementary subproblems. One method is to scale them uniformly, and the other is to scale them exponentially. These two scalings are employed not only because of their prevalence in the literature of genetic algorithms but also because they are abstract versions of many decomposable problems (Goldberg, 2002). Uniformly scaled problems resemble the decomposable problems with



(a) Uniformly scaled building blocks



(b) Exponentially scaled building blocks

Figure 4.2: Two different scaling methods used in this study to concatenate multiple building blocks to construct larger test problems. As shown in (a), each of the uniformly scaled building blocks contributes to the fitness equally, while as shown in (b), the salience of each exponentially scaled building block is scaled exponentially according to some scaling factor.

subproblems of equal salience, while exponentially scaled problems represent those with subproblems of distinguishable importance. As an illustration, Figure 4.2 shows these two kinds of scalings for a six-building-block problem. Note that in exponentially scaled problems, for convenience, we scale the contribution to fitness of each building block according to its position, but the idea is only to create building blocks of distinguishable salience.

4.3 Summary

In this chapter, the assumptions regarding the framework based on which we develop the theoretical models as well as regarding the genetic algorithm structure we adopted in this work were described. After introducing the key assumptions that we employ throughout this dissertation, the definition of the elementary test problem, which are trap functions, and the construction of the larger test problems used in the study were presented in detail. In the following chapters, we will start to improve and understand the linkage learning genetic algorithm by proposing practical mechanisms as well as developing theoretical models in order to achieve scalable genetic linkage learning for a unimetric approach.

Chapter 5

A First Improvement: Using Promoters

Harik (1997) took Holland (1975)'s call for evolution of tight genetic linkage and proposed the linkage learning genetic algorithm (LLGA), which used a special probabilistic expression mechanism and a unique combination of the (*gene number*, *allele*) coding scheme and an exchange crossover operator to create an evolvable genotypic structure that made genetic linkage learning natural and viable for genetic algorithms. This integration of data structure and mechanism led to successful genetic linkage learning, particularly on problems composed of badly scaled building blocks. Interestingly, the nucleation procedure, which refers to the process of building-block formation, was less successful on problems with uniformly scaled building blocks, and this chapter seeks to better understand why this was so and to correct the deficiency by adopting a coding mechanism, *promoters*[†‡], that exists in genetics.

In particular, this chapter focuses on the following topics:

- Investigate the key deficiency which prevents the linkage learning genetic algorithm from effectively separating building blocks;
- Introduce the use of promoters and a modified exchange crossover operator to work

[†]Promoters were previously called *start expression genes* in the original paper.

[‡]Promoters are those regions on the DNA structure from where the transcription process begins. We use the term *promoters* in this work to indicate the starting points on the artificial chromosome from where the interpretation procedure begins. The use of this term is not precisely biologically correct but a metaphor in the algorithm we develop in the study.

with promoters to improve the performance of the linkage learning genetic algorithm on uniformly scaled problems.

This chapter starts with an investigation of the difficulty encountered by the original linkage learning genetic algorithm when it is working on uniformly scaled problems, followed by proposing the use of promoters, which is a coding mechanism existing in genetics, to overcome the difficulty. The experimental results demonstrating the performance improvement on uniformly scaled problems are then presented.

5.1 A Critique of the Original LLGA

The linkage learning genetic algorithm was original, provocative, and interesting. It works quite well on badly scaled problems. Unfortunately, it performs poorly on uniformly scaled problems. To better understand that failure, in this section, we investigate what the linkage learning genetic algorithm is supposed to do and observe how and why it fails.

5.1.1 Test Function

In order to observe the process in which the linkage learning genetic algorithm solves problems, we need to set up a controlled testing environment first. In this chapter, a family of trap functions (Ackley, 1987; Deb & Goldberg, 1993; Deb, Horn, & Goldberg, 1993; Deb & Goldberg, 1994) described in section 4.2 is adopted as subproblems for constructing all test problems because trap functions provide decent linkage structures among variables, and good linkage is necessary for solving problems consisting of traps. Thus, the experimental results can be analyzed and compared to what we expect. In particular, the order-4 trap function,

$$\mathrm{trap}_4(u) = \begin{cases} u & u = 4 \\ 3 - u & \text{otherwise} \end{cases} , \tag{5.1}$$

is used in all of the experiments in this chapter. Moreover, the overall fitness of the test problem is a sum of the fitness values of all subproblems as the construction of uniformly scaled problems described in section 4.2.

5.1.2 What Is the LLGA Supposed to Do?

One of the main motivations behind the linkage learning genetic algorithm was to permit a genetic algorithm to gather genes belonging to different building blocks separately and tightly. Note that there are two things needed here. Not only do individual building blocks need to be tight, they also need to be separate to permit effective building-block exchange via crossover. The analysis of linkage skew and linkage shift provided by Harik and Goldberg (1996) addresses the tight linkage side of the ledger, but it says little or nothing about the evolution of separate building blocks.

In selectionist schemes, sometimes good things happen of their own accord, so we examine a four-building-block problem and see if separate and tight linkage can be evolved. In particular, we use the original linkage learning genetic algorithm with EPE-2 to solve a problem composed of four uniformly scaled order-4 traps described in the previous section. In this experiment, we use the following parameter settings which are arbitrarily chosen and not fine tuned. The population size is 300. The tournament size is 3. The crossover rate is 1.0. There are 800 non-coding elements encoded on the chromosome. Then, we examine typical individuals at generations 0, 50, 100, 150, and 200 shown in Figure 5.1.

As shown in the figure, at generation 0, the genetic material is randomly distributed on the chromosome. The solid line-segments between genes represent consecutive non-coding elements. Later, some genes are getting together to form partial building blocks at generation 50. Longer partial building blocks are formed at around generations 100 and 150, the genes belonging to the same building block also get closer and closer. In the end of the run, the four building blocks are formed correctly and separately on the chromosome.

As we can observe in this particular run, the linkage learning genetic algorithm does

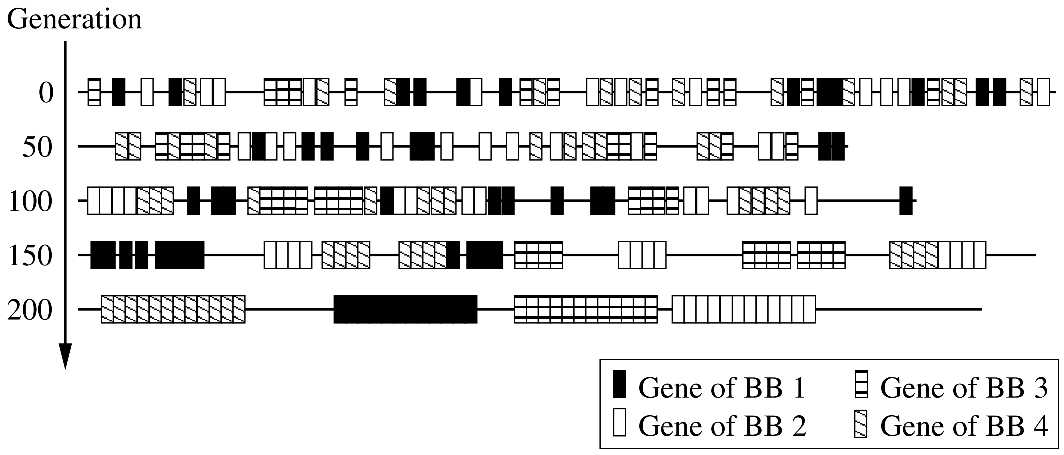


Figure 5.1: In a successful run, the original linkage learning genetic algorithm is able to correctly separate the four building blocks in the problem and make each of them tightly linked on the chromosome (Chen & Goldberg, 2002). The building-block formation process is called *nucleation*, in which building blocks are correctly separated and tightly linked.

construct separate and tight building blocks as desired. We call this building-block formation process *nucleation* that correctly separates building blocks and gets them tightly linked. This might seem to put an end to the matter, but we examine a run that did not go this way.

5.1.3 How Does the LLGA Fail?

The process above yields results along the lines of those hoped for. However, it is not the only result we might get when we use the linkage learning genetic algorithm to solve four uniformly scaled building blocks. Another possible result is that genes of different building blocks get together to form a single, *intertwined building block*, which is composed of genes belonging to several building blocks and getting mixed together.

By examining the results of an unsuccessful run of the identical experiment shown in Figure 5.2, we can easily observe the process of forming an intertwined building block. We call this process *misnucleation*, in which building blocks are not separated correctly and misidentified building blocks are formed.

As in the previous case, the genetic material is initially randomly distributed on the

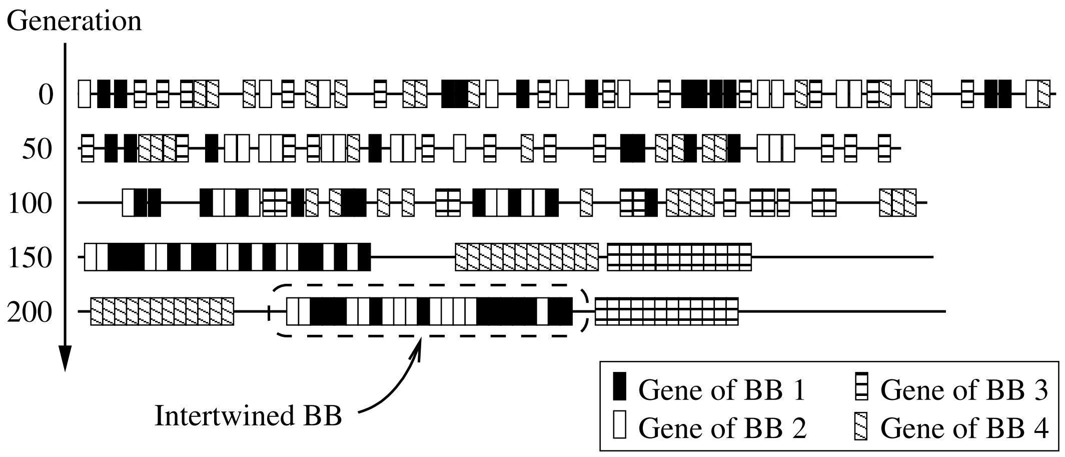


Figure 5.2: In an unsuccessful run, the original linkage learning genetic algorithm cannot correctly separate the four uniformly scaled building blocks. An *intertwined building block* of two actual building blocks is formed during the process, called *misnucleation*, in which building blocks are not separated correctly and misidentified building blocks are formed (Chen & Goldberg, 2002).

chromosome as shown in the figure. At generation 50, partial building blocks are being constructed gradually. At generations 100 and 150, we can easily see that two of the four building blocks tend to intertwine each other. The genes belonging the these two building blocks start to go closer to one another. Finally, only two actual building blocks are correctly identified and separated. The other two building blocks are intertwined as they were a single building block on the chromosome.

In this case, two building blocks formed an intertwined building block along the misnucleation process. The intertwined building block was regarded as a single building block on the chromosome, and the linkage learning genetic algorithm could only make it tight instead of separated. Hence, the real building blocks cannot be identified under such a condition. The intertwined building block prevented the building blocks from being mixed well and effectively and therefore also prevented the problem from being solved.

5.1.4 Separation Inadequacy: Key Deficiency of the LLGA

Based on the results presented in the previous sections and those from a good number of experiments, the original linkage learning genetic algorithm seems only able to deal with a very small number (around 2 to 5) of uniformly scaled building blocks. Misnucleation becomes very likely to happen when the number of building blocks increases much beyond that number because the nature of uniformly scaled problems requires all of the building blocks to be identified and separated at the same time, but the original linkage learning genetic algorithm has no mechanism to do that. Therefore, the original linkage learning genetic algorithm performs poorly on uniformly scaled building blocks.

By revisiting the two linkage learning mechanisms, we find that there are only two categories of the genetic material:

1. the genetic material that affects the solution;
2. the genetic material that does not affect the solution.

Neither the linkage skew nor the linkage shift separates different building blocks and makes each of them tightly linked. What they actually do is make the genetic linkage of the genetic material that improves the solution quality tighter, no matter the genetic material belongs to the same building block or not. The original linkage learning genetic algorithm does not have an appropriate mechanism for appropriate building-block separation.

The same argument also applies to the badly scaled problems. When the linkage learning genetic algorithm solves a badly scaled problem, it deals with the building blocks one by one. Fitness scaling causes the most salient building block to be processed first, then the next most salient, and so on. Therefore, the original linkage learning genetic algorithm can easily handle badly scaled problems or problems consisting of the building blocks of distinguishable salience because under this condition, effectively, the linkage learning genetic algorithm faces only one or a few building blocks at a time.

As shown by the observation in the previous sections and as discussed above, there is no appropriate, specific mechanism built in the linkage learning genetic algorithm to correctly separate building blocks. The reason why the linkage learning genetic algorithm can still handle and separate a few number of uniformly scaled building blocks is that the schema theorem (Holland, 1975; De Jong, 1975; Goldberg, 1989c; Goldberg & Sastry, 2001) delivers marginal differentiable signals for separating building blocks and drives the linkage learning genetic algorithm toward building-block separation.

To see this, according to the schema theorem, schemata (or building blocks) with shorter defining lengths are favored in the evolutionary process and are harder to be disrupted by crossover. For simple genetic algorithms with fixed genetic operators and chromosome representations, this implication of the schema theorem may provide only the description of the dynamics of genetic algorithms without regarding genetic linkage. However, in the linkage learning genetic algorithm, since the genotypic structure is evolvable, this insight from the schema theorem also provides us the explanation of the evolution of genetic linkage. Because shorter building blocks are harder to be disrupted by crossover, the genes belonging to the same building block tend to maintain their relative positions on the chromosome once they are put close to one another. Otherwise, the genes may be randomly distributed again due to the disruption of crossover. Therefore, the schema theorem delivers passive power to the linkage learning genetic algorithm to separate building blocks. The whole building-block formation process can be described as follows:

1. The genes of one building block are distributed at random on the chromosome.
2. If these genes are close to one another by chance, they tend to maintain their relative positions on the chromosome because of a low probability to be disrupted by crossover.
3. Otherwise, if these genes are far from one another, by the schema theorem, they tend to be disrupted by crossover. In the linkage learning genetic algorithm, the effect of disruption of exchange crossover is equivalent to redistributing these genes at random.

According to this process, the schema theorem does not play an active role in building-block formation. It provides only passive, marginal power to separating building blocks. Hence, building blocks are only sometimes formed under this circumstance. Because the power to separate building blocks delivered by the schema theorem is not enough, the original linkage learning genetic algorithm usually cannot correctly separate more than five building blocks at the same time.

5.2 Improve Nucleation Potential with Promoters

After knowing this limit of the original linkage learning genetic algorithm, what we should do now is to improve the nucleation and separation of building blocks. In the original linkage learning genetic algorithm, any point can be a point of interpretation. Such a design provides no advantage for building-block separation at the chromosome-representation level because totally random points of interpretation create the instability of building-block formation. In order to overcome this, we introduce the use of promoters that act as the only possible points of interpretation (Chen & Goldberg, 2002). This should lower the instability of building-block formation and improve nucleation potential during the evolutionary process. We also modify the exchange crossover operator so it can work with promoters. The modified exchange crossover operator uses the point of interpretation of the donor as one of the cutting points to further reduce the randomness of building-block formation. Additionally, in the modified linkage learning genetic algorithm, the probabilistic expression is used instead of the extended probabilistic probabilistic expression. In the remainder of this section, these modifications are discussed in detail. The experimental results show that the modifications indeed make the linkage learning genetic algorithm have better ability to correctly identify and separate uniformly scaled building blocks.

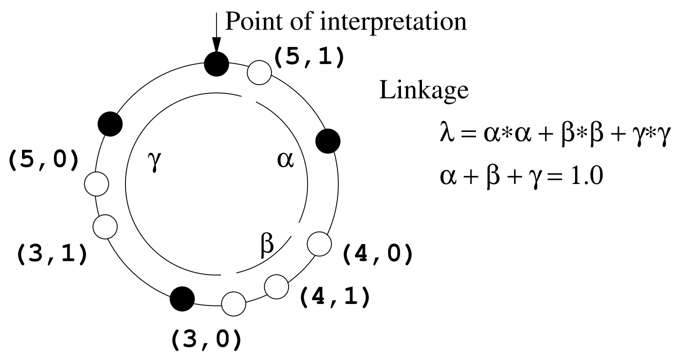


Figure 5.3: Calculation for the genetic linkage of a three-gene building block with promoters on the chromosome. The genetic linkage is calculated according to the expressed genes only.

5.2.1 How Do Promoters Work?

Promoters are special non-coding elements in the chromosome. While in the original linkage learning genetic algorithm, all functional genes and non-coding elements could be chosen as a point of interpretation of the offspring after crossover, in the modified version of the LLGA, only promoters can be the points of interpretation. The way to quantify the genetic linkage is identical to that described in section 3.3.1. Figure 5.3 shows an example to calculate the genetic linkage of a three-gene building block with promoters.

In the original modified linkage learning genetic algorithm, when doing the exchange crossover operator, the grafting point selected on the recipient is the new point of interpretation of the generated offspring. When using promoters in the chromosome representation, only promoters can be chosen as a point of interpretation. That is, the grafting point can still be any functional gene or non-coding element. But in a crossover event, the nearest promoter *before* the grafting point is determined as the point of interpretation of the offspring. Therefore, after randomly choosing the grafting point, we find the the first promoter just before the grafting point and make it the point of interpretation of the child individual. The genetic material between the promoter and the grafting point is first transferred to the child, then the segment from the donor is duplicated, and finally the rest of the recipient gets copied. Figure 5.4 illustrates how promoters work, and the black circles are promoters.

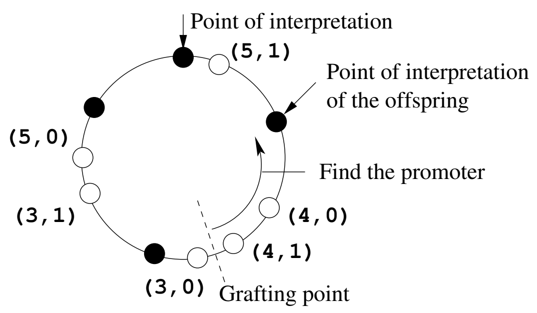


Figure 5.4: After selecting the grafting point on the recipient, the nearest promoter *before* the grafting point is then the point of interpretation of the offspring.

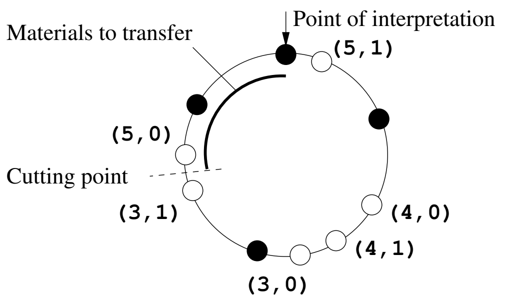


Figure 5.5: After selecting the cutting point on the donor, the genetic material *after* the cutting point and before the current point of interpretation is transferred.

5.2.2 Modified Exchange Crossover

After adding promoters to the chromosome, the exchange crossover operator needs to be modified so it can work with promoters. By modifying the exchange crossover operator, we create a further relationship between this genetic operator and the point of interpretation. The modified exchange crossover operator can reduce randomness of building-block formation and improve nucleation potential.

The original exchange crossover operator cuts a *random* segment from the donor and injects the segment into the recipient. In order to reduce the randomness, the modified exchange crossover operator selects only one cutting point at random. The other cutting point used by the modified exchange crossover operator is always the gene or non-coding element just before the point of interpretation of the donor. At the chromosome-representation level, the operation is like one-point crossover; while at the expressed-string level, the operation is like uniform crossover. Figure 5.5 shows the portion of the genetic material to be transferred.

More specifically, the modified exchange crossover operator works as follows:

1. Select a grafting point at random on the recipient;

2. Determine the point of interpretation of the child as described in section 5.2.1;

3. Copy the genetic material between the current point of interpretation and the grafting

point to the child;

4. Select the cutting point at random on the donor;
5. Graft the genetic material between the cutting point and the point of interpretation of the donor to the child;
6. Copy the rest genetic material of the recipient to the child;
7. Remove the duplicate functional genes and non-coding elements of the child to ensure the validity.

By applying these steps, the exchange crossover operator is modified to work with the LLGA chromosomes that contains promoters.

5.2.3 Effect of the Modifications

In order to observe and verify the effect of the modifications we made in the previous sections, now we use the linkage learning genetic algorithm equipped with promoters to solve the problem composed of four uniformly scaled building blocks as we did in section 5.1. Identical to the experimental settings we used in section 5.1, the population size is 300, the tournament size is 3, the crossover rate is 1.0, and there are 800 non-coding elements encoded on the chromosome. Furthermore, the modified linkage learning genetic algorithm has an extra parameter: the number of promoters, n_s. In the experiment, the number of promoters on each chromosome is set to 20. That is, $n_s = 20$. Figure 5.6 shows that the modified linkage learning genetic algorithm can correctly separate the four uniformly scaled order-4 traps.

In addition to those experiments on four building blocks, we vary the number of building blocks to see if the modified version of linkage learning genetic algorithm can do better to separate uniformly scaled building blocks on larger problems. We still use order-4 traps in this experiment. Except for the number of building blocks, all parameters are identical to the previous experiments in this chapter.

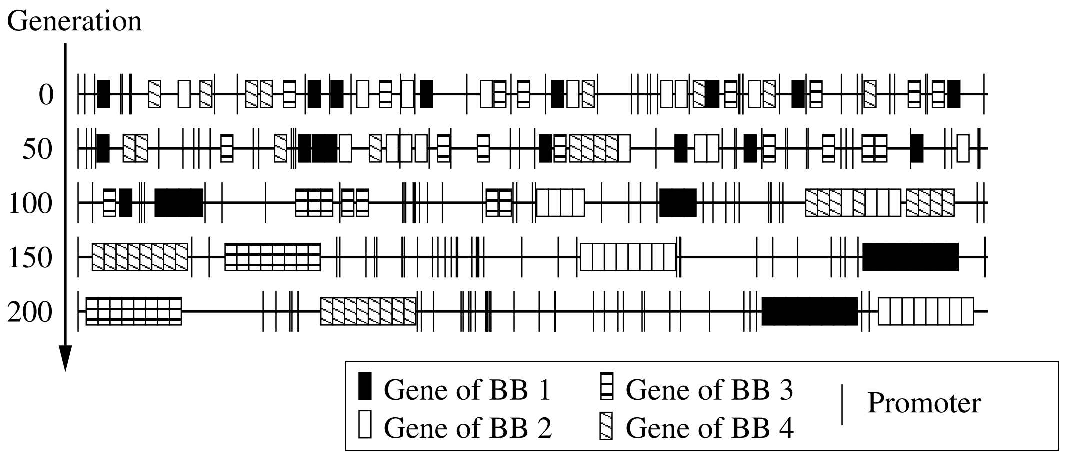
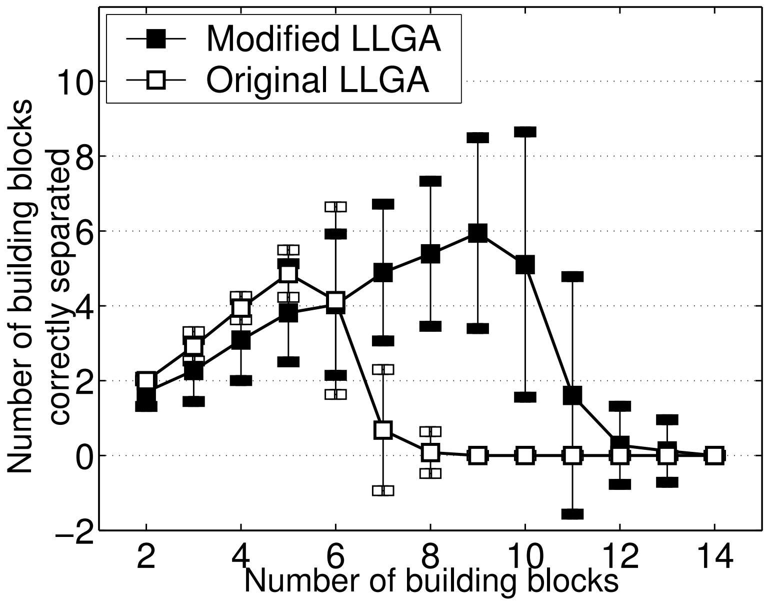


Figure 5.6: Using the modified linkage learning genetic algorithm to solve a problem composed of four uniformly scaled order-4 traps (Chen & Goldberg, 2002).

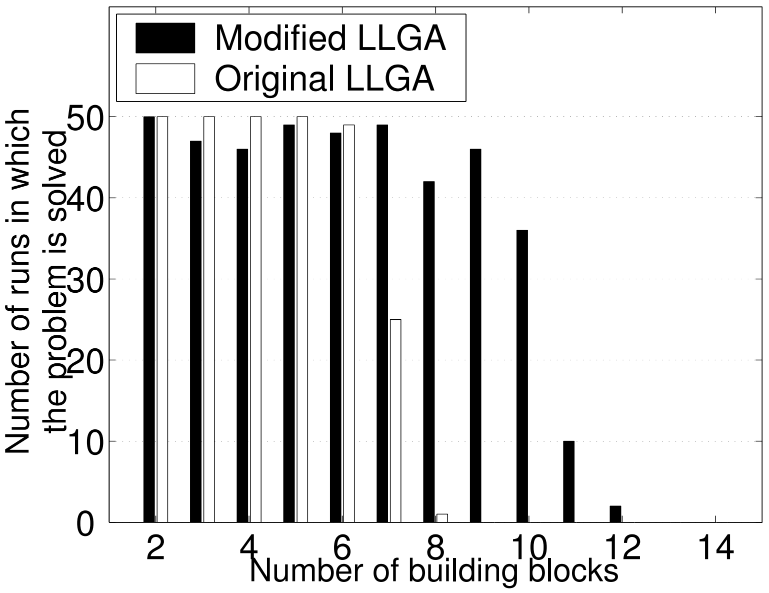
Figure 5.7 shows that the original linkage learning genetic algorithm failed if the number of building blocks is greater than 6. It cannot correctly separate the building blocks and reliably solve the problem. According to these experimental results, equipped with promoters and the modified exchange crossover operator, the linkage learning genetic algorithm is capable of separating and identifying up to eleven building blocks correctly.

5.3 Summary

This chapter introduced the use of promoters in the linkage learning genetic algorithm, which is a coding mechanism exists in biological systems. First, the test function used to study the algorithm was presented. Then, the key deficiency of the linkage learning genetic algorithm, the lack of a building block separation mechanism, was pointed out with example runs. The effect of promoters on the LLGA chromosome and the modification of exchange crossover were described in detail, followed by the experimental results which demonstrated that the use of promoters improved the performance of the linkage learning genetic algorithm on uniformly scaled problems.



(a) The number of building blocks correctly separated at the same time. The results are averaged over 50 independent runs.



(b) The number of runs in which the test problem is solved in the 50 independent runs.

Figure 5.7: Using the linkage learning genetic algorithm equipped with promoters and a modified exchange crossover operator to solve problems consisting of different numbers of uniformly scaled building blocks. As shown in the figures, up to eleven building blocks can be correctly separated and identified (Chen & Goldberg, 2002).

Chapter 6

Convergence Time for the Linkage Learning Genetic Algorithm

As indicated in the previous chapter, inspired by the coding mechanism existing in genetics, introducing the use of promoters in the linkage learning genetic algorithm can reduce randomness of building-block formation and improve nucleation potential. Although adopting promoters enhances the performance of the linkage learning genetic algorithm on uniformly scaled problems, better understanding of the algorithm from a theoretical point of view should be still addressed in order to design a better algorithm. This chapter aims to gain better understanding of the linkage learning genetic algorithm in theory. Particularly, a convergence time model is constructed to explain why the linkage learning genetic algorithm needs exponentially growing computational time to solve uniformly scaled problems.

Therefore, in this chapter, we will develop the convergence time model for the linkage learning genetic algorithm step by step and focus on the following topics:

- Identify a consistent, sequential behavior of the linkage learning genetic algorithm on both exponentially scaled problems and uniformly scaled problems;
- Develop a tightness time model for quantifying the linkage learning time for a single building block based on the two identified linkage learning mechanisms;
- Understand the different situations when the linkage learning genetic algorithm handles a problem of one building block and of multiple building blocks;

- Construct the convergence time model by channeling the model built for one building block into that for multiple building blocks and integrating these results with the identified sequential behavior.

We start with describing the settings of all the experiments for observing the working behavior of the linkage learning genetic algorithm and verifying the theoretical results. Based on the observation, the sequential behavior of the linkage learning genetic algorithm is identified. By extending the two linkage learning mechanisms, linkage skew and linkage shift, the tightness time model for a single building block is proposed, followed by the connection between the sequential behavior and the tightness time model when multiple building blocks exist in the problem. Finally, a convergence time model for the linkage learning genetic algorithm is constructed thereafter by integrating these results and models.

6.1 Experimental Settings

In order to identify the behavior of the linkage learning genetic algorithm, we need to establish the experimental environment in which we can observe how the linkage learning genetic algorithm works and compare the observation to what we expect. In this section, we will describe the test problems and algorithmic parameters in what follows.

First, trap functions (Ackley, 1987; Deb & Goldberg, 1993; Deb, Horn, & Goldberg, 1993; Deb & Goldberg, 1994) are used to construct the test problems in this chapter. As described in section 4.2, trap functions are employed because they provide decent linkage structures, and good linkage is required to solve problems consisting of traps. Specifically, an order-k trap function can be expressed as

$$\mathrm{trap}_k(u) = \begin{cases} u & u = k \\ k - 1 - u & \text{otherwise} \end{cases} . \tag{6.1}$$

For uniformly scaled problems, the fitness is a sum of all the fitness values of each building

Table 6.1: Parameters used to calculate the population sizes based on the gambler's ruin model for population sizing and order-4 trap functions.

Parameter	Value
α	0.1
k	4
σ_{bb}	1.102
d	1.0
d'	0.5253

block. To construct exponentially scaled problems, the scaling factor is 5.0 in this chapter.

Tournament selection without replacement is employed in the algorithm. Although Harik (1997) proposed using a high selection pressure to drive both search and linkage learning processes, according to our recent study (Chen & Goldberg, 2002), the selection pressure required is not necessarily so high. As a consequence, we set the tournament size to 3.

The population size is another essential parameter of genetic algorithms. Using a fixed population size to handle problems of various sizes is inappropriate for the present work because the difficulty of a problem usually increases with the problem size. In this chapter, we employ the gambler's ruin model (Harik, Cantú-Paz, Goldberg, & Miller, 1997; Harik, Cantuú-Paz, Goldberg, & Miller, 1999) described in section 1.3 for population sizing with the signal adjusted by Equation (1.15) for tournament size 3. Specifically, Table 6.1 shows the parameters used in this chapter to calculate the population sizes based on the gambler's ruin model for population sizing, and Table 6.2 shows the population sizes, n, we use in the experiment to handle the problems consisting of different numbers of building blocks, m.

Other parameters are set as follows. The crossover rate is 1.0 such that the crossover event always happens. The maximum number of generations is 10,000. The number of promoters is set to $2m$, where m is the number of building blocks in the problem. The number of non-coding elements is set to $100m$ to maintain a constant disruption probability as soon as a building block is tightly linked. Finally, all results in this chapter are averaged over 50 independent runs for each experiment.

Table 6.2: Population sizes, n, used in the experiment to handle the problems consisting of different numbers of building blocks, m.

m	n	m	n	m	n	m	n
1	N/A	6	154	11	218	16	266
2	70	7	168	12	228	17	276
3	98	8	182	13	238	18	284
4	120	9	194	14	248	19	292
5	138	10	206	15	256	20	300

6.2 Sequentiality for Exponentially Scaled BBs

As indicated in section 3.5, the linkage learning genetic algorithm seems to have an inconsistent behavior when solving the problem consisting of multiple building blocks with different scalings. Therefore, the first step here is to identify a consistent underlying working behavior if it exists. Experiments for exponentially scaled problems and uniformly scaled problems are conducted respectively for observing the working behavior of the linkage learning genetic algorithm. As expected, the empirical results reveal a consistent, sequential behavior. In this section and the following section, the procedures to observe the sequential behavior on exponentially scaled problems and uniformly scaled problems are presented.

We start with the exponentially scaled problem. Harik (1997) identified that the problems composed of exponentially scaled building blocks are solved by the linkage learning genetic algorithm sequentially. These building blocks get tightly linked and solved one by one. In what follows, we conduct our own experiments to verify this sequential behavior of the linkage learning genetic algorithm on exponentially scaled problem.

6.2.1 Time to Convergence

First, we use the linkage learning genetic algorithm to solve exponentially scaled problems composed of different numbers of building blocks. The range of the number of building blocks in the test problems is from 4 to 20. The number of generations is recorded when the

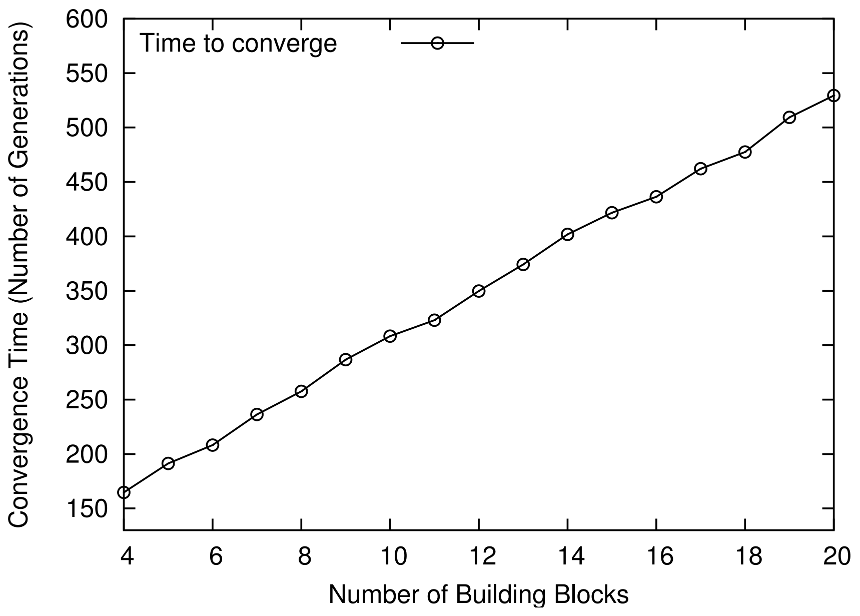


Figure 6.1: Time for the linkage learning genetic algorithm to converge when solving problems composed of exponentially scaled building blocks. The convergence time grows linearly with the number of building blocks. The convergence condition is defined as the difference between the number of building blocks solved in the generational best individual and the average number of building blocks solved in the current population is less than 0.0001 for consecutive 20 generations. The experimental results are averaged over 50 independent runs (Chen & Goldberg, 2004a).

population converges. A population is defined as converged if the following condition is true for consecutive 20 generations:

> The difference between the number of building blocks solved in the generational best individual and the average number of building blocks solved in the current population is less than 0.0001.

The experimental results are shown in Figure 6.1. As we can see, the time for the linkage learning genetic algorithm to converge when solving exponentially scaled problems increases linearly as the number of building blocks increases.

Table 6.3: Experiments for observing the building-block propagation when using the linkage learning genetic algorithm to solve exponentially scaled building blocks. In an experiment, the test problem is composed of totally $m + j$ building blocks in which j building blocks are pre-solved before running the algorithm.

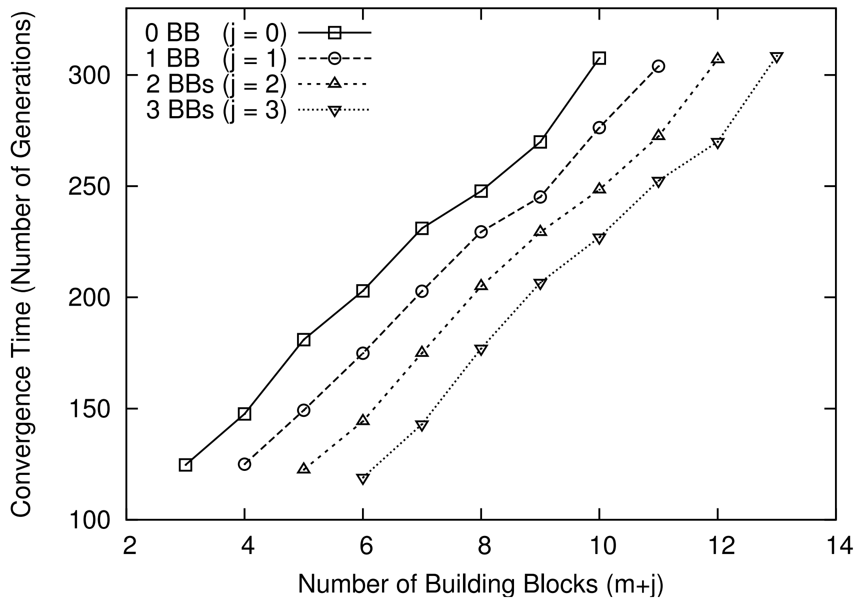
Number of pre-solved BBs (j)	Total number of BBs ($m + j$)
0	3, 4, $\cdots$, 10
1	4, 7, $\cdots$, 11
2	5, 8, $\cdots$, 12
3	6, 9, $\cdots$, 13

6.2.2 Building-Block Propagation

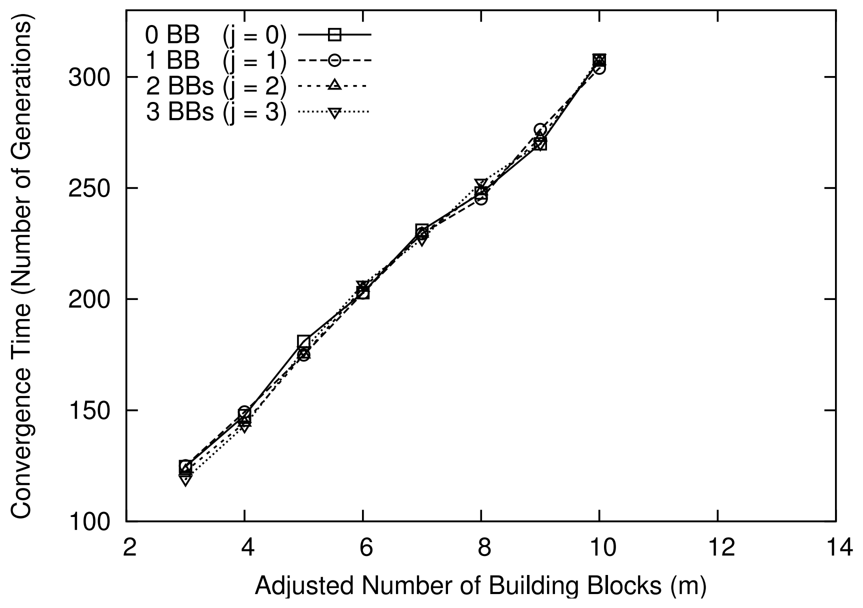
Now we verified that the time for solving exponentially scaled problems grows linearly with the size of the problem as previously reported. For more detailed information about the working behavior, we would like to understand the difference between solving m building blocks from scratch and solving $m+j$ building blocks with j building blocks already solved. In this set of experiments, we solve and tighten several building blocks before running the linkage learning genetic algorithm and observe the convergence time under different settings and configurations. Specifically, Table 6.3 lists all the experiments conducted for this purpose. Figure 6.2 shows the experimental results. It is observed that for convergence time, solving $m + j$ exponentially scaled building blocks with j building blocks pre-solved is equivalent to solving m building blocks from scratch. This means that the solved building blocks are propagated smoothly through the process without being disrupted. Otherwise, solving $m + j$ building blocks with j building blocks pre-solved would take more time than solving m building blocks does.

6.2.3 Time to Tighten the First Building Block

Linear convergence time with proper building-block propagation implies that the time for the first building block to converge should remain constant. Thus, for every g generations, one building block is solved and effectively disappears from the processing scope of the linkage



(a)



(b) Shifted by the number of pre-solved building blocks

Figure 6.2: Convergence time for solving exponentially scaled building blocks with some building blocks pre-solved. It shows that the time required to solve $m + j$ building blocks when j building blocks are solved equals the time needed to solve m building blocks. The experimental results are averaged over 50 independent runs (Chen & Goldberg, 2004a).

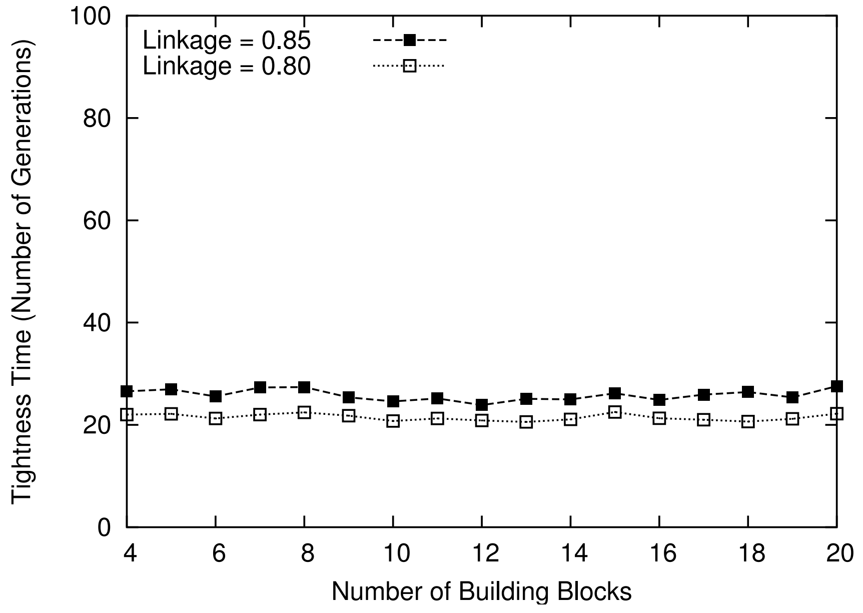


Figure 6.3: Time for the linkage learning genetic algorithm to tighten the first building block holds constant for different numbers of building blocks when solving exponentially scaled problems. The experimental results are averaged over 50 independent runs (Chen & Goldberg, 2004a).

learning genetic algorithm. Note that the "first building block" here does not necessarily mean the building block of the highest number, the most salient building block, or any particular building block. The "first building block" refers to the building block achieving certain genetic linkage first among all building blocks. The results are shown in Figure 6.3. As predicted, the time for the first building block to converge seems to hold constant when solving the problem consisting of different numbers of building blocks.

6.3 Sequentiality for Uniformly Scaled BBs

After observing the behavior of the linkage learning genetic algorithm on exponentially scaled building blocks, we verified the sequentiality reported previously by Harik. In order to establish a consistent behavior for the linkage learning genetic algorithm, in this section, we turn to uniformly scaled building blocks to observe the outcome of similar experiments.

6.3.1 Time to Convergence

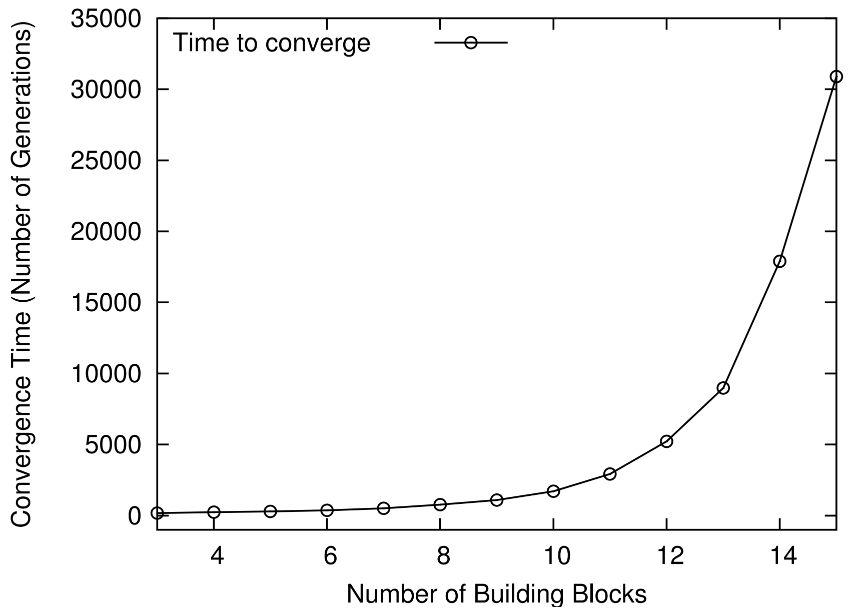
Similar to what we did in section 6.2.1, we use the linkage learning genetic algorithm to solve problems consisting of uniformly scaled building blocks. We vary the number of building blocks from 3 to 15 and record the number of generations when the convergence condition is true for consecutive 20 generations. Figure 6.4 shows the results of the experiments. The convergence time to solve uniformly scaled problems grows exponentially with the number of building blocks. If we consider the overall computational complexity, these results do not contradict those reported by Harik, because the overall complexity grows exponentially.

6.3.2 Building-Block Propagation

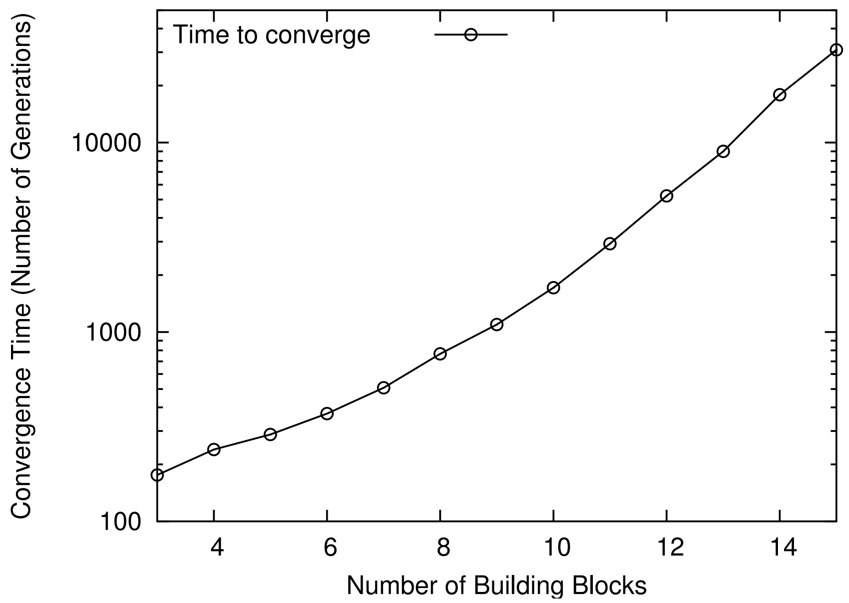
For uniformly scaled problems, we also wish to know if there is any difference between solving m building blocks from scratch and solving $m + j$ building blocks with j building blocks already solved such that a possible consistent working behavior can be established. We use the identical experiments listed in Table 6.3 as we did in section 6.2.2. Figure 6.5 shows the experimental results. It was unexpected that for convergence time, solving $m + j$ uniformly scaled building blocks with j building blocks solved is also equivalent to solving m building blocks from scratch. Because the convergence time grows exponentially, it was expected that building block creation and disruption play a more important role than they do in solving exponentially problems. However, the empirical results of this experiment show that building-block propagation also works well when solving problems consisting of uniformly scaled building blocks.

6.3.3 Time to Tighten the First Building Block

According to the experimental results from the previous experiments, we might expect that for uniformly scaled building blocks, the time for the first building block to converge grows exponentially. Because building-block propagation also works for uniformly scaled problems,

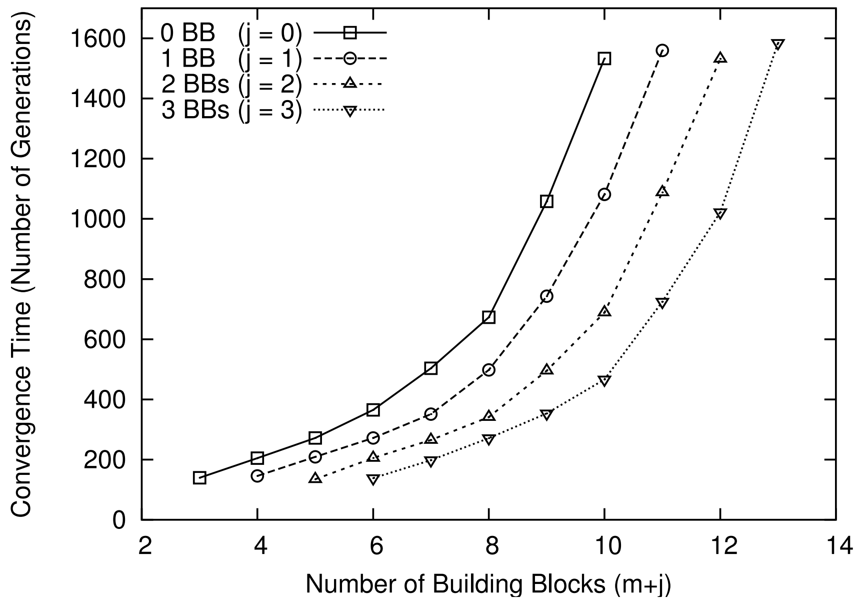


(a)

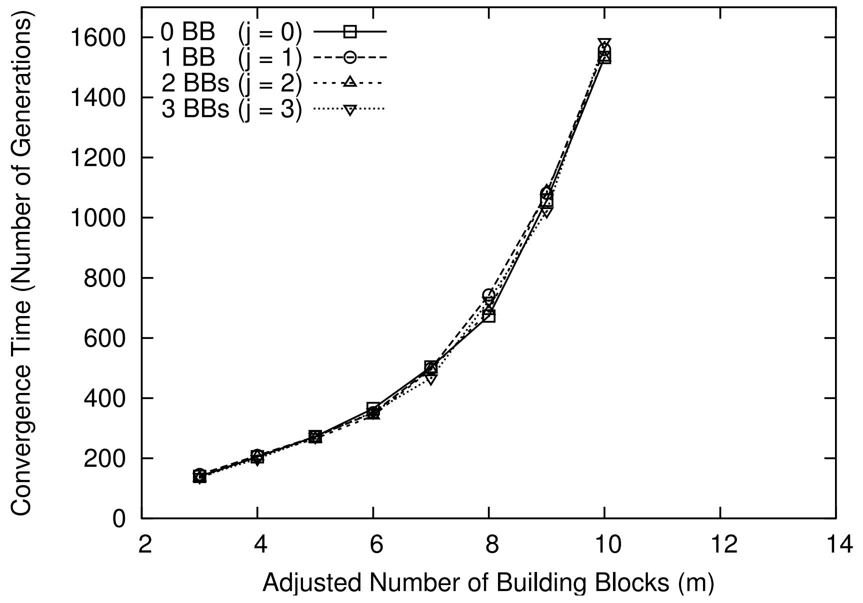


(b) semi-log scale

Figure 6.4: Time for the linkage learning genetic algorithm to converge when solving uniformly scaled building blocks. The convergence time grows greater than exponentially with the number of building blocks. Straight lines on a semi-log scale are indicative of exponential growth. The results are averaged over 50 independent runs (Chen & Goldberg, 2004a).



(a)



(b) Shifted by the number of pre-solved building blocks

Figure 6.5: Convergence time for solving uniformly scaled building blocks with some building blocks pre-solved. It shows that the time required to solve $m+j$ building blocks when j building blocks are solved equals the time needed to solve m building blocks. The experimental results are averaged over 50 independent runs (Chen & Goldberg, 2004a).

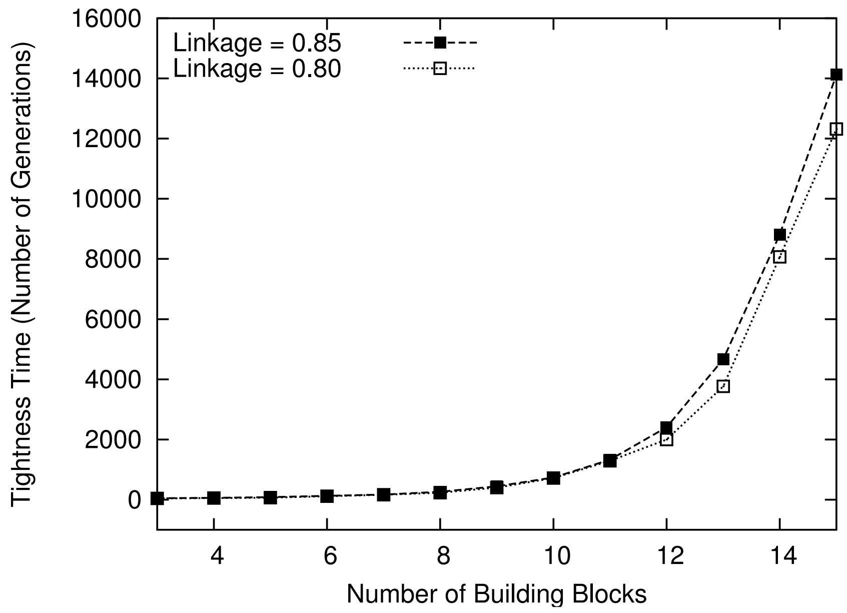
there seems no way for the convergence time to increase exponentially if the time to tighten the first building block does not grow exponentially. Figure 6.6 shows the time for the linkage learning genetic algorithm to tighten the first building block. As predicted, the time for the first building block to converge grows exponentially with the number of building blocks. It implies that the convergence time growth is mainly determined by the time to tighten the first building block in both cases.

These results not only ensure the building-block propagation but also imply that if there are m unsolved building blocks in question, when a building block is solved, the situation or configuration is equivalent to that there are $m - 1$ unsolved building blocks and the whole process restarts. Hence, we propose the *first-building-block model* for describing the sequential behavior of the linkage learning genetic algorithm in the next section.

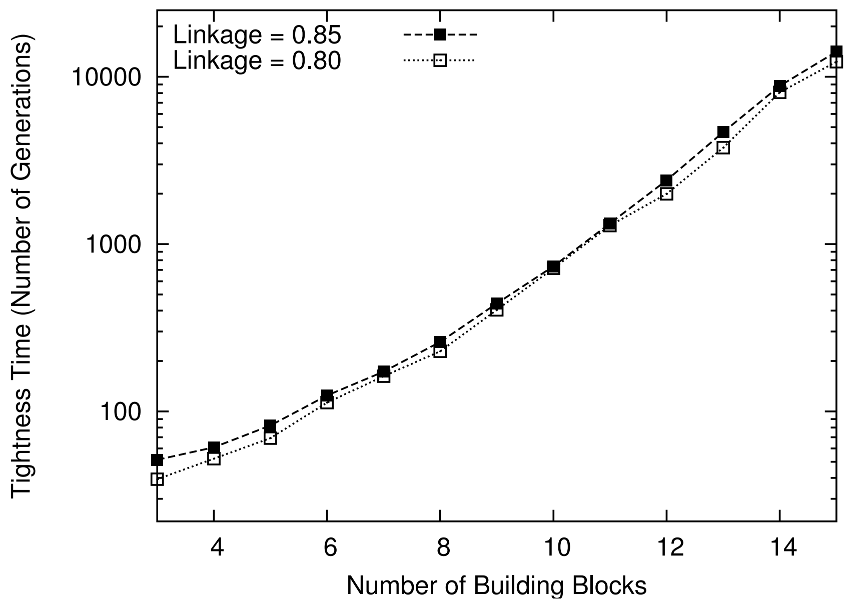
6.4 Macro View: Sequential Behavior

Based on those empirical results obtained in the previous sections, we identified a consistent, sequential behavior of the linkage learning genetic algorithm. In this section, we propose a simple model, called the *first-building-block model* (FBB model) in a top-down manner to describe the sequential behavior of the linkage learning genetic algorithm (Chen & Goldberg, 2004a). The FBB model is proposed from the macro view because this model is established according to the observation made by us as an external observer instead of based on the underlying mechanisms of the algorithm. By assuming that the convergence time is an accumulation of the time to tighten the first building block, we can develop the first-building-block model as follows. First, we define the function t_{fbb} as

$$t_{\mathrm{fbb}}(m) = \text{time to tighten the first building block} \tag{6.2}$$



(a)



(b) semi-log scale

Figure 6.6: Time for the linkage learning genetic algorithm to tighten the first building block grows exponentially with the number of building blocks when solving uniformly scaled problems. The results are averaged over 50 independent runs (Chen & Goldberg, 2004a).

and the function t_c for the convergence time as

$$t_c(m) = \text{convergence time for solving } m \text{ building blocks.} \tag{6.3}$$

By our assumption of the first-building-block model, we can express t_c as

$$t_c(m) = \sum_{i=1}^{m} t_{\text{fbb}}(i) + t_{c0} \ , \tag{6.4}$$

where t_{c0} is a constant. The sequential behavior is therefore established through the first-building-block model. By rewriting t_c as

$$t_c(m) = \sum_{i=i_0}^{m} t_{\text{fbb}}(i) + t_c(i_0 - 1) \ , \tag{6.5}$$

the model can be empirically verified. The experimental results as shown in Figures 6.7 and 6.8 are averaged over 50 independent runs and agree with the proposed model in both cases of the exponentially scaled problem and the uniformly scaled problem.

6.5 Extending Linkage Learning Mechanisms

After identifying the consistent, sequential behavior of the linkage learning genetic algorithm and proposing the FBB model to describe this behavior from the macro view in the previous sections, the next step toward the convergence time model for the whole genetic linkage learning process is to derive the *tightness time*, which is the linkage learning time for a single building block (Chen & Goldberg, 2003b). We first examine the current theoretical analysis of the linkage learning mechanisms (Harik & Goldberg, 1996; Harik, 1997). Models for both linkage learning mechanisms, linkage skew and linkage shift, are refined and extended in this section. The theoretical results are confirmed with experiments. Then, in the next section, a model for *tightness time* is constructed based on the extended models and empirically

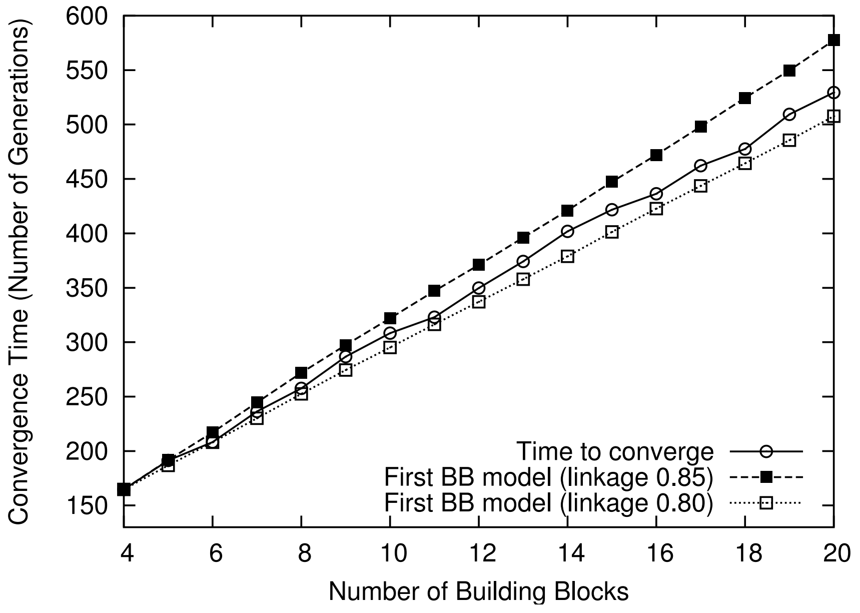
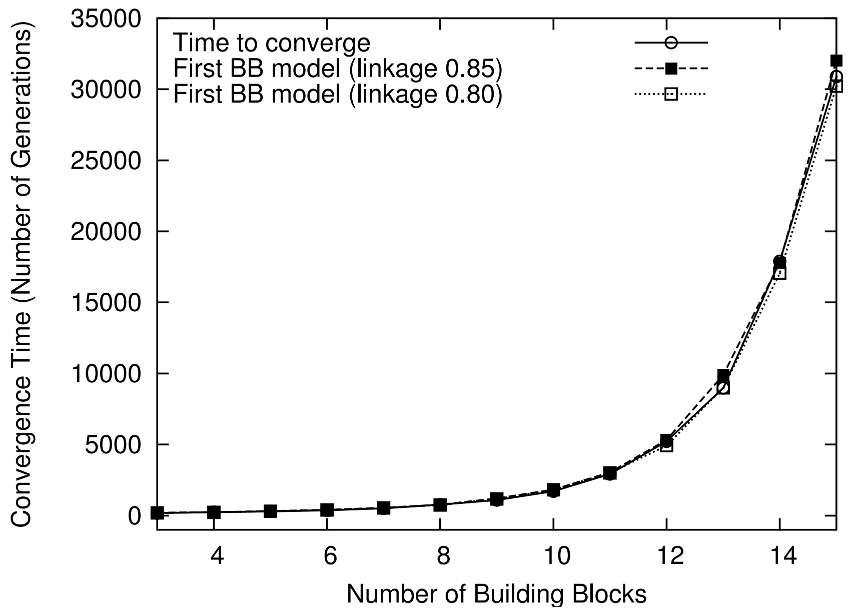


Figure 6.7: The experimental results on problems composed of exponentially scaled building blocks agrees with the first-building-block model described by Equation (6.5). The experimental results are averaged over 50 independent runs (Chen & Goldberg, 2004a).

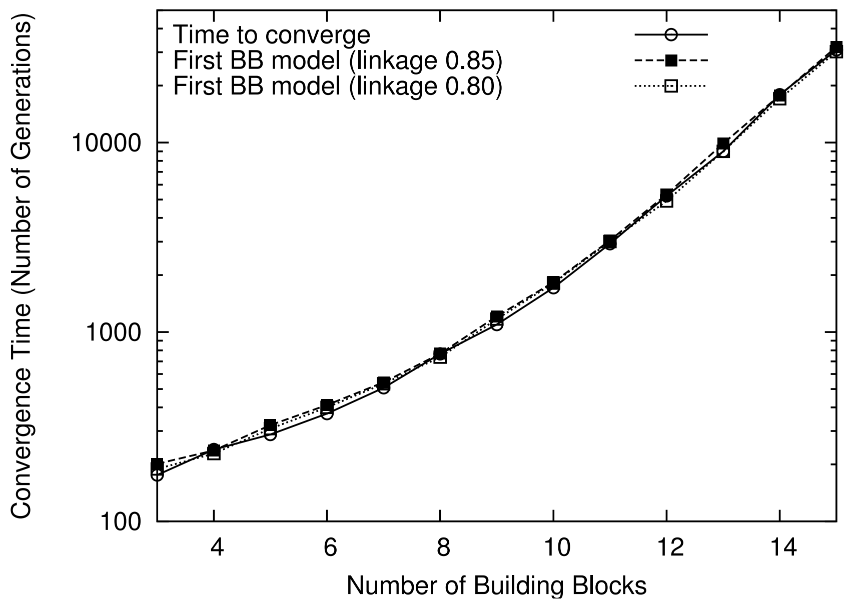
verified. Because artificial evolutionary systems usually create fitness associated with a greedy choice of the best alleles before linkage has evolved, understanding the race between allelic convergence and linkage convergence is critical to designing linkage learning genetic algorithms that work well. Therefore, tightness time is one of the fundamental elements contributing to the theoretical understanding.

6.5.1 Extending the Linkage-Skew Model

As discussed, we first extend linkage skew and then refine linkage shift. Linkage skew, the first linkage learning mechanism, occurs when an optimal building block is successfully transferred from the donor onto the recipient. The conditions for an optimal building block to be transferred are (1) the optimal building block resides in the cut segment and (2) the optimal building block gets expressed before the deceptive one does. The effect of linkage skew was found to modify the whole linkage distribution by eliminating loosely linked building blocks and propagating tight building blocks among individuals. Linkage skew does not make the



(a)



(b) semi-log scale

Figure 6.8: The experimental results on problems composed of uniformly scaled building blocks agrees with the first-building-block model described by Equation (6.5). The experimental results are averaged over 50 independent runs (Chen & Goldberg, 2004a).

genetic linkage of a building block of any single particular individual tighter. Instead, it makes the average of the genetic linkage distribution higher.

Let $\Lambda_t(\lambda)$ be the probability density function of the random variable λ which represents the linkage of an optimal building block at generation t. The following model to describe the evolution of linkage under linkage skew only has been proposed (Harik & Goldberg, 1996):

$$\Lambda_{t+1}(\lambda) = \frac{\lambda \Lambda_t(\lambda)}{\overline{\Lambda_t}} \ . \tag{6.6}$$

Based on Equation (6.6), the genetic linkage average at generation $t+1$ can be calculated as (Harik, 1997):

$$\overline{\Lambda_{t+1}} = \int_0^1 \lambda \Lambda_{t+1}(\lambda)\, d\lambda = \int_0^1 \lambda \frac{\lambda \Lambda_t(\lambda)}{\overline{\Lambda_t}}\, d\lambda = \frac{\overline{\Lambda_t^2}}{\overline{\Lambda_t}} \ , \tag{6.7}$$

and thus,

$$\overline{\Lambda_{t+1}} = \overline{\Lambda_t} + \frac{\sigma^2\left(\Lambda_t\right)}{\overline{\Lambda_t}} \ , \tag{6.8}$$

where $\sigma^2\left(\Lambda_t\right)$ is the variance in the linkage distribution at generation t.

In addition to the average, which is the first moment, of $\Lambda_{t+1}(\lambda)$, we can actually calculate all other moments of $\Lambda_{t+1}(\lambda)$ in the same way:

$$\overline{\Lambda_{t+1}^n} = \int_0^1 \lambda^n \Lambda_{t+1}(\lambda)\, d\lambda = \int_0^1 \lambda^n \frac{\lambda \Lambda_t(\lambda)}{\overline{\Lambda_t}}\, d\lambda = \frac{\overline{\Lambda_t^{n+1}}}{\overline{\Lambda_t}} \ . \tag{6.9}$$

According to the properties of a probability distribution, the moments about the origin completely characterize a probability distribution (Zwillinger & Kokoska, 2000). From Equation (6.9), we can construct all the moments about the origin of $\Lambda_{t+1}(\lambda)$ as follows:

$$\overline{\Lambda_{t+1}^n} = \frac{\overline{\Lambda_t^{n+1}}}{\overline{\Lambda_t}} \qquad n = 1, 2, 3, \cdots \tag{6.10}$$

Hence, the relation between $\Lambda_t(\lambda)$ and $\Lambda_{t+1}(\lambda)$ is established with their moments.

After knowing the relation between $\Lambda_t(\lambda)$ and $\Lambda_{t+1}(\lambda)$, the moment generating function (mgf) defined by the following formula can help us simplify the calculation:

$$m_{\Lambda_t}(s) = E\left[e^{s\Lambda_t}\right] = \int_{-\infty}^{\infty} e^{s\lambda}\Lambda_t(\lambda)\,d\lambda\ . \tag{6.11}$$

Assume that the moment generating function of $\Lambda_t(\lambda)$ exists, it can be written as

$$\begin{aligned} m_{\Lambda_t}(s) &= E\left[e^{s\Lambda_t}\right] \\ &= E\left[1 + \Lambda_t s + \frac{(\Lambda_t s)^2}{2!} + \frac{(\Lambda_t s)^3}{3!} + \cdots\right] \\ &= 1 + \overline{\Lambda_t} s + \overline{\Lambda_t^2}\frac{s^2}{2!} + \overline{\Lambda_t^3}\frac{s^3}{3!} + \cdots \end{aligned} \tag{6.12}$$

The r^{th} moment of $\Lambda_t(\lambda)$ can be obtained with

$$\overline{\Lambda_t^r} = m_{\Lambda_t}^{(r)}(0) = \left.\frac{d^r m_{\Lambda_t}(s)}{ds^r}\right|_{s=0}\ . \tag{6.13}$$

Given the relation between $\Lambda_t(\lambda)$ and $\Lambda_{t+1}(\lambda)$ and the property of the moment generating function, we can now get the moment generating function of $\Lambda_{t+1}(\lambda)$ as

$$m_{\Lambda_{t+1}}(s) = m'_{\Lambda_t}\left(\frac{s}{m'_{\Lambda_t}(0)}\right)\ . \tag{6.14}$$

Therefore, we have a model to calculate $\Lambda_{t+1}(\lambda)$ from $\Lambda_t(\lambda)$ when the moment generating function of $\Lambda_t(\lambda)$ is available.

Furthermore, also based on Equation (6.10), we can obtain the following result:

$$\begin{aligned} \overline{\Lambda_t^n} &= \frac{\overline{\Lambda_{t-1}^{n+1}}}{\overline{\Lambda_{t-1}}} = \frac{\overline{\Lambda_{t-2}^{n+2}}/\overline{\Lambda_{t-2}}}{\overline{\Lambda_{t-2}^2}/\overline{\Lambda_{t-2}}} \\ &= \frac{\overline{\Lambda_{t-2}^{n+2}}}{\overline{\Lambda_{t-2}}} = \cdots \\ &= \frac{\overline{\Lambda_0^{t+n}}}{\overline{\Lambda_0}}\ , \end{aligned} \tag{6.15}$$

which clearly indicates that under linkage skew, any moment of the linkage distribution at any given generation can be predicted with the information of the initial linkage distribution. The linkage learning process is solely determined by the initial distribution if there is only linkage skew working in the process.

Based on its property, linkage skew does not really tighten building blocks in any individual. It drives the genetic linkage distribution to a higher state by propagating tight building blocks among individuals. Therefore, the genetic linkage cannot exceed the maximum linkage in the initial population. The evolution of linkage described by Equation (6.14) is bounded by the initial maximum linkage.

The extended linkage-skew model was empirically verified with the problems consisting of order-4 and order-6 trap functions. We follow an experimental procedure identical to that presented by Harik and Goldberg (1996). In the initial population, all of the individuals contain the optimal building block with random linkages. At every generation, all individuals go through the exchange crossover operator and are crossed with artificially generated individuals containing deceptive building blocks with random linkages for simulating the condition defined for linkage skew. In crossover events, the individuals containing the optimal building block are donors, and the generated individuals containing the deceptive building block are recipients. After crossover, only the offspring has the optimal building block survives and gets a copy in the next generation. The purpose for such experimental procedure is to observe the evolution of linkage solely under linkage skew.

In order to simulate infinite-length chromosomes, we let the functional genes occupy only one percent of the chromosome. That is, for the order-4 building block, the 4 genes are embedded in a 400-gene chromosome with 396 non-coding elements; for the order-6 building block, the 6 genes are embedded in a 600-gene chromosome with 594 non-coding elements. The population size is 5,000 in both cases of order-4 and order-6 traps. All experiments are repeated 50 times and the results are averaged over these 50 independent runs.

For the illustration purpose, we show only the prediction of the average, which is the first moment, and the variance, which is the second moment minus the square of the average, in figures, although the extended linkage-skew model can predict all moments of the linkage distribution at any given generation. Figures 6.9 and 6.10 show the experimental results compared to the theoretical prediction. The theoretical prediction was made based on Equation (6.15). With Equation (6.15), we predict the growth of the average linkage. However, as discussed previously, since linkage skew does not actually change the linkage of building blocks in any individual, the linkage cannot exceed the maximum linkage existing the initial population and is therefore bounded by the maximum initial linkage. Figures 6.9(a) and 6.10(a) specifically show the results that we expected. The growth of the linkage is a patchquilt of the theoretical prediction and upper bound. Therefore, as shown in the figures, the experimental results agree with the prediction of our facetwise model pretty well, and the extended linkage-skew model is experimentally confirmed.

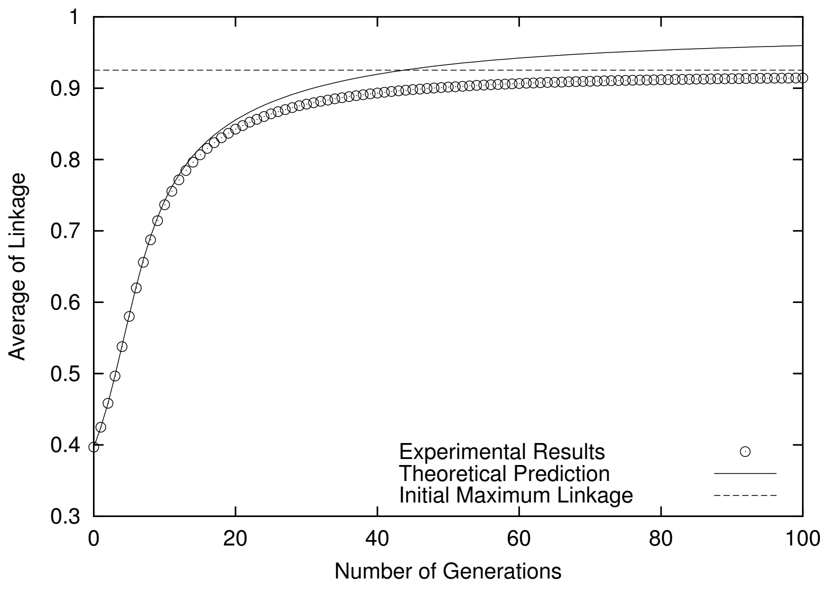
6.5.2 Extending the Linkage-Shift Model

After extending linkage skew, the next step is to refine linkage shift. Linkage shift is the second linkage learning mechanism (Harik & Goldberg, 1996). It occurs when an optimal building block resides in the recipient and survives a crossover event. For an optimal building block to survive, there cannot be any gene contributing to a deceptive building block transferred. Linkage shift gets the genetic linkage of a building block in an individual higher with deletion of duplicate genetic material caused by injection of exchange crossover. Compared to linkage skew, linkage shift gets the linkage of building blocks in each individual higher.

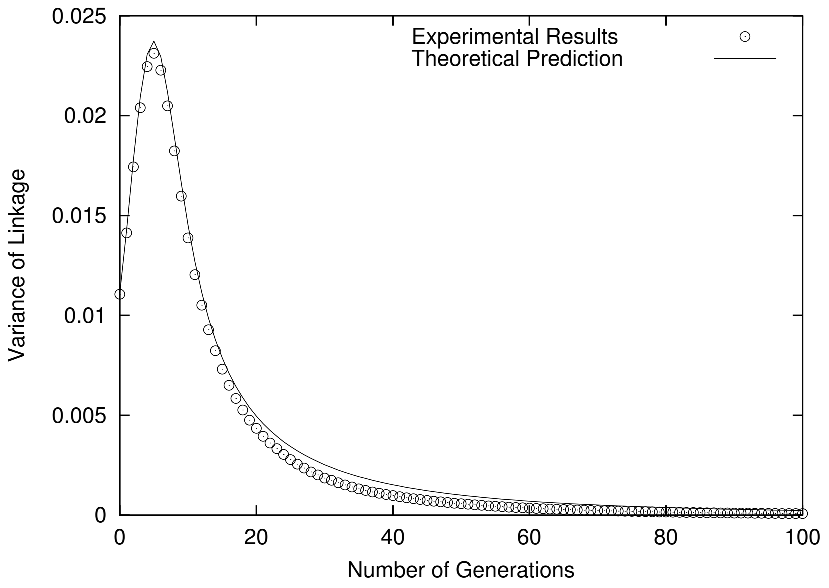
For linkage shift, the following recurrence equation was used (Harik & Goldberg, 1996) to depict the effect of the second mechanism on building blocks that survive crossover events:

$$\overline{\lambda_0(t+1)} = \lambda_0(t) + (1 - \lambda_0(t))\frac{2}{(k+2)(k+3)} \ , \tag{6.16}$$

for an order-k building block. Tracking only the average of genetic linkage, we can approxi-

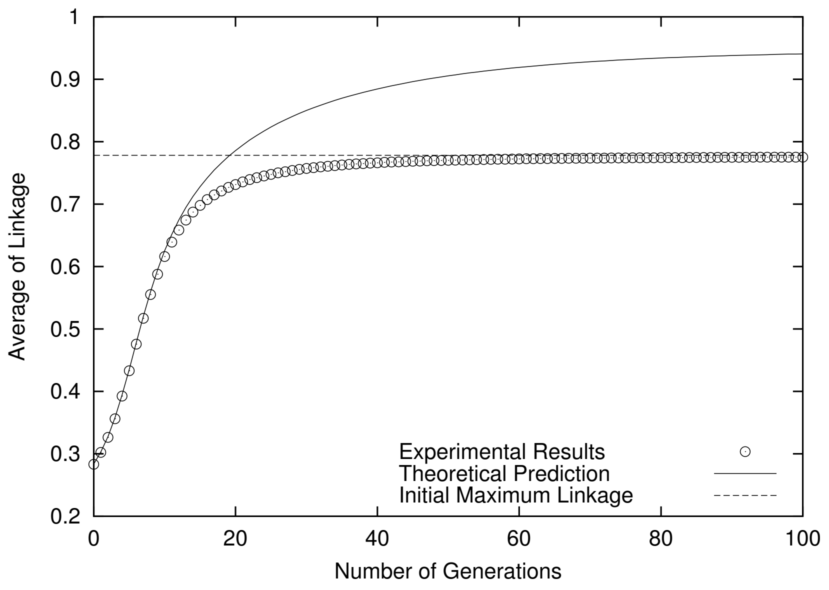


(a) Average of genetic linkage.

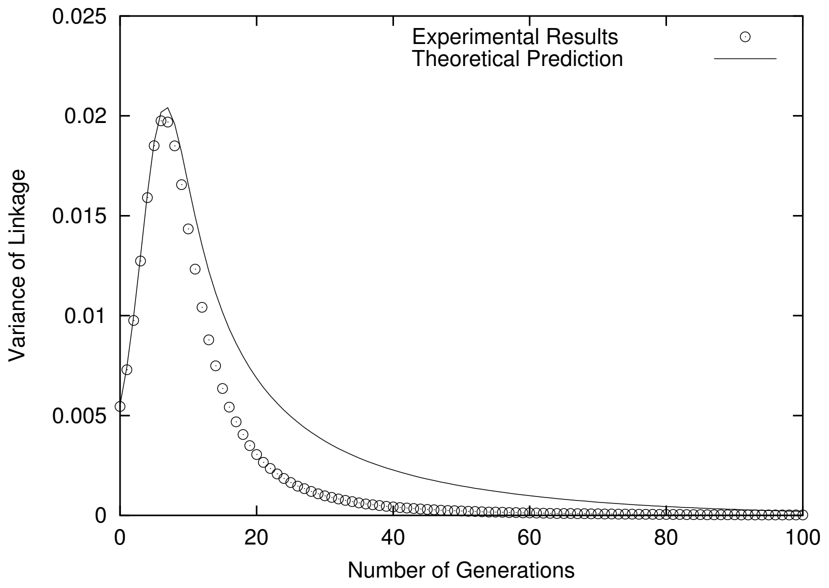


(b) Variance of genetic linkage.

Figure 6.9: Linkage skew on an order-4 trap building block. As expected, under only linkage skew, the experimental results agree with the patch-quilt of the extended linkage-skew model, Equation (6.15), and the upper bound, the maximum linkage. The constructed facetwise model can predict the evolution of genetic linkage for an order-4 trap building block. The experimental results are averaged over 50 independent runs (Chen & Goldberg, 2003b).



(a) Average of genetic linkage.



(b) Variance of genetic linkage.

Figure 6.10: Linkage skew on an order-6 trap building block. As expected, under only linkage skew, the experimental results agree with the patch-quilt of the extended linkage-skew model, Equation (6.15), and the upper bound, the maximum linkage. The constructed facetwise model can predict the evolution of genetic linkage for an order-6 trap building block. The experimental results are averaged over 50 independent runs (Chen & Goldberg, 2003b).

mately rewrite Equation (6.16) as

$$\overline{\Lambda_{t+1}} = \overline{\Lambda_t} + (1 - \overline{\Lambda_t})\frac{2}{(k+2)(k+3)} \ . \tag{6.17}$$

Given a fixed k, let $c = \frac{2}{(k+2)(k+3)}$, we can get the following recurrence relation:

$$\begin{aligned} \overline{\Lambda_{t+1}} &= \overline{\Lambda_t} + c(1 - \overline{\Lambda_t}) \\ &= \overline{\Lambda_t} + c - c\overline{\Lambda_t} \\ &= \overline{\Lambda_t}\,(1 - c) + c \ . \end{aligned} \tag{6.18}$$

By solving the recurrence relation, the new linkage-shift model is obtained as

$$\overline{\Lambda_t} = 1 - (1 - \overline{\Lambda_0})(1 - c)^t \ . \tag{6.19}$$

Therefore, the rate of linkage learning is mainly determined by the genetic linkage average of the initial linkage distribution. Moreover, the higher order the building block is, the longer it takes to evolve to some specific level of genetic linkage.

As for linkage skew, the extended linkage-shift model is also empirically verified. The experimental procedure that we used to verify the extended linkage-skew model is repeated to verify the extended linkage-shift model. The only difference is that in crossover events, the individuals containing the optimal building block are recipients, and the generated individuals containing the deceptive building block are donors. Moreover, the experiment settings are also identical to that used to test the extended linkage-skew model. For linkage shift, we predict the average of genetic linkage on both order-4 and order-6 traps. Figure 6.11 shows the experimental results. The theoretical prediction was made according to Equation (6.19). We also employ the adjustment scheme proposed by Harik and Goldberg (1996) to reflect the difference between the infinite-length chromosome model and the real LLGA chromosomes in experiments. In this study, the maximum possible genetic linkage in both cases is $(0.99)^2 = 0.9801$, and the extended linkage-shift model, Equation (6.19), is adjusted accordingly. As indicated in Figure 6.11, the experimental results agree with the adjusted theoretical prediction, and we are now given a good reason to believe that the extended model is accurate.

6.6 Micro View: Tightness Time

Equipped with the extended models for genetic linkage learning, we are now ready to develop the tightness time model in a bottom-up manner to describe the linkage learning time needed by a single building block. The tightness time model is proposed from the micro view because this model is constructed based on the underlying linkage learning mechanisms. Based on the observation and intuition, the working relationship between linkage skew and linkage shift is as follows. Linkage shift is responsible for making the genetic linkage of a building block in each individual tighter; linkage skew is responsible for propagating the tight building blocks all over the population. Considering linkage shift as a refiner, linkage skew as a propagator, and that the effect of linkage skew comes pretty fast based on the experimental results, the genetic linkage learning bottleneck is in fact linkage shift. Hence, we start to develop the model for tightness time based on the most critical component in the framework first.

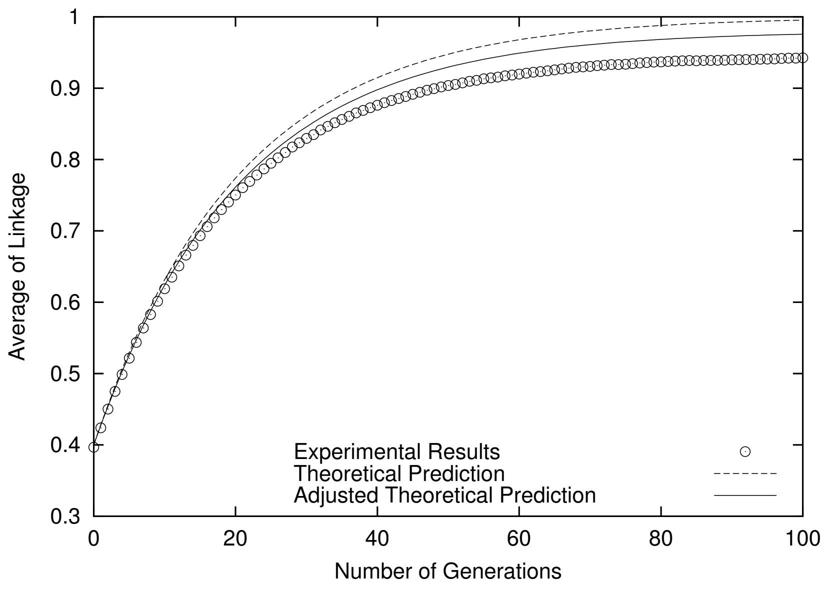
Start from Equation (6.19), we can obtain

$$t = \frac{\log(1 - \overline{\Lambda_t}) - \log(1 - \overline{\Lambda_0})}{\log(1 - c)} \,. \tag{6.20}$$

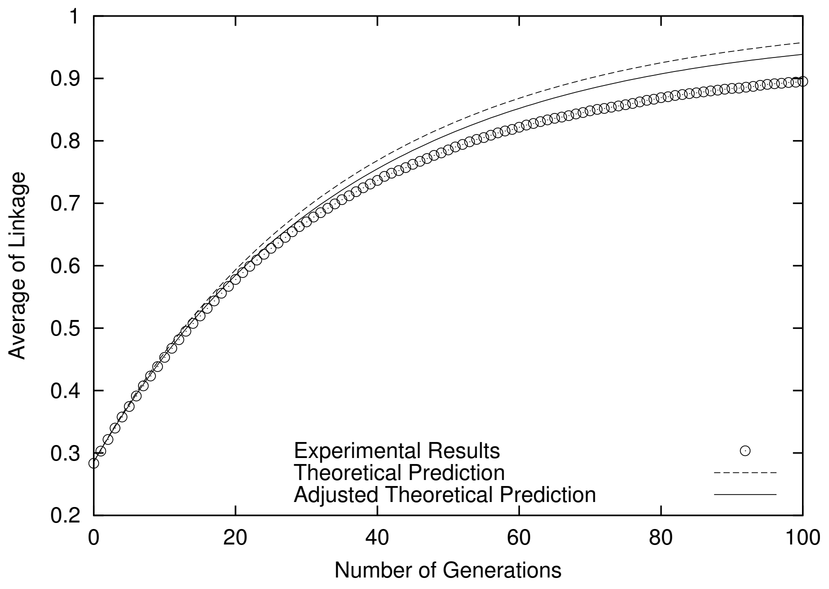
Then, it can be rewritten as a function of linkage λ:

$$t(\lambda) = \frac{\log(1 - \lambda) - \log(1 - \overline{\Lambda_0})}{\log(1 - c)} \,. \tag{6.21}$$

By taking the propagation effect of linkage skew into account, a constant c_s standing for the linkage learning speed-up caused by linkage skew is added into the model. Thus, we obtain



(a) Order-4 trap building block.



(b) Order-6 trap building block.

Figure 6.11: Linkage shift on an order-4 trap and an order-6 trap. The experimental results shows that under only linkage shift, the extended linkage-shift model, Equation (6.19), can predict the evolution of genetic linkage for both order-4 and order-6 traps. The experimental results are averaged over 50 independent runs (Chen & Goldberg, 2003b).

the tightness time model as follows:

$$t_\ell(\lambda) = \frac{\log(1-\lambda) - \log(1-\overline{\Lambda_0})}{c_s \log(1-c)} \ , \tag{6.22}$$

where $t_\ell(\lambda)$ is the tightness time for a given genetic linkage λ, and $c_s \approx 2$ is a constant which is determined empirically.

Furthermore, given the initial genetic linkage distribution, $\overline{\Lambda_0}$ remains constant during the whole evolutionary process. For simplicity, we can define

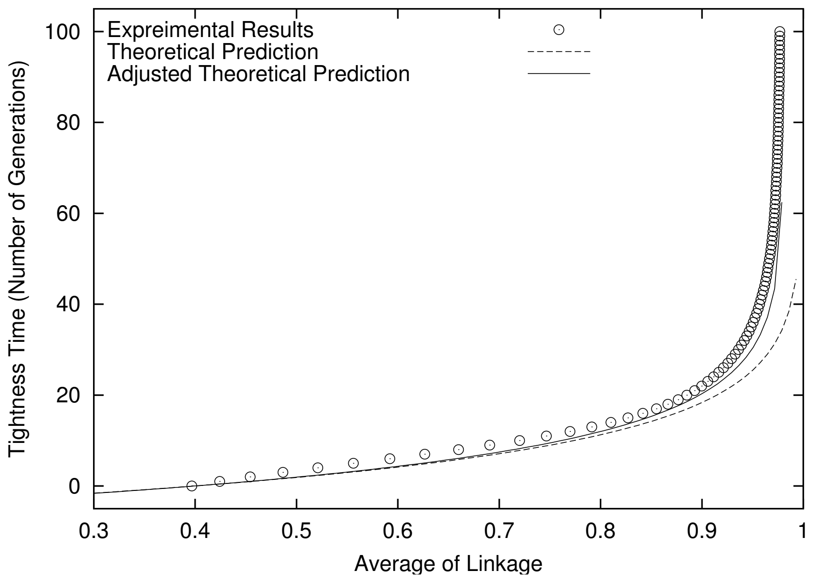
$$\epsilon = 1 - \lambda \ ,$$
$$\overline{\epsilon_0} = 1 - \overline{\Lambda_0} \ .$$

Also, $c = \frac{2}{(k+2)(k+3)} \approx \frac{2}{k^2}$ when $k \to \infty$. Therefore, Equation (6.22) can be rewritten as a function of ϵ as

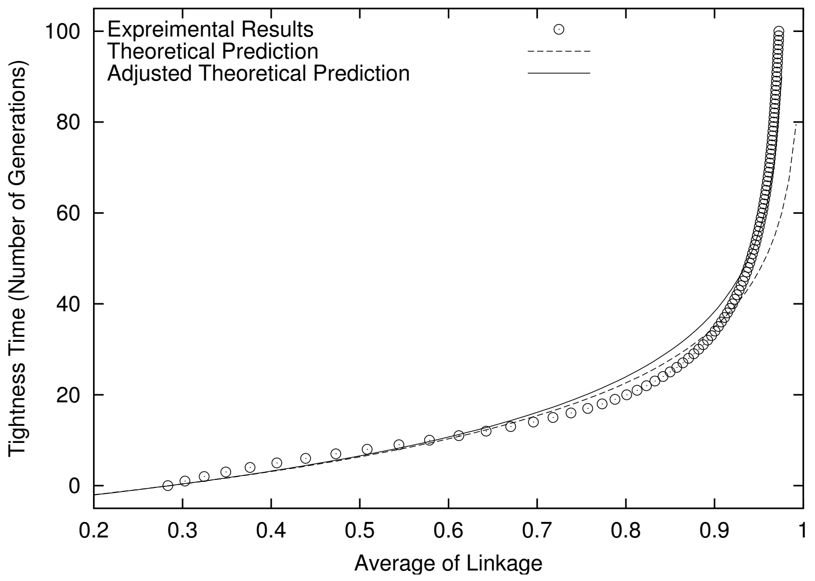
$$t'_\ell(\epsilon) = \frac{k^2}{2c_s} \log \frac{\epsilon}{\overline{\epsilon_0}} \ . \tag{6.23}$$

Equation (6.23) shows that tightness time is proportional to the square of the order of building blocks. The longer the building block, the much longer the tightness time. Moreover, tightness time is proportional to the logarithm of the desired linkage.

Experiments were also performed to verify the model for tightness time. In order to verify the tightness time model, we also employ the identical experimental procedure as we verified the extended linkage-skew and linkage-shift models. Since we are interested in the combined effect of linkage skew and linkage shift, in the crossover events, the individual containing the optimal building block from the population is a donor with a probability of 0.5, and after determining the role of the individual, the role of the generated individual containing the deceptive building block is decided accordingly. Using the same parameter settings as we used to verify linkage skew and linkage shift, both genetic linkage learning mechanisms work together in the experiments as described. Figure 6.12 shows the experi-



(a) Order-4 trap building block.



(b) Order-4 trap building block.

Figure 6.12: Tightness time for an order-4 and an order-6 traps. The experimental results shows that under both linkage skew and linkage shift, the tightness time model, Equation (6.22), can predict the evolution of genetic linkage for both order-4 and order-6 traps. The experimental results are averaged over 50 independent runs (Chen & Goldberg, 2003b).

mental results. The theoretical prediction made based on Equation (6.22) is also adjusted with the maximum possible genetic linkage 0.9801 according to the chromosome encoding in the experiments. The obtained numerical data agree with our tightness time model quite well, and our hypothesis and tightness time model are therefore empirically verified.

6.7 From One Building Block to m Building Blocks

After identifying the sequential behavior of the linkage learning genetic algorithm in a top-down manner and developing the tightness time model for a single building block based on the linkage learning mechanisms in a bottom-up manner, the missing link here is to understand how multiple building blocks affect the tightness time for the first building block when the building blocks are uniformly scaled in the problem. Therefore, we will now identify the effect of multiple building blocks on the tightness time for a single building block to learn genetic linkage in this section. Then, in the following section, we will construct the convergence time model for the linkage learning genetic algorithm by integrating the models from the macro view, the micro view, and the connection in between.

Because the tightness time model assumes (1) a single building block and (2) all events are useful and effective for genetic linkage learning, when dealing with m uniformly scaled building blocks, we need to take into consideration the probability of a linkage learning event. First of all, we define a linkage learning event as either a linkage-skew event or a linkage-shift event. Note that when we analyze the tightness time, the probability of a linkage learning event, $p_{\ell\ell}(1)$, is assumed to be 1.0 for $m = 1$. When handling uniformly scaled building blocks, $p_{\ell\ell}(m)$ will be lower than 1.0 due to the interaction among the building blocks of equal importance. Since we are interested in the dimensional model of convergence time, the following analysis assumes the middle stage of the genetic linkage learning process.

6.7.1 Genetic Material from the Donor

First, we consider the genetic material from the donor. When exchange crossover operates, a donor and a recipient are selected from the current population. Based on the linkage learning mechanisms, assume that (1) a segment from the donor containing only one complete building block (and probably other incomplete building blocks) contributes genetic linkage learning and (2) the m building blocks are uniformly distributed on the chromosome in the middle stage of the genetic linkage learning process. On average, there are $m/2$ building blocks transferred from the donor to the recipient. The number of possible conditions are

$$\binom{m}{\frac{m}{2}} . \tag{6.24}$$

Therefore, the probability for the segment coming from the donor to contain only one complete building block is

$$\binom{m}{1} \Big/ \binom{m}{\frac{m}{2}} . \tag{6.25}$$

Note that the above analysis is inspired by that proposed by Thierens (1995) for analyzing the genetic algorithm.

6.7.2 Genetic Material on the Recipient

In addition to the genetic material coming from the donor, $p_{\ell\ell}(m)$ also depends on the genetic material residing on the recipient. If the building block in question is disrupted by the grafting point, there will be no linkage learning event. Base on the calculation of random linkage (Harik, 1997), the probability for a building block of order k to reside on one of the two segments is described by Equation (3.3). If the building block is in the segment before the grafting point, a linkage-shift event occurs. If the building block is in the segment after the grafting point, a linkage-skew event happens.

6.7.3 Tightness Time for m Uniformly Scaled Building Blocks

After combining the results obtained from the above analysis and discussion, the probability of a genetic linkage learning event can be expressed as the following equation:

$$p_{\ell\ell}(m) = \frac{2}{k+1}\left[\binom{m}{1} \Big/ \binom{m}{\frac{m}{2}}\right] . \tag{6.26}$$

By integrating the tightness time model for a single building block and the probability of genetic linkage learning events, we obtain the tightness time model for m uniformly scaled building blocks as

$$\begin{aligned} t(m,\epsilon) &= t_\ell(\epsilon)\frac{1}{p_{\ell\ell}(m)} ; \\ &= t_\ell(\epsilon)\frac{k+1}{2}\left[\binom{m}{\frac{m}{2}} \Big/ \binom{m}{1}\right] . \end{aligned} \tag{6.27}$$

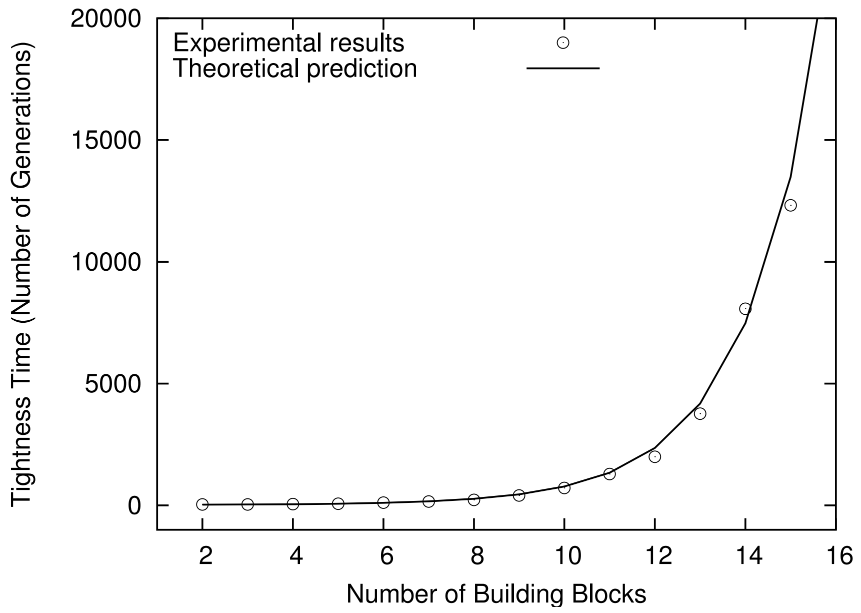
Applying the Stirling approximation

$$m! \approx \left(\frac{m}{e}\right)^m \sqrt{2\pi m} , \tag{6.28}$$

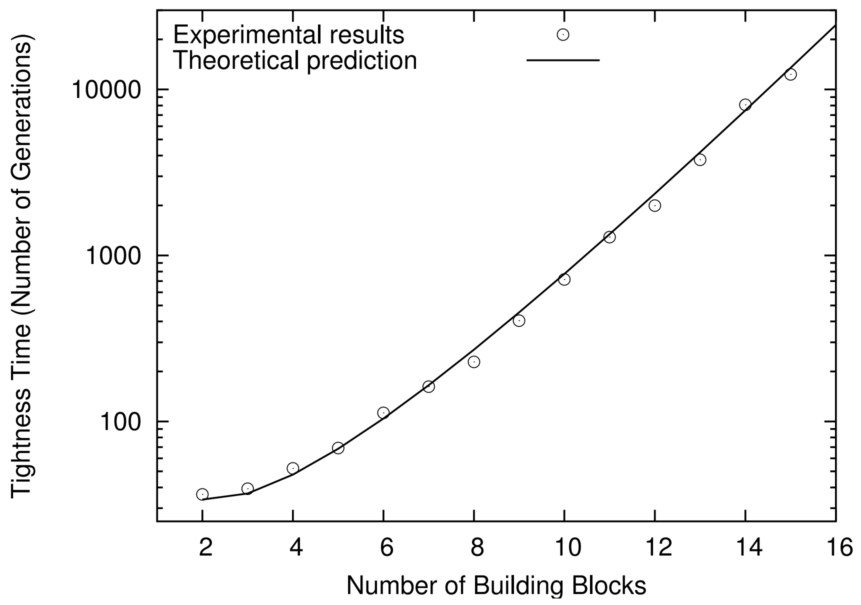
we obtain

$$\begin{aligned} t(m,\epsilon) &= t_\ell(\epsilon)\left(\frac{k+1}{2}\right)\frac{\sqrt{2}}{\sqrt{\pi}}\frac{2^m}{m\sqrt{m}} ; \\ &= t_\ell(\epsilon)\,\frac{k+1}{\sqrt{2\pi}}\,\frac{2^m}{m\sqrt{m}} . \end{aligned} \tag{6.29}$$

Figure 6.13 shows that the experimental results agree with the tightness time model for m uniformly scaled building blocks. The figure shows the trend of the number of generations needed for the genetic linkage λ of the first building block among m uniformly scaled building block to achieve 0.80 during the evolutionary process.



(a)



(b) semi-log scale

Figure 6.13: Time for the genetic linkage λ of the first building block among m uniformly scaled building block to achieve 0.80 during the evolutionary process. The experimental results are averaged over 50 independent runs (Chen & Goldberg, 2004a).

6.8 Convergence Time Model for the LLGA

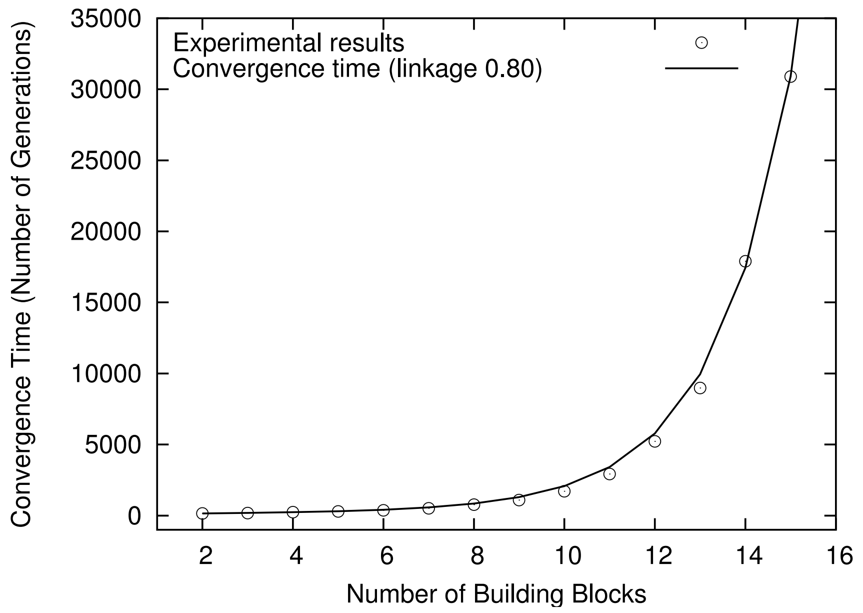
Finally, by integrating the results from the sequential behavior (Equation (6.4)), the tightness time model (Equation (6.23)), and the connection in between (Equation (6.29)), we can obtain the linkage learning genetic algorithm convergence time model for some desired genetic linkage as follows:

$$\begin{aligned} t_{\mathrm{C}}(m,\epsilon) &= \sum_{i=1}^{m} t(i,\epsilon) + t_{\mathrm{C}0} \ ; \\ &= \sum_{i=1}^{m} \left(t_\ell(\epsilon) \, \frac{k+1}{\sqrt{2\pi}} \, \frac{2^i}{i\sqrt{i}} \right) + t_{\mathrm{C}0} \ ; \\ &= \sum_{i=1}^{m} \left(\left(\frac{k^2}{2c_s} \log \frac{\epsilon}{\epsilon_0} \right) \frac{k+1}{\sqrt{2\pi}} \, \frac{2^i}{i\sqrt{i}} \right) + t_{\mathrm{C}0} \ ; \\ &= \left(\frac{k^2(k+1)}{2c_s\sqrt{2\pi}} \log \frac{\epsilon}{\epsilon_0} \right) \sum_{i=1}^{m} \frac{2^i}{i\sqrt{i}} + t_{\mathrm{C}0} \ , \end{aligned} \tag{6.30}$$

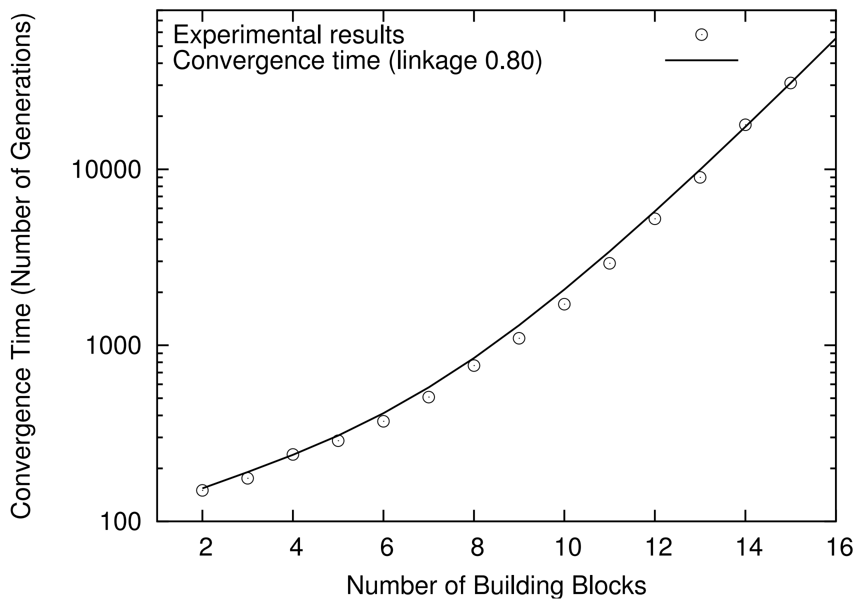
where c_s and $t_{\mathrm{C}0}$ are constants and determined empirically, m is the number of uniformly scaled building blocks, k is the length of the single building block, $\epsilon = 1 - \lambda$, and λ is the desired genetic linkage. As shown in Figure 6.14, the experimental results agree with the proposed convergence time model for the linkage learning genetic algorithm.

6.9 Summary

In this chapter, the sequential behavior was observed while the linkage learning genetic algorithm was solving both exponentially scaled problems and uniformly scaled problems. The first-building-block model was proposed accordingly from the macro view. By extending and integrating the linkage learning mechanisms, the tightness time model for a single building block to learn genetic linkage was developed from the micro view. Establishing the connection between these models, a convergence time model for the linkage learning genetic algorithm was constructed and empirically verified. The proposed convergence time model



(a)



(b) semi-log scale

Figure 6.14: Convergence time for the linkage learning genetic algorithm to solve problems composed of multiple uniformly scaled building blocks (the desired genetic linkage $\lambda = 0.80$). The experimental results are averaged over 50 independent runs (Chen & Goldberg, 2004a).

explains why the linkage learning genetic algorithm requires exponential time to solve uniformly scaled problems and gives us an insight into how the linkage learning genetic algorithm processes building blocks.

We identified a consistent, sequential behavior of the linkage learning genetic algorithm. It was previously believed that when solving a uniformly scaled problem, the linkage learning genetic algorithm works on all building blocks simultaneously, while when solving an exponentially scaled problem, the linkage learning genetic algorithm works on the most salient building block, the second most salient building block, and so on. By identifying the sequential behavior of the linkage learning genetic algorithm, we gain better understanding about how the linkage learning genetic algorithm works—on one building block at a time. The difference is that for exponentially scaled building blocks, the most salient building block is tightened first with a very high probability due to its selection advantage, but for uniformly scaled building blocks, each building block has the same probability to be tightened first. Recognizing the sequential behavior might shed light on developing a better design of the linkage learning genetic algorithm which can perform well on exponentially scaled problems as well as uniformly scaled problems.

The proposed convergence time model indicates that the time required by the linkage learning genetic algorithm to solve a uniformly scaled problem grows exponentially with the number of building blocks. The exponential growth of time, according to our analysis, mainly comes from the competition among the building blocks of equal salience. The decrease of the probability of crossover events increases the genetic linkage learning time correspondingly. Therefore, based on the model, the possible ways to improve the performance of the linkage learning genetic algorithm on uniformly scaled problems might include (1) effectively reducing the number of building blocks simultaneously processed by the linkage learning genetic algorithm and (2) employing certain mechanisms or procedures to make the linkage learning genetic algorithm to process building blocks sequentially. These two ways to improve the performance of the linkage learning genetic algorithm may be promising research directions.

Chapter 7

Introducing Subchromosome Representations

While the linkage learning genetic algorithm achieved successful genetic linkage learning on problems with badly scaled building blocks, it was less successful on problems consisting of uniformly scaled building blocks. The convergence time model for the linkage learning genetic algorithm developed in the previous chapter explains the difficulty faced by the linkage learning genetic algorithm and reveals the performance limit of the linkage learning genetic algorithm on uniformly scaled problems. This chapter seeks to enhance the design of the linkage learning genetic algorithm based on the time models in order to improve the performance of the linkage learning genetic algorithm.

The study initiates a better design of the linkage learning genetic algorithm that can lead to scalable genetic linkage learning. In particular, the subchromosome representation is proposed in this chapter to avoid the performance limit implied by the convergence time model and to escape from the *combinatorial overload*, which is caused by attempts to move too much genetic material on the chromosome. This chapter focuses on the following topics:

- Introducing the subchromosome representation;
- Verifying the utility of the subchromosome representation.

The chapter starts with the limit to competence of the linkage learning genetic algorithm indicated by the convergence time model. Then, it presents a preliminary implementation of the proposed representation. Finally, the experimental results that demonstrate the utility of subchromosomes are presented.

7.1 Limit to Competence of the LLGA

First of all, before providing the prescription to remedy the performance issue of the linkage learning genetic algorithm, we need to understand the implication of the convergence time model and then enhance the design of the linkage learning genetic algorithm accordingly. In addition to describing the way the linkage learning genetic algorithm works on uniformly scaled problems as well as explaining the seemingly inconsistent behavior of the linkage learning genetic algorithm on problems with building blocks of different scalings, the convergence time model (Equation (6.30)) for the linkage learning genetic algorithm proposed in chapter 6 also reveals a critical limit to competence of the linkage learning genetic algorithm that the computational time for the linkage learning genetic algorithm to solve uniformly scaled problems grows exponentially with the number of building blocks in the problem.

By examining the proposed convergence time model more carefully, we can find that the parameters involved in the model are either the properties of the problem to solve, such as the order of building blocks, k, and the number of building blocks, m, or the uncontrollable constants and variables, such as the linkage-skew coefficient, c_s, and the level of linkage, ϵ. Unlike many facetwise models that contain algorithmic parameters and can shed light on how to appropriately set these parameters to enable the genetic algorithm, little guidance can be obtained from the time model for setting the existing algorithmic parameters of the linkage learning genetic algorithm. For example, the schema theorem (De Jong, 1975; Holland, 1975; Goldberg, 1989c) describes the market share growth of building blocks in terms of selection pressure and crossover probability. It shows us the correct way to choose these two parameters. However, the convergence time model cannot provide us such practical indications because there is no algorithmic parameters involved in the model. Therefore, instead of trying to adjust those existing algorithmic parameters, another way to improve the performance of the linkage learning genetic algorithm on uniformly scaled problems has to be taken. In the remainder of this chapter, we seek a new design to enhance the linkage learning genetic algorithm based on the insight provided by the convergence time model, take an initial step to realize the design, and present the preliminary experimental results.

7.2 Subchromosome Representations

According to the convergence time model for the linkage learning genetic algorithm described by Equation (6.30), one possible way to enhance the performance of the linkage learning genetic algorithm on uniformly scaled problems is to modify the LLGA chromosome representation such that the number of building blocks, m, is effectively lowered at run time. Thus, the exponential growth of convergence time can be reduced. This section introduces the subchromosome representation to the linkage learning genetic algorithm (Chen & Goldberg, 2004b). The subchromosome representation is first described in detail, and then, the exchange crossover operator for handling subchromosome representations is discussed.

7.2.1 Chromosome Representation

The subchromosome representation in the linkage learning genetic algorithm separates a LLGA chromosome into several parts, called *subchromosomes*. The structure of a subchromosome is identical to that of a regular LLGA chromosome. Like an LLGA chromosome described in sections 3.1 and 5.2.1, a subchromosome contains moveable genes, non-coding segments, as well as promoters and is interpreted with probabilistic expression. The union of all subchromosomes belonging to one individual forms a complete LLGA chromosome. In subchromosome representations, there is no separate fitness measurement for each subchromosome. The fitness that corresponds to the solution obtained from interpreting the complete chromosome is used by all subchromosomes. Therefore, the unimetric character-

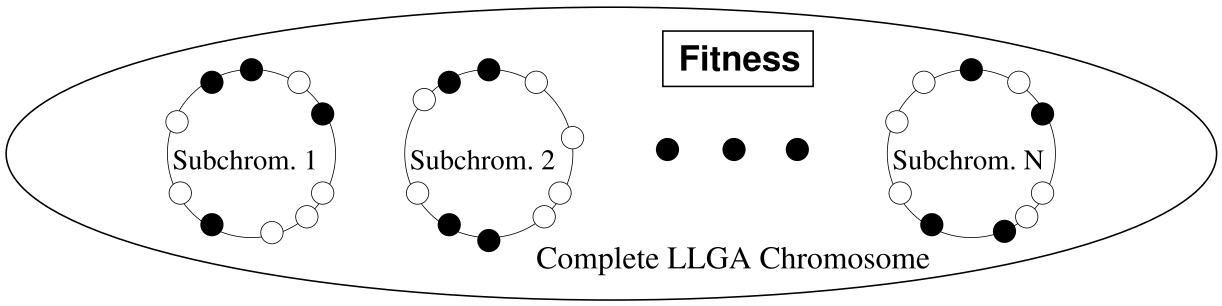


Figure 7.1: The structure of a subchromosome is identical to that of an LLGA chromosome. Each subchromosome contains moveable genes, non-coding segments, as well as promoters and is interpreted with probabilistic expression. The union of all subchromosomes belonging to one individual forms a complete LLGA chromosome. The fitness corresponding to the solution obtained from interpreting the complete chromosome is considered the fitness of each subchromosome.

istic of the linkage learning genetic algorithm is still maintained because there is no extra measurement regarding the structure of the chromosome added into the fitness. Figure 7.1 shows an LLGA chromosome consisting of subchromosomes.

The goal of subchromosome representations is to create a flexible encoding mechanism that makes LLGA chromosomes capable of grouping closely related building blocks to form higher-level building blocks in addition to moving genes together on the chromosome to form the first-level building blocks. Similar to genetic linkage learning, the process of forming higher-level building blocks should be integrated with the evolutionary and problem-solving process. The subchromosomes of an LLGA chromosome shown in Figure 7.1 can be considered as building blocks of the second level. The subchromosome representation can be designed to hierarchically express building blocks of even higher levels, such as the third level, the fourth level, and so on.

However, as an initial step to realize this representation scheme and as a pilot study of the effect to use subchromosomes in the linkage learning genetic algorithm, only subchromosomes of the second level are implemented and examined in this chapter. Moreover, the groups of building blocks are pre-defined as well as correctly placed, and genes on each subchromosome do not migrate to other subchromosomes. Within a subchromosome, genes and non-coding segments are still randomly distributed in the initialization step as they are on a regular LLGA chromosome without subchromosomes.

7.2.2 Exchange Crossover

Due to the adoption of the new representation, the exchange crossover operator is modified to handle subchromosomes. Since in this chapter, the subchromosomes are pre-defined and do not exchange its own genetic material with other subchromosomes as described in the previous section, for simplicity, after determining the donor and the recipient, the exchange crossover operator works on subchromosomes one by one. For a pair of subchromosomes, one from the donor and the other from the recipient, exchange crossover works as it does on conventional LLGA chromosomes as described in sections 3.2 and 5.2.2. It cuts the genetic material from the subchromosome of the donor and injects them into the corresponding subchromosome of the recipient. The transferred genetic material is determined at random for each pair of subchromosomes. Figure 7.2 shows how the modified exchange crossover operator works on a pair of LLGA chromosomes consisting of subchromosomes.

7.3 Empirical Verification

The experiments to observe the effect of using the subchromosome representation in the linkage learning genetic algorithm are presented in this section. First, the parameter settings of the experiments are described in detail. Then, the experimental results are shown in the remainder of this section.

7.3.1 Experimental Settings

As we did in the previous chapters, trap functions (Ackley, 1987; Deb & Goldberg, 1993; Deb, Horn, & Goldberg, 1993; Deb & Goldberg, 1994) described in section 4.2 are also used for examining the effect of adopting subchromosome representations in the linkage learning

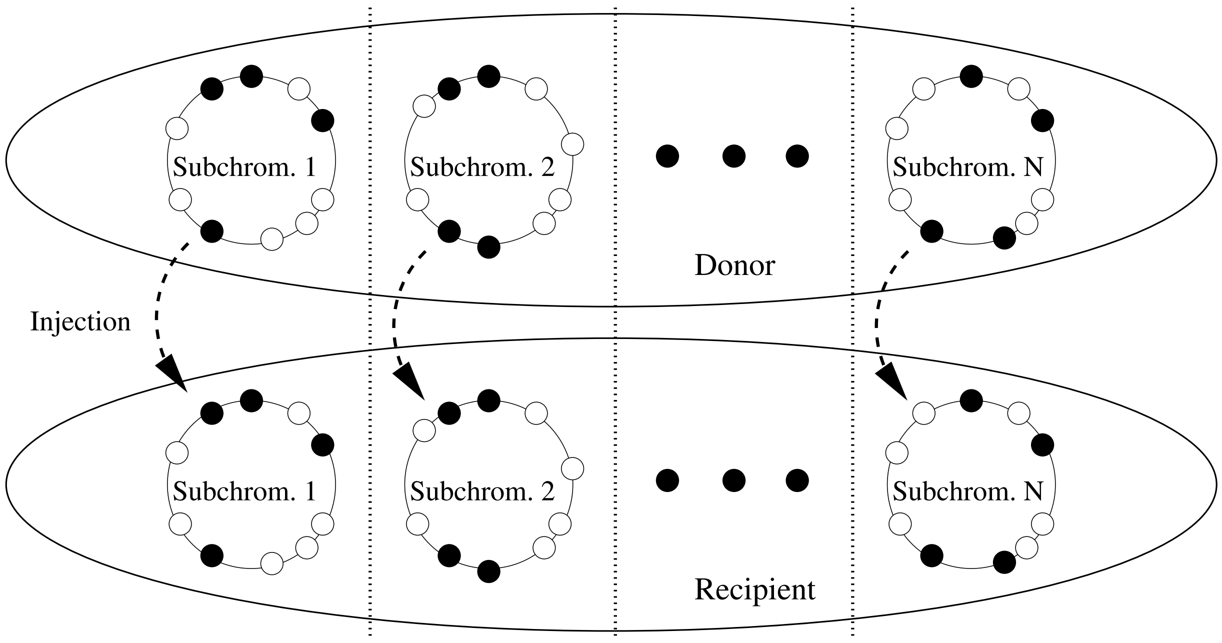


Figure 7.2: For a pair of subchromosomes, exchange crossover works as it does on conventional LLGA chromosomes. The operator cuts the genetic material from the subchromosome of the donor and injects them into the corresponding subchromosome of the recipient. The transferred genetic material is determined at random for each pair of subchromosomes.

genetic algorithm because trap functions provide decent linkage structures, and good linkage is required in order to solve problems consisting of traps. The experiments in this chapter were done for order-4 traps, and all traps contribute equally to the fitness. In particular, the order-4 trap function is expressed as

$$\mathrm{trap}_4(u) = \begin{cases} u & u = 4 \\ 3 - u & \text{otherwise} \end{cases} . \tag{7.1}$$

To simulate the infinite-length chromosome, we let one order-4 building block embedded in 250 genes, including functional genes and non-coding elements. For example, for five order-4 building blocks, the 20 genes are embedded in a 1,250-gene chromosome with 1,230 non-coding elements. Table 7.1 lists all experiments conducted for determining the effect of using subchromosome representations in the linkage learning genetic algorithm. The total number of building blocks in one experiment is n_{bb} (the number of building blocks per

Table 7.1: All experiments conducted for examining the effect of using subchromosome representations in the linkage learning genetic algorithm. From 2 to 8 building blocks per subchromosome, all conditions for the total number of building blocks less than or equal to 80 are included in the experiments in this chapter.

BBs per Subchromosome (n_{bb})	Number of Subchromosomes (n_{s})
2	2, 3, 4, 5, 6, . . . , 36, 37, 38, 39, 40
3	2, 3, 4, 5, 6, . . . , 22, 23, 24, 25, 26
4	2, 3, 4, 5, 6, . . . , 16, 17, 18, 19, 20
5	2, 3, 4, 5, 6, . . . , 12, 13, 14, 15, 16
6	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
7	2, 3, 4, 5, 6, 7, 8, 9, 10, 11
8	2, 3, 4, 5, 6, 7, 8, 9, 10

subchromosome) $\times$ n_{s} (the number of of subchromosomes). From 2 to 8 building blocks per subchromosome, all conditions for the total number of building blocks less than or equal to 80 are included in the experiments in this chapter.

The gambler's ruin model (Harik, Cantú-Paz, Goldberg, & Miller, 1997) described in section 1.3 is utilized for population sizing. Other parameters are set as follows. The crossover rate is 1.0 such that the crossover event always happens. The maximum number of generations is 100,000. The number of promoters on each subchromosome is set to $2m$, where m is the number of building blocks on the subchromosome. Finally, each experiment was repeated with 50 independent runs.

7.3.2 Experimental Results

For each experiment listed in Table 7.1, the success rate is calculated according to the results obtained in the 50 independent runs. Here, a success is determined by the final solution quality. The solution quality is the ratio between the number of correctly solved building blocks in the end of the run and that of the total building blocks in the trial. For example, if in a particular run for solving 20 building blocks, 12 building blocks are correctly solved, the solution quality of this run is 0.6. If the final solution quality of a run is equal to or greater

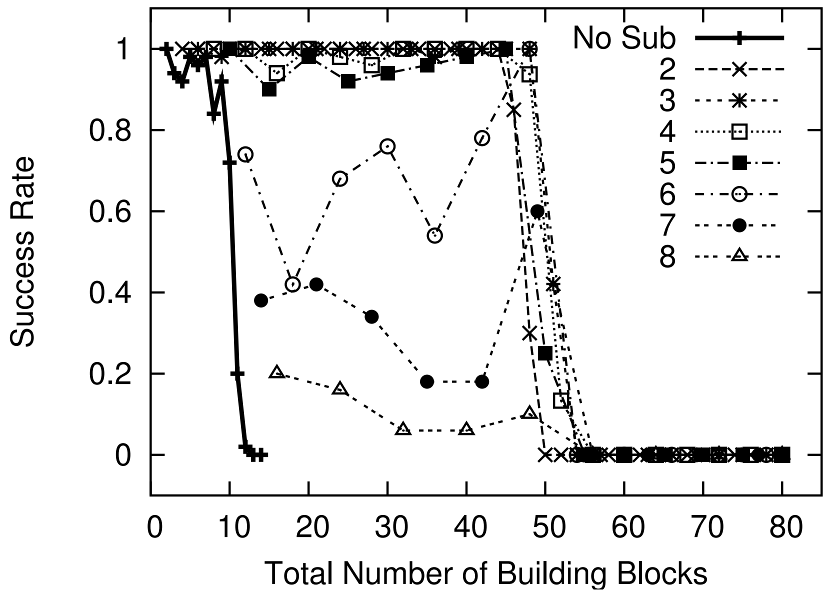


Figure 7.3: Results of the experiments with less than or equal to 80 building blocks. The number of building blocks distributed on each subchromosome varies from 2 to 8. "No Sub" indicates the linkage learning genetic algorithm without the subchromosome representation. The results show that utilizing subchromosome representations in the linkage learning genetic algorithm can significantly improve its performance on the uniformly scaled problems up to five times higher in terms of the total number of building blocks. These results are collected from 50 independent runs for each combination of the number of subchromosomes and the number of building blocks on one subchromosome (Chen & Goldberg, 2004b).

than 0.9, the run is recorded as a success. The success rate is therefore the ratio between the number of success trials and that of the total runs.

Figure 7.3 gives the success rates of all the experiments with the total number of building blocks less than or equal to 80 as listed in Table 7.1. The number of building blocks distributed on each subchromosome varies from 2 to 8. The results for each number of building blocks on subchromosomes are shown with different line-point styles in the figure. As shown in Figure 7.3, utilizing subchromosome representations in the linkage learning genetic algorithm can significantly improve the performance of the linkage learning genetic algorithm on the uniformly scaled problems. Compared to the results reported in chapter 5, the linkage learning genetic algorithm with the subchromosome representation can solve uniformly scaled problems about five times larger in terms of the total number of building blocks than that can be solved by the linkage learning genetic algorithm without subchromosomes.

In addition to the performance improvement, Figure 7.3 also shows that the limit for the linkage learning genetic algorithm with the subchromosome representation in our first attempt of implementation to solve uniform scaled problems seems to be around 50 in terms of the total number of building blocks. Even with different numbers of building blocks distributed on one subchromosome, no successful trial was found among all the experiments with totally 60 or more building blocks. However, according to the importance of building-block identification and exchange we discussed in section 1.4 and the insight we gained from studying the linkage learning genetic algorithm, we had to use a crossover probability as high as 1.0 in the linkage learning genetic algorithm without subchromosomes in order to effectively promote the linkage learning process (building-block identification), which proceeds with only the *differential selection of linkage* (Goldberg, 2002), which is generated indirectly from the schema theorem (Holland, 1975; Goldberg, 1989c; Goldberg & Sastry, 2001).

Such a high crossover probability not only promotes the linkage learning process but also raises the probability to disrupt building blocks (Thierens & Goldberg, 1993; Thierens, 1995). Therefore, by using the same setting of the crossover probability in these experiments, the procedure to apply exchange crossover on subchromosomes may cause serious building-block disruption, as it does in the linkage learning genetic algorithm without subchromosomes. While the original design of the linkage learning genetic algorithm prevents us from lowering the disruption rate and maintaining the mixing rate at the same time, the linkage learning genetic algorithm with the subchromosome representation provides us a viable way to appropriately adjust the probability for applying the exchange crossover operator. Because an LLGA chromosome is now composed of several subchromosomes as proposed, although the exchange crossover operator executes the same task on a pair of subchromosomes, we can lower the overall crossover probability by randomly choosing some pairs of subchromosomes to go through exchange crossover. In such a manner, it is now possible for us to adjust the crossover probability without losing too much mixing capability.

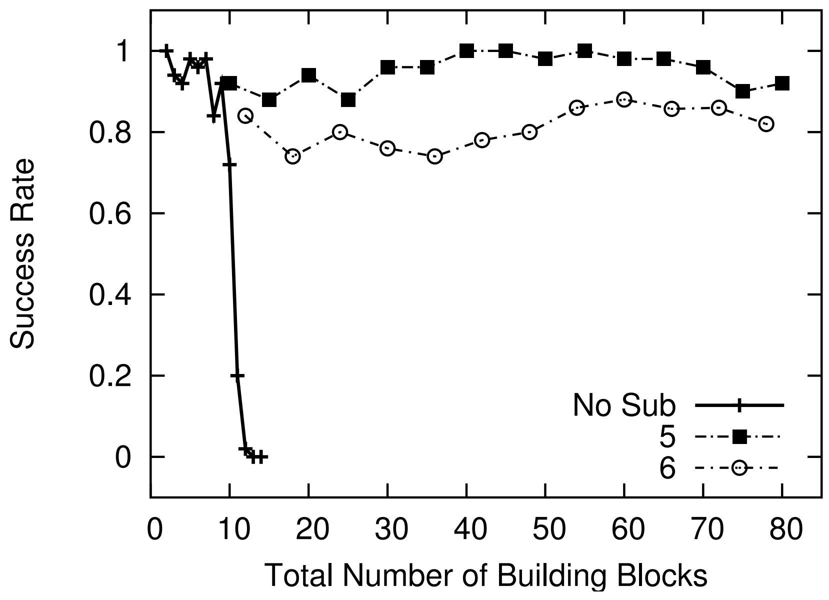


Figure 7.4: Instead of applying exchange crossover to all pairs of subchromosomes, each pair is now chosen with a probability of 0.5 for applying exchange crossover in order to reduce building-block disruption as much as possible without influencing the mixing rate too much. The results indicate that the subchromosome representation works well in the range of these experiments. These results are collected from 50 independent runs for each combination of the number of subchromosomes and the number of building blocks on one subchromosome (Chen & Goldberg, 2004b).

Based on the discussion, the exchange crossover operator is slightly modified as follows. Instead of applying exchange crossover to all pairs of subchromosomes, each pair is now chosen with a probability of 0.5 for applying exchange crossover in order to reduce building-block disruption as much as possible without influencing the mixing rate too much. The previous experiments were repeated for only $n_{\mathrm{bb}} = 5$ and 6 to check the effect of adjusting the crossover probability. The results are shown in Figure 7.4 and indicate that the subchromosome representation works well in the range of these experiments.

7.4 Summary

This chapter started with describing the limit to competence of the linkage learning genetic algorithm. The subchromosome representation was employed in the linkage learning genetic algorithm for effectively lowering the number of building blocks to escape from the limit implied by the convergence time model. An initial step to realize subchromosome representations in the linkage learning genetic algorithm was taken in this chapter. The preliminary experimental results of using subchromosomes in the linkage learning genetic algorithm indicated that the proposed coding scheme can improve the performance of the linkage learning genetic algorithm on uniformly scaled problems.

In addition to showing that the subchromosome representation can improve the performance of the linkage learning genetic algorithm, the initial step for implementing the representation also leads to a possible way to make the linkage learning genetic algorithm capable of incorporating prior linkage information. With the use of subchromosomes, the distribution of genes, non-coding segments, and building blocks can be determined according to the available linkage information. In the linkage learning genetic algorithm without subchromosomes, utilizing prior linkage information is extremely difficult if not impossible. Overall, the results reveal a promising path for achieving scalable genetic linkage learning.

Finally, before running off to try more new variations of the algorithm and to do more mass quantities of computation, we should think carefully about the key lessons of this chapter. In going from the limited results of Figure 7.3 to the much better results of Figure 7.4, we recall that the primary difference was the limited amount of mechanical disruption that was permitted in the modified subchromosome exchange crossover. In looking back over all studies of the linkage learning genetic algorithm to this point, it is clear that these procedures can only tolerate a certain amount of *rearrangement disruption.* Large amounts of fitness variance are not problematic because they can be overcome through larger populations; however, attempts to move too much genetic material around have always caused a combinatorial overload that cannot be sorted out. In designing mechanisms to move genes around either physically or virtually, we must recognize that the overall structure can assimilate only so much movement at any one time. Attention to this should guide the design of mechanisms to realize the potential of the linkage learning genetic algorithm.

Chapter 8

Conclusions

This chapter concludes the dissertation. It starts with the summary of the progress, results, and status of the research project, followed by tasks of potential future projects, including the tasks that can proceed immediately and those that contribute to our long-term goals and objectives. Finally, the main conclusions from this dissertation are discussed.

8.1 Summary

In this dissertation, we studied the scalability of the linkage learning genetic algorithm from both theoretical and practical aspects. After identifying the drawback of the original linkage learning genetic algorithm that there was no proper mechanism to separate uniformly scaled building blocks, the use of promoters on the chromosome was proposed to improve nucleation potential and avoid misnucleation. In order to advance our understanding of the operations of the linkage learning genetic algorithm in theory, by integrating the observed sequential behavior, the tightness time model, and the connection between the two models from different views, the convergence time model was constructed for explaining the behavior of the linkage learning genetic algorithm. Based on the convergence time, the subchromosome representation was designed for the linkage learning genetic algorithm to escape from the limit to speed of convergence indicated by the convergence time model, and an initial step to realize the design was already taken.

Chapter 1 contained an introduction of genetic algorithms. Definitions of genetic algorithm terms and the global flow-control of a simple genetic algorithm were presented. The design-decomposition theory and the gambler's ruin model for population sizing were described. Then, it introduced the concept of genetic linkage in both biological systems and genetic algorithms. The linkage problem, also known as the ordering problem, that exists in common genetic algorithm practice was described in detail, and the importance of handling and learning genetic linkage in genetic algorithms was also discussed. Chapter 1 connected the fields of biology and genetic algorithms, provided a potential way to interpret the results in the context of biology, and presented the key reason to develop linkage learning techniques.

Chapter 2 provided a set of classifications of existing genetic linkage learning techniques such that different views from several facets of these techniques were revealed and depicted. It also presented the lineage of the linkage learning genetic algorithm to demonstrate how it was developed and constructed from its precursors and ancestors and identified the position of the linkage learning genetic algorithm among the existing genetic linkage learning techniques so that the connection to other methodologies can be established and the importance of the linkage learning genetic algorithm can be emphasized. Chapter 2 helped to understand the different views of the existing genetic linkage learning techniques as well as the relation between the linkage learning genetic algorithm and other linkage learning methodologies.

Chapter 3 described in detail the linkage learning genetic algorithm, including (1) the chromosome representation, (2) the exchange crossover operator, (3) two mechanisms that enable the linkage learning genetic algorithm, (4) accomplishments of the linkage learning genetic algorithm, and (5) difficulties encountered by the linkage learning genetic algorithm. The linkage learning genetic algorithm uses moveable genes, non-coding elements, exchange crossover, and a special expression mechanism, probabilistic expression, to create an evolvable genotypic structure that makes genetic linkage learning natural and viable for genetic algorithms. The identified linkage learning mechanisms revealed how the linkage learning genetic algorithm learns the linkage of a single building block and described the evolution of genetic linkage under different mechanisms. The accomplishments of the linkage learning genetic algorithm were included, and main difficulties faced by the linkage learning genetic algorithm were presented in this chapter.

Chapter 4 presented the assumptions regarding the framework based on which we developed the theoretical models as well as regarding the genetic algorithm structure we adopted in this work. After introducing the key assumptions that we employed throughout this dissertation, the definition of the elementary test problem, which were trap functions, and the construction of the larger test problems used in the study were described in detail. Chapter 4 provided a background establishment for the following chapters.

Chapter 5 introduced the use of promoters and a modified exchange crossover operator to work with promoters in the linkage learning genetic algorithm to improve its performance on uniformly scaled problems. In contrast to the original linkage learning genetic algorithm, in which every non-coding element and functional gene can be the point of interpretation of the offspring, only promoters can be the new point of interpretation after crossover in the modified version. Chapter 5 first investigated the difficulty encountered by the original linkage learning genetic algorithm when it was working on uniformly scaled problems. Then, a coding mechanism that exists in genetics to overcome the difficulty was proposed. Promoters, the modified exchange crossover operator, and corresponding modifications to the linkage learning genetic algorithm were described in detail. The experimental results provided in this chapter demonstrated that the linkage learning genetic algorithms with promoters was able to overcome the difficulty encountered by the original version.

Chapter 6 developed the convergence time model for the linkage learning genetic algorithm step by step. It started with describing the settings of all the experiments for observing the behavior of the linkage learning genetic algorithm and verifying the theoretical results. Based on the observation, the sequential behavior of the linkage learning genetic algorithm was identified from the macro view. By extending the two linkage learning mechanisms, the tightness time model for a single building block to learn genetic linkage was proposed from the micro view. By establishing the connection between the sequential behavior and the tightness time model for the cases of multiple building blocks, a convergence time model for the linkage learning genetic algorithm was constructed. Although adopting promoters on the chromosome enhances the performance of the linkage learning genetic algorithm on uniformly scaled problems, better understanding of the algorithm from a theoretical point of view should be still addressed in order to design a better algorithm. Chapter 6 aimed to gain better understanding of the linkage learning genetic algorithm in theory. By constructing a convergence time model, we are now able to explain why the linkage learning genetic algorithm needs exponentially growing time to solve uniformly scaled problems.

Chapter 7 proposed the use of subchromosome representations in the linkage learning genetic algorithm. It started with the limit to competence of the linkage learning genetic algorithm indicated by the convergence time model. Then, the subchromosome representation was described in detail, including how the conventional LLGA chromosome was encoded with subchromosomes and how the exchange crossover was modified to work with these subchromosomes. The experimental results for observing the utility of subchromosomes were also presented to demonstrate that the use of subchromosome representations may be a promising way to design a better linkage learning genetic algorithm. The subchromosome representation was developed to avoid the performance limit implied by the convergence time model. Chapter 7 initiated a better design of the linkage learning genetic algorithm that may lead to scalable genetic linkage learning. It presented a preliminary implementation of the proposed representation and verified the performance improvement with empirical results.

8.2 Future Work

This dissertation has investigated the difficulty faced by the linkage learning genetic algorithm in theory with developing the time models, including the tightness time and convergence time, and extended the scalability of the linkage learning genetic algorithm by devising new chromosome designs, such as the use of promoters and subchromosomes. Based on the results in both theory and practice obtained in this work, the following is a list of research directions for future consideration suggested by the author.

- **Realization of the subchromosome representation.** The experimental results presented in chapter 7 are tantalizing in that parallel evolution of linkage of large number of gene groupings has been demonstrated provided the appropriate genes are associated in the same linkage group. The key challenge left is to develop mechanisms to permit or encourage the evolution of these proper associations. Different mechanisms can be imagined for this purpose, and the potential of each is introduced and briefly outlined in the following paragraphs:

  Gene migration: Gene migration moves genes among subchromosomes belonging one LLGA chromosome. Proper associations can be achieved through gene migration and favored by the evolutionary process.

  Gene duplication or redundant genes: Similar to using coexisting alleles of one gene on LLGA chromosomes with probabilistic expression, gene duplication increases the probability to form correct gene groups or clusters on subchromosomes by introducing redundant genes.

  Adaptive expression: Working with the redundant genes on subchromosomes, adaptive expression resolves the conflicts caused by gene duplication and promotes those building blocks which are correctly identified.

- **Formal methods for designing genetic operators.** While genetic and evolutionary algorithms are proposed for solving black-box-optimization problems, many successful applications of genetic and evolutionary algorithms actually use certain customized operators or specialized components, such as problem specific chromosome representations or specially designed genetic operators involving the problem domain knowledge. Although the representation issue has been tackled in the field of genetic linkage learning techniques, still, the design of genetic operators is usually done in an ad-hoc manner. One of the possible ways to achieve formal methods for designing genetic operators is to adopt the current research implications from genetic linkage learning techniques, such as the linkage learning genetic algorithm. Looking into the computational results of the linkage learning step and designing genetic operators based on those results should be a promising direction to develop formal methods for designing genetic operators to handle specific problems.

- **Practical linkage learning genetic algorithm.** The linkage learning genetic algorithm has been known to perform well on exponentially scaled problems and poorly on uniformly scaled problems as mentioned in chapter 3. This dissertation further demonstrates that the convergence time which the linkage learning genetic algorithm requires for solving uniformly scaled problems grows exponentially. However, because the scaling of building blocks in practical problems might not be necessarily strictly uniform like what we use for studying the behavior of the algorithm, the linkage learning genetic algorithm might still be used to handle real-world applications, although this research project does not focus on solving real-world problems. According to our recent study which indicates that the signal difference between building blocks does not need to be significantly large for the linkage learning genetic algorithm to perform as expected, it seems promising to put the linkage learning genetic algorithm in the practical use and identify the appropriate problem classes for the algorithm.

- **Incorporating prior linkage information.** As pointed out in section 7.4, according to the original design of the LLGA chromosome, it is very difficult for the linkage learning genetic algorithm to utilize prior linkage information when such knowledge is available. Several extra operators or mechanisms have to be added into the linkage learning genetic algorithm in order to implement this ability. With the help of subchromosomes, incorporating available linkage information becomes viable and easy. Genes closely related to one another can be simply placed on the same subchromosome, while those unrelated genes can be placed on other subchromosomes. This extension to the linkage learning genetic algorithm should be useful for the practical use.

- **Theory on genetic linkage models.** Yu and Goldberg (2004) developed a theoretical framework on the quality and efficiency of model building for genetic algorithms, which is closely related to the theoretical aspect of the linkage learning genetic algorithm, because the distributions of functional genes as well as non-coding elements on the LLGA chromosome can be considered as implicit probabilistic models. It is possible to utilize the same theoretical results or to employ the similar modeling technique to derive an equivalent critical number of errors for the linkage learning genetic algorithm. Such derivation might shed light on explaining the observation that the linkage learning genetic algorithm can only tolerate a certain amount of rearrangement disruption and provide better understanding of building genetic linkage models.

- **Biological ramifications.** Since genetic and evolutionary algorithms were developed based on the paradigms and principles of evolution in nature, it is possible to find a way to channel the theoretical and computational results of this field into the related fields of biology. For the linkage learning genetic algorithm, as a unimetric approach, the interpretation and translation of those theoretical models and practical frameworks in the context of biology might serve as highly simplified computational models and possibly provide insights from different points of view. Therefore, interdisciplinary cooperation is in order for this research direction such that our understanding of genetic and evolutionary computation as well as biology and nature might be advanced.

8.3 Main Conclusions

This dissertation addresses the scalability issue of the linkage learning genetic algorithm from both theoretical and practical aspects. By making use of simple, dimensional models, we gain better understanding of the behavior of the linkage learning genetic algorithm in theory. By adopting coding mechanisms existing in genetics and biological systems, we improve the performance of the linkage learning genetic algorithm on uniformly scaled problems in practice. According to the status and outcomes of this research project, the main conclusions that may be drawn from this dissertation are listed as follows:

- **Promoters improve nucleation potential.** The observation of the genetic linkage learning process showed that the randomness of choosing cutting points, grafting points, and points of interpretation caused the instability of allele expression and led to misnucleation of building blocks. By introducing promoters to the chromosome representation, such randomness was effectively reduced, and the modified linkage learning genetic algorithm was made able to solve more uniformly scaled building blocks than the original version could. Therefore, promoters can improve nucleation potential and promote correct formation of building blocks.

- **The linkage learning genetic algorithm has a consistent, sequential behavior.** It was previously believed that when solving a uniformly scaled problem, the linkage learning genetic algorithm worked on all building blocks simultaneously, while when solving an exponentially scaled problem, the linkage learning genetic algorithm worked on the most salient building block, the second most salient building block, and so on. However, in this dissertation, we successfully identified a consistent, sequential behavior of the linkage learning genetic algorithm. By identifying the sequential behavior, we gained better understanding about how the linkage learning genetic algorithm handled and processed the building blocks of different scalings.

- **Tightness time is derived based on linkage skew and linkage shift.** Harik and Goldberg (1996) proposed two linkage learning mechanisms, linkage skew and linkage shift, to explain why and how the linkage learning genetic algorithm worked. In this dissertation, we extended and combined the two linkage learning mechanisms and derived the tightness time model for describing the time needed to tighten a single building block. This time model was obtained from the micro view of the linkage learning process and played an essential role in constructing the convergence time model for the linkage learning genetic algorithm.

- **Competition among building blocks of equal salience slows down the genetic linkage learning process.** According to section 6.7, one of the key components in the convergence time model that contributes to the time delay of the linkage learning genetic algorithm is the interaction or competition among multiple building blocks of equal salience. The competition severely slows down the genetic linkage learning process because the probability of linkage learning events is significantly reduced.

- **Convergence time grows exponentially with the number of building blocks of equal salience.** As shown by Equation (6.30), given a fixed length of the building block and a desired genetic linkage, the convergence time for the linkage learning genetic algorithm to solve a uniformly scaled problem grows exponentially with the number of building blocks. Such a high order of computational time poses a limit to competence of the linkage learning genetic algorithm because the time required to finish a computation renders the problem infeasible to be solved.

- **Subchromosome representations reduce the competition among building blocks.** The subchromosome representation was employed in the linkage learning genetic algorithm for effectively lowering the number of building blocks, and therefore, it reduced the competition among building blocks by separating them on different subchromosomes. An initial step to realize subchromosome representations was taken in this dissertation, and the preliminary experimental results indicated that the proposed coding scheme can improve the performance of the linkage learning genetic algorithm on uniformly scaled problems.

- **Prior linkage information can be incorporated into the linkage learning genetic algorithm.** The initial step for implementing the subchromosome representation in this dissertation also led to a possible way to make the linkage learning genetic algorithm capable of incorporating prior linkage information. In the linkage learning genetic algorithm without subchromosomes, utilizing prior linkage information is extremely difficult because of those necessary, significant modifications. With the use of subchromosomes, the distribution of genes, non-coding segments, and building blocks can be appropriately arranged in order to utilize the available linkage information.
- **Scalable genetic linkage learning for a unimetric approach is possible.** By using the subchromosome representation, the linkage learning genetic algorithm might be able to solve uniformly scaled problems of reasonable sizes. Although we took only an initial step to implement the proposed representation, the experimental results obtained in this work indicated a promising direction which should lead to scalable genetic linkage learning for the linkage learning genetic algorithm as a unimetric approach.

More work along this line still needs to be done from both theoretical and practical aspects. From the theoretical aspect, more accurate and sophisticated models of the linkage learning process are required for further understanding the nature of genetic linkage learning. Advancing our knowledge on linkage learning or building-block identification in theory may shed light on better designs of genetic algorithms. On the other hand, from the practical aspect, the results presented in this work reveal a promising path for achieving scalable genetic linkage learning. New representations, linkage learning mechanisms, or building-block identification procedures should be investigated, constructed, and verified for improving the performance of the linkage learning genetic algorithm.

Bibliography

Abramowitz, M., & Stegun, I. (1972). *Handbook of mathematical functions with formulas, graphs, and mathematical tables*. New York, NY: Dover Publications.

Ackley, D. H. (1987). *A connectionist machine for genetic hill climbing*. Boston: Kluwer Academic.

Bäck, T. (1995). Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA-95)*, 2–8.

Bagley, J. D. (1967). *The behavior of adaptive systems which employ genetic and correlation algorithms*. Doctoral dissertation, University of Michigan, Ann Arbor, MI. (University Microfilms No. 68-7556).

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. *Proceedings of the International Conference on Genetic Algorithms and Their Applications*, 101–111.

Blickle, T., & Thiele, L. (1995). A mathematical analysis of tournament selection. *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA-95)*, 9–16.

Blickle, T., & Thiele, L. (1996). A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, *4* (4), 361–394.

Cantú-Paz, E. (1999). *Designing efficient and accurate parallel genetic algorithms*. Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL. (Also IlliGAL Report No. 99017).

Chen, Y.-p. (2002). Unpublished Experimental Data.

Chen, Y.-p., & Goldberg, D. E. (2002). Introducing start expression genes to the linkage learning genetic algorithm. *Proceedings of the Seventh International Conference on Parallel Problem Solving from Nature (PPSN VII)*, 351–360. (Also IlliGAL Report No. 2002007).

Chen, Y.-p., & Goldberg, D. E. (2003a). An analysis of a reordering operator with tournament selection on a GA-hard problem. *Proceedings of Genetic and Evolutionary Computation Conference 2003 (GECCO-2003)*, 825–836. (Also IlliGAL Report No. 2003003).

Chen, Y.-p., & Goldberg, D. E. (2003b). Tightness time for the linkage learning genetic algorithm. *Proceedings of Genetic and Evolutionary Computation Conference 2003 (GECCO-2003)*, 837–849. (Also IlliGAL Report No. 2003002).

Chen, Y.-p., & Goldberg, D. E. (2004a). Convergence time for the linkage learning genetic algorithm. *Proceedings of the 2004 Congress on Evolutionary Computation (CEC2004)*, N/A. (To appear, also IlliGAL Report No. 2003025).

Chen, Y.-p., & Goldberg, D. E. (2004b). Introducing subchromosome representations to the linkage learning genetic algorithms. *Proceedings of Genetic and Evolutionary Computation Conference 2004 (GECCO-2004)*, N/A. (To appear, also IlliGAL Report No. 2004001).

Chen, Y.-p., & Goldberg, D. E. (2005). Convergence time for the linkage learning genetic algorithm. *Evolutionary Computation*, N/A. (Accepted).

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

Deb, K. (1991). *Binary and floating-point function optimization using messy genetic algorithms.* Doctoral dissertation, University of Alabama, Tuscaloosa, AL. (Also IlliGAL Report No. 91004).

Deb, K., & Goldberg, D. E. (1991). *mGA in C: A messy genetic algorithm in C* (IlliGAL Report No. 91008). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Deb, K., & Goldberg, D. E. (1993). Analyzing deception in trap functions. *Foundations of Genetic Algorithms 2*, 93–108.

Deb, K., & Goldberg, D. E. (1994). Sufficient conditions for deceptive and easy binary functions. *Annals of Mathematics and Artificial Intelligence*, *10*, 385–408. (Also IlliGAL Report No. 92001).

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

Feller, W. (1970). *An introduction to probability theory and its applications.* New York, NY: Wiley.

Forrest, S., & Mitchell, M. (1993). Relative building-block fitness and the building-block hypothesis. *Foundations of Genetic Algorithms 2*, 109–126.

Frantz, D. R. (1972). *Nonlinearities in genetic adaptive search.* Doctoral dissertation, University of Michigan, Ann Arbor, MI. (University Microfilms No. 73-11116).

Goldberg, D. E. (1987). Simple genetic algorithms and the minimal, deceptive problem. In L., D. (Ed.), *Genetic Algorithms and Simulated Annealing* (Chapter 6, pp. 74–88). Los Altos, CA: Morgan Kaufmann Publishers. ISBN: 0-2730-8771-1.

Goldberg, D. E. (1989a). Genetic algorithms and Walsh functions: Part I, a gentle introduction. *Complex Systems*, *3*(2), 129–152. (Also TCGA Report 88006).

Goldberg, D. E. (1989b). Genetic algorithms and Walsh functions: Part II, deception and its analysis. *Complex Systems*, *3*(2), 153–171. (Also TCGA Report 89001).

Goldberg, D. E. (1989c, January). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley Publishing Co. ISBN: 0-201-15767-5.

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

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

Goldberg, D. E. (1999). The race, the hurdle, and the sweet spot: Lessons from genetic algorithms for the automation of design innovation and creativity. In Bentley, P. (Ed.), *Evolutionary Design by Computers* (Chapter 4, pp. 105–118). San Mateo, CA: Morgan Kaufmann.

Goldberg, D. E. (2002, June). *The design of innovation: Lessons from and for competent genetic algorithms*, Volume 7 of *Genetic Algorithms and Evoluationary Computation*. Kluwer Academic Publishers. ISBN: 1-4020-7098-5.

Goldberg, D. E., & Bridges, C. L. (1990). An analysis of a reordering operator on a GA-hard problem. *Biological Cybernetics*, *62*, 397–405. (Also TCGA Report No. 88005).

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

Goldberg, D. E., Deb, K., & Horn, J. (1992). Massive multimodality, deception, and genetic algorithms. *Proceedings of the Second International Conference on Parallel Problem Solving from Nature (PPSN II)*, 37–46. (Also IlliGAL Report No. 92007).

Goldberg, D. E., Deb, K., Kargupta, H., & Harik, G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, 56–64. (Also IlliGAL Report No. 93004).

Goldberg, D. E., Deb, K., & Korb, B. (1990). Messy genetic algorithms revisited: Studies in mixed size and scale. *Complex Systems*, *4*(4), 415–444.

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

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

Goldberg, D. E., & Liepens, G. (1991). Theory tutorial. (Tutorial presented at the 1991 International Conference on Genetic Algorithms, La Jolla, CA).

Goldberg, D. E., & Lingle, Jr., R. (1985). Alleles, loci, and the traveling salesman problem. *Proceedings of the International Conference on Genetic Algorithms and Their Applications*, 154–159.

Goldberg, D. E., & Sastry, K. (2001). A practical schema theorem for genetic algorithm design and tunning. *Proceedings of Genetic and Evolutionary Computation Conference 2001 (GECCO-2001)*, 328–335. (Also IlliGAL Report No. 2001017).

Goldberg, D. E., Sastry, K., & Latoza, T. (2001). On the supply of building blocks. *Proceedings of Genetic and Evolutionary Computation Conference 2001 (GECCO-2001)*, 336–342. (Also IlliGAL Report No. 2001015).

Grefenstette, J. J., & Baker, J. E. (1989). How genetic algorithms work: A critical look at implicit parallelism. *Proceedings of the Third International Conference on Genetic Algorithms (ICGA-89)*, 20–27.

Harik, G., 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 1997 IEEE International Conference on Evolutionary Computation*, 7–12.

Harik, G., Cantuú-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. R. (1997). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms.* Doctoral dissertation, University of Michigan, Ann Arbor, MI. (Also IlliGAL Report No. 97005).

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

Harik, G. R., & Goldberg, D. E. (1996). Learning linkage. *Foundations of Genetic Algorithms 4*, 247–262. (Also IlliGAL Report No. 96006).

Harik, G. R., & Goldberg, D. E. (2000, June). Learning linkage through probabilistic expression. *Computer Methods in Applied Mechanics and Engineering*, *186*(2–4), 295–310.

Hartl, D. L., & Jones, E. W. (1998, January). *Genetics: principles and analysis* (4th ed.). Sudbury, MA: Jones and Bartlett Publishers. ISBN: 0-7637-0489-X.

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

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press. ISBN: 0-262-58111-6.

Kargupta, H. (1995). *SEARCH, polynomial complexity, and the fast messy genetic algorithm.* Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL. (Also IlliGAL Report No. 95008).

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

Larrañaga, P., & Lozano, J. A. (2001, October). *Estimation of distribution algorithms: A new tool for evolutionary computation*, Volume 2 of *Genetic algorithms and evolutionary computation.* Boston, MA: Kluwer Academic Publishers. ISBN: 0-7923-7466-5.

Levenick, J. R. (1991). Inserting introns improves genetic algorithm success rate: Taking a cue from biology. *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA-91)*, 123–127.

Lobo, F. G. (2001). Personal Communications.

Lobo, F. G., Deb, K., Goldberg, D. E., Harik, G. R., & Wang, L. (1998, August). Compressed introns in a linkage learning genetic algorithm. In *Proceedings of the Third Annual Conference on Genetic Programming (GP 98)* (pp. 551–558). University of Wisconsin, Madison, WI: Morgan Kaufmann. (Also IlliGAL Report No. 97010).

Lobo, F. G., & Harik, G. R. (1999). *Extended compact genetic algorithm in C++* (IlliGAL Report No. 99016). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Merkle, L. D. (1996). *Analysis of linkage-friendly genetic algorithms.* Doctoral dissertation, Air Force Institute of Technology, Air University, Albuquerque, New Mexico. (Also Technical Report No. AFIT/DS/ENG 96-11).

Miller, B. L. (1997, May). *Noise, sampling, and efficient genetic algorithms.* Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL. (Also IlliGAL Report No. 97001).

Miller, B. L., & Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, *9*(3), 193–212. (Also IlliGAL Report No. 95006).

Miller, B. L., & Goldberg, D. E. (1996). Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, *4*(2), 113–131. (Also IlliGAL Report No. 95009).

Mühlenbein, H., & Mahnig, T. (1999). FDA - a scalable evolutionary algorithm for the optimization for the optimization of additively decomposed functions. *Evolutionary Computation*, *7*(4), 353–376.

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

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

Munetomo, M. (2002a). Linkage identification based on epistasis measures to realize efficient genetic algorithms. *Proceedings of the 2002 Congress on Evolutionary Computation (CEC2002)*, 1332–1337.

Munetomo, M. (2002b). Linkage identification with epistasis measure considering monotonicity conditions. *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL2002)*, 550–554.

Munetomo, M., & Goldberg, D. E. (1998). *Identifying linkage by nonlinearity check* (IlliGAL Report No. 98012). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Munetomo, M., & Goldberg, D. E. (1999). Identifying linkage groups by nonlinearity/non-monotonicity detection. *Proceedings of Genetic and Evolutionary Computation Conference 1999 (GECCO-99)*, 433–440.

Pelikan, M., & Goldberg, D. E. (2001). Escaping hierarchical traps with competent genetic algorithms. *Proceedings of Genetic and Evolutionary Computation Conference 2001 (GECCO-2001)*, 511–518. (Also IlliGAL Report No. 2001003).

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

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (2000). 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. G. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, *21*(1), 5–20. (Also IlliGAL Report No. 99018).

Reeves, C. (1993). Using genetic algorithms with small populations. *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, 92–99.

Riopka, T. P. (2002). *Intelligent recombination using genotypic learning in a Collective Learning Genetic Algorithm.* Doctoral dissertation, The George Washington University, Washington, DC.

Riopka, T. P., & Bock, P. (2000). Intelligent recombination using individual learning in a Collective Learning Genetic Algorithm. *Proceedings of Genetic and Evolutionary Computation Conference 2000 (GECCO-2000)*, 104–111.

Rosenberg, R. S. (1967). *Simulation of genetic populations with biochemical properties.* Doctoral dissertation, University of Michigan, Ann Arbor, MI. (University Microfilms No. 67-17836).

Salman, A. A., Mehrotra, K., & Mohan, C. K. (1998). Adaptive linkage crossover. *Proceedings of ACM Symposium on Applied Computing (SAC'98)*, 338–342.

Salman, A. A., Mehrotra, K., & Mohan, C. K. (1999). Linkage crossover operator. *Proceedings of Genetic and Evolutionary Computation Conference 1999 (GECCO-99)*, 564–571.

Salman, A. A., Mehrotra, K., & Mohan, C. K. (2000). Linkage crossover operator. *Evolutionary Computation*, *8*(3), 341–370.

Sastry, K. (2002, January). *Evaluation-relaxation schemes for genetic and evolutionary algorithms.* Master's thesis, University of Illinois at Urbana-Champaign, Urbana, IL. (Also IlliGAL Report No. 2002004).

Schaffer, J. D., & Morishima, A. (1987). An adaptive crossover distribution mechanism for genetic algorithms. *Proceedings of the Second International Conference on Genetic Algorithms (ICGA-87)*, 36–40.

Smith, J., & Fogarty, T. C. (1995). An adaptive poly-parental recombination strategy. *Proceedings of AISB-95 Workshop on Evolutionary computing*, 48–61.

Smith, J., & Fogarty, T. C. (1996). Recombination strategy adaptation via evolution of gene linkage. *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, 826–831.

Thierens, D. (1995). *Analysis and design of genetic algorithms*. Doctoral dissertation, Katholieke Universiteit Leuven, Leuven, Belgium.

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, 38–45.

Thierens, D., & Goldberg, D. E. (1994a). Convergence models of genetic algorithm selection schemes. *Proceedings of the Third International Conference on Parallel Problem Solving from Nature (PPSN III)*, 116–121.

Thierens, D., & Goldberg, D. E. (1994b). Elitist recombination: An integrated selection recombination ga. *Proceedings of the First IEEE International Conference on Evolutionary Computation*, 508–512.

Thierens, D., & Goldberg, D. E. (1998). Domino convergence, drift, and the temporal-salience structure of problems. *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, 535–540.

Wu, A. S., & Lindsay, R. K. (1995). Empirical studies of the genetic algorithm with noncoding segments. *Evolutionary Computation*, *3*(2), 121–147.

Wu, A. S., & Lindsay, R. K. (1996). A survey of intron research in genetics. *Proceedings of the Fourth International Conference on Parallel Problem Solving from Nature (PPSN IV)*, 101–110.

Wu, A. S., Lindsay, R. K., & Smith, M. D. (1994). Studies on the effect of non-coding segments on the genetic algorithm. *Proceedings of the Sixth IEEE Conference on Tools with Artificial Intelligence*.

Yu, T.-L., & Goldberg, D. E. (2004). Quality and efficiency of model building for genetic algorithms. *Proceedings of Genetic and Evolutionary Computation Conference 2004 (GECCO-2004)*, N/A. (To appear, also IlliGAL Report No. 2004004).

Yu, T.-L., Goldberg, D. E., Yassine, A., & Chen, Y.-p. (2003). Genetic algorithm design inspired by organizational theory: Pilot study of a dependency structure matrix driven genetic algorithm. *Proceedings of Artificial Neural Networks in Engineering 2003 (ANNIE 2003)*, 327–332. (Also IlliGAL Report No. 2003007).

Zwillinger, D., & Kokoska, S. (2000). *CRC standard probability and statistics tables and formulae*. Boca Raton, Florida: Chapman & Hall/CRC. ISBN: 1-58488-059-7.

Vita

Ying-ping Chen was born in Taipei, Taiwan, on November 20, 1972. He graduated from the National Taiwan University in 1995 with a Bachelor of Science in Engineering. In 1997, he completed a Master of Science in Computer Science from the National Taiwan University. Before relocating to Champaign, Illinois, USA, to pursue further graduate study in Computer Science, Chen joined the Taiwanese Air Force as a communication officer with a rank of Second Lieutenant to fulfill his military obligation. While his study in the Illinois Genetic Algorithms Laboratory and the Department of Computer Science at the University of Illinois at Urbana-Champaign, Chen took the opportunity and worked at Microsoft in Redmond, Washington, USA, as an intern in the summers of 2000 and 2001. Following the completion of his Ph.D., Chen will engage in academic work involving teaching and research.