Convergence Time for the Linkage Learning Genetic Algorithm

Ying-ping Chen[†] ypchen@csie.nctu.edu.tw
Department of Computer Science
National Chiao Tung University, Hsinchu City 300, TAIWAN

David E. Goldberg deg@uiuc.edu
Department of General Engineering
University of Illinois, Urbana, IL 61801, USA

Abstract

This paper identifies the sequential behavior of the linkage learning genetic algorithm, introduces the tightness time model for a single building block, and develops the connection between the sequential behavior and the tightness time model. By integrating the first-building-block model based on the sequential behavior, the tightness time model, and the connection between these two models, a convergence time model is constructed and empirically verified. The proposed convergence time model explains the exponentially growing time required by the linkage learning genetic algorithm when solving uniformly scaled problems.

Keywords

Genetic algorithms, genetic linkage, linkage learning, linkage learning genetic algorithm, sequential behavior, tightness time, convergence time

1 Introduction

Genetic algorithms (GAs) are powerful search techniques based on principles of evolution. They are now widely applied to solve problems in many fields. Most genetic algorithms employed in practice nowadays are simple genetic algorithms with fixed chromosomes and operators. Unable to learn linkage among genes, these traditional genetic algorithms suffer from the linkage problem (Goldberg, Korb, & Deb, 1989; Goldberg, Deb, & Thierens, 1993; Thierens, 1995). They require the users to possess prior domain knowledge of the problem so that the genes on chromosomes can be correctly arranged with respect to the chosen operators. When problem domain knowledge is available, the problem can be solved using traditional genetic algorithms with an appropriate chromosome representation. However, when that knowledge is not available, one way to handle the problem is to make the GA capable of adapting and learning linkage.

Harik (1997) took Holland's call for the evolution of tight linkage (Holland, 1975) and proposed the linkage learning genetic algorithm (LLGA). The LLGA uses the (*gene number*, *allele*) coding scheme with non-coding segments to permit genetic algorithms to learn tight linkage of building blocks through probabilistic expression (PE). While the LLGA performs much better on badly scaled problems than traditional simple GAs, it

[†]The work was done while the author was a graduate student in the Department of Computer Science at the University of Illinois at Urbana-Champaign and a member of Illinois Genetic Algorithms Laboratory.

does not work well on uniformly scaled problems. Therefore, we need to understand why the LLGA does not succeed in solving uniformly scaled problems and need to know how to design a better algorithm. This paper aims to gain better understanding of the LLGA. In particular, a convergence time model is constructed to explain why the LLGA needs exponentially growing time to solve uniformly scaled problems. In addition to the convergence time model, the sequential convergence behavior of the LLGA is also identified.

This paper starts with a brief survey of the existing genetic linkage learning techniques for genetic algorithms in section 2. Current linkage learning methods are classified into three categories according to the way they detect and process genetic linkage. Section 3 describes the essential components of the LLGA in detail as well as the difficulty faced by the LLGA. Then, in section 4, the sequential behavior of the LLGA is identified, the tightness time model for a single building block is introduced, and the connection between the sequential behavior and the tightness time model is developed. A convergence time model for the LLGA is constructed thereafter by integrating these results. Finally, the summary and conclusions of this paper are presented in section 5.

2 Genetic Linkage Learning Techniques

In the literature, a lot of efforts have been made to handle and process genetic linkage learning. This section briefly presents some of the existing linkage learning schemes for genetic algorithms. Readers who are interested in this topic should refer to other materials (Larrañaga & Lozano, 2001; Pelikan, Goldberg, & Lobo, 2002; Smith, 2002) for more information. Existing linkage learning techniques can be broadly classified according to the way they detect and process genetic linkage, into the following three categories proposed by Munetomo and Goldberg (1999b):

- perturbation-based methods;
- linkage adaptation techniques;
- probabilistic model builders.

Each category is described in the remainder of this section.

2.1 Perturbation-based Methods

Perturbation-based methods detect the genetic linkage between genes by perturbing the individual and observing the fitness difference caused by the perturbation. These methods consider the nonlinearity or epistasis detectable with perturbation as linkage and extract such information via sampling or enumerating. The *messy genetic algorithm* (mGA) (Goldberg, Korb, & Deb, 1989; Goldberg, Deb, & Korb, 1990) encoded genes as (*gene number*, *allele*) pairs, called *messy coding*, and adopted *messy operators* to cut and splice chromosomes. With a process consisting of heterogeneous phases and epoch-wise iterations, mGA was able to learn genetic linkage by using perturbation. To achieve a scalable computational speed, Goldberg, Deb, Kargupta, and Harik (1993) modified mGA and developed the *fast messy genetic algorithm* (fmGA), which can solve difficult problems in a sub-quadratic computational time by utilizing probabilistic complete enumeration, building block filtering, and extended thresholding. Kargupta (1996) proposed the *gene expression messy genetic algorithm* (gemGA), which utilized a genetic linkage identifying procedure, called the *transcription* operator, for constructing linkage groups. The name of gemGA is somewhat misleading, because messy coding and operators are not employed in gemGA.

In addition to developing complete genetic algorithms capable of learning linkage, Munetomo and Goldberg (1998) proposed the *linkage identification by nonlinearity check* (LINC), as a procedure for identifying linkage groups based on a perturbation methodology. LINC assumed that the linkage can be detected by checking the nonlinearity of the change of fitness values reflecting the perturbations in a pair of genes. When using LINC, each pair of genes is perturbed, and the corresponding fitness values are obtained. If the fitness change indicates the non-allowable nonlinearity for some pair of genes, those genes are considered linked. Moreover, Munetomo and Goldberg (1999a) proposed another linkage identifying procedure, the *linkage identification by non-monotonicity detection* (LIMD), which checked the violation of monotonicity conditions instead of nonlinearity. The equivalence of LIMD and LINC-AN was proved in their work. LINC was later extended to *linkage identification based on epistasis measures* (LIEM) (Munetomo, 2002a) by substituting the *epistasis measure* for LINC criterion to check the linkage strength among pairs of genes, and LIMD was extended to *linkage identification with epistasis measure considering monotonicity conditions* (LIEM^2) (Munetomo, 2002b) with the same extension.

2.2 Linkage Adaptation Techniques

Linkage adaptation techniques employ specifically designed representations, operators, and mechanisms for adapting genetic linkage along with the evolutionary process. Compared to the perturbation-based methods or the probabilistic model builders, which will be introduced in the next section, linkage adaptation techniques are closer to the biological metaphor of evolutionary computation because of their representations, operators, and mechanisms. Schaffer and Morishima (1987) added the *punctuation marks*, which were extra bits attached to each gene, onto the chromosome representation. The punctuation marks indicate whether a position on the chromosome is a crossover point in order to adapt the recombination operation. They are also transferred with the allele bits to which they are attached during crossover events such that the adaptation of the punctuation marks are achieved in the same evolutionary process as well. With a similar coding scheme, Levenick (1995) proposed a generic *endogenous crossover control*, which utilizes the *metabits* to implement the *differential crossover probability*. The difference between metabits and punctuation marks is that metabits indicate the probability for a position to be chosen as a crossover point, while punctuation marks are used to determined whether or not a position is a crossover point. Following his previous conclusion of using *introns* in the genotype (Levenick, 1991), Levenick proposed to use metabits for controlling the probability of crossover positions to simulate the effect of inserting introns into the chromosome.

Along this line, Smith and Fogarty (1996) proposed the *linkage evolving genetic operator* (LEGO) for adapting the recombination strategy via evolution of genetic linkage. In LEGO, two boolean flags are attached to each gene for indicating whether a gene is linked to its "left" neighbor or to its "right" neighbor on the chromosome, respectively. If the mutual flags of two adjacent genes are both true, these two genes are considered linked; otherwise, they are not linked. Consecutive linked genes on the chromosome are considered as a building block, and the whole population is treated as a pool of building blocks. Offspring are then created via iterations of competitions held among building blocks starting at the same position. The last existing linkage learning method in this category is the *linkage learning genetic algorithm* (LLGA), which uses a special chromosome representation and an expression mechanism for learning genetic linkage. As the subject topic of this study, the LLGA will be introduced in detail later.

2.3 Probabilistic Model Builders

Probabilistic model builders learn genetic linkage via building probabilistic models based on the current population and generating new individuals according to the obtained probabilistic models. The relationship, which can be considered as genetic linkage, between genes or variables is demonstrated in the form of probabilistic models. Compared to the linkage adaptation techniques, these model builders are computation-oriented instead of biology-oriented, and many of them currently have the best performance in this field. In the early development stage of probabilistic model builders, genes are assumed independent of each other. Therefore, these precursors to the model builders are not linkage learning techniques but are important to this research direction. For example, the *population-based incremental learning* (PBIL) algorithm (Baluja, 1994) which replaces the population with a probability vector to represent the probability distribution of each allele is one of the first attempts to build probabilistic models in this field. Offspring are created based on the probability vector and selected with truncation selection. The probability vector is updated toward those selected individuals with a specified learning factor. With a different probability vector updating rule, the *univariate marginal distribution algorithm* (UMDA) (Mühlenbein & Paaß, 1996) processes the whole population to calculate the frequency of each allele and uses the results as the next probability vector. According to the current probability vector, the *compact genetic algorithm* (cGA) (Harik, Lobo, & Goldberg, 1999) generates two individuals and lets them compete with each other. The probability vector is updated toward the winner with a learning factor of the inverse of the virtual population size.

By using pairwise models or complex probabilistic models involving more than two variables, the probabilistic model builders can be considered as linkage learning techniques in the context of this paper. For example, the *mutual-information-maximizing input clustering* (MIMIC) algorithm (Bonet, Isbell, & Viola, 1996) uses a chain distribution to maximize the mutual information of adjacent variables. For efficiency, MIMIC adopts a greedy search method for constructing the probabilistic model. The *combining optimizers with mutual information trees* (COMIT) method (Baluja & Davies, 1997) builds a dependency tree to express the relations between genes. A polynomial maximal branching algorithm (Edmonds, 1967) that can obtain globally optimal models is utilized. The *bivariate marginal distribution algorithm* (BMDA) (Pelikan & Mühlenbein, 1999) uses a set of dependency trees to model the relations instead of a single tree. Pearson's chi-square test (Marascuilo & McSweeney, 1977) is used in BMDA for determining the linkage between variables.

For more complex models, Harik (1999) proposed the *extended compact genetic algorithm* (ECGA) to capture the relations involving one or more genes by building the *marginal product model* based on a minimum description length (MDL) metric. In ECGA, one or more variables are included in one joint probability distribution in order to minimize the MDL metric of the overall model. Therefore, genes are divided into several linkage groups after building the marginal product model. The *Bayesian optimization algorithm* (BOA) (Pelikan, Goldberg, & Cantú-Paz, 2000) utilizes the Bayesian network as its underlying model to learn probabilistic dependency among genes. To determine whether one model is better than another, the Bayesian-Dirichlet (BD) scoring metric (Heckerman, Geiger, & Chickering, 1994) is employed, and to search for good models, a greedy search method is currently used due to efficiency. However, other metrics and search algorithms can be adopted in BOA as well. Finally, by the nature of probability distributions, the paradigm of probabilistic model builders can be extended to handle real-valued variables, such as the *factorized distribution algorithm* (FDA) (Mühlenbein &

Mahnig, 1999), the IDEA framework (Bosman & Thierens, 2001), and the like. For more detailed information of related algorithms in the continuous domain, readers should consult other complete and specific surveys of probabilistic model builders (Larrañaga & Lozano, 2001; Pelikan, Goldberg, & Lobo, 2002).

3 The Linkage Learning Genetic Algorithm

In this section, key elements of the linkage learning genetic algorithm (LLGA) proposed by Harik (1997) are reviewed. The LLGA is capable of learning genetic linkage in the evolutionary process without the help of extra measurements and techniques. A modified version of the LLGA working with *promoters* proposed in our previous work (Chen & Goldberg, 2002) is used in this study and described in this section. Interested readers may consult other materials (Harik, 1997; Harik & Goldberg, 2000; Chen & Goldberg, 2002) for more detailed background information.

The following topics are presented:

- chromosome representation;
- the exchange crossover operator;
- the linkage learning mechanisms.

In addition to introducing the LLGA, this section also presents the difficulty faced by the LLGA and the purpose of the present work.

3.1 Chromosome Representation

The LLGA chromosome representation is composed of

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

Each of these elements is described in what follows.

The LLGA chromosome consists of moveable genes encoded as (*gene number*, *allele*) pairs and is considered as a circle. The genes in the LLGA are allowed to reside anywhere in any order on the chromosome, while those in a traditional GA are unmovable and fixed at specified loci. To create a genotypic structure capable of expressing linkage, non-coding segments are included in the LLGA chromosome. Non-coding segments have been widely used and studied in genetic algorithms (Levenick, 1991; Wu, Lindsay, & Smith, 1994; Wineberg & Oppacher, 1996) and genetic programming (Nordin, Francone, & Banzhaf, 1996; Andre & Teller, 1996; Iba & Terao, 2000). In the LLGA, non-coding segments act as non-functional elements on the chromosome. Unlike non-coding segments used elsewhere, the LLGA's non-coding segments are moveable like the functional genes. By using non-coding segments, linkage of building blocks can be more accurately expressed. For example, the linkage learning methods, such as punctuation marks, LEGO, LINC/LIMD, and the messy GA family, use only binary linkage to specify the relationship between genes. In these methods, genes are either linked or not linked, and all linked genes form a building block or a linkage group. Compared to these methods, the chromosome design of the LLGA can provide many different levels

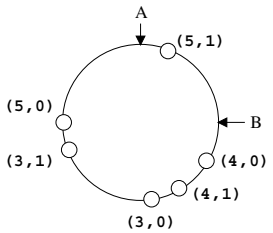


Figure 1: Different points of interpretation might interpret a PE chromosome as different solutions.

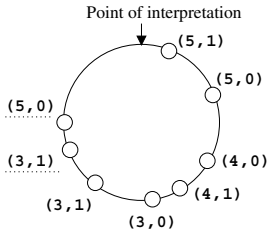


Figure 2: An example of EPE-2 chromosomes. Each gene can have up to 2 complement genes.

of building-block linkage in the form of distances of genes belonging to the same building block on the chromosome. As a consequence, genetic linkage of building blocks can be accurately expressed in the LLGA.

Moreover, probabilistic expression (PE) was proposed to preserve diversity at the building-block level. A PE chromosome contains all possible alleles for each gene. For the purpose of fitness 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 chromosome. As a consequence, a chromosome represents a probability distribution over the range of possible solutions instead of a single solution.

When different points of interpretation are selected, a PE chromosome might be interpreted as different solutions. Figure 1 shows genes 3, 4, and 5 of a PE chromosome composed of 6 genes, of which genes 0, 1, and 2 are not shown in the figure for simplicity. If point A is the point of interpretation, the part of these 3 genes of 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 since genes 0, 1, and 2 are not shown here, "*" is used to represent the unknown alleles. Moreover, if point B is the point of interpretation, the part of the chromosome will then 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 a shadowed gene, we can generalize PE to let a chromosome contain more than one copy of a shadowed gene. Therefore, the extended probabilistic expression (EPE) can be defined with a parameter k that permits a chromosome to contain up to k copies of a shadowed gene. Figure 2 shows an example of an EPE-2 chromosome.

Chen and Goldberg (2002) proposed the use of *promoters*, which were called *start expression genes*, in the LLGA to handle separation inadequacy and to improve nucleation potential so that the building blocks can be more easily separated and tightened. Promoters are special non-functional elements on the chromosome. While in the LLGA without promoters, all genes and non-coding segments can be the points of interpretation of the child after crossover, only promoters can be the points of interpretation in the LLGA with promoters. Furthermore, the LLGA with promoters uses PE instead of EPE to encode its chromosome.

3.2 Exchange Crossover

In addition to PE and EPE, the exchange crossover operator is another key mechanism for the LLGA to learn genetic linkage. Exchange crossover is defined on a pair of 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 onto the recipient. The grafting point is the point of interpretation of the generated offspring. Starting from the point of interpretation, redundant genetic material caused by injection is removed right after crossover to ensure the validity of the offspring.

In the LLGA with promoters, although the grafting point can still be any genes or non-coding segments, the point of interpretation of the offspring is no longer the grafting point. Instead, the new point of interpretation is the nearest promoter *before* the grafting point on the chromosome. After the grafting point is randomly chosen, we look for the first promoter in front of the grafting point and make it the point of interpretation for the offspring. The genetic material is then transferred in the following order: (1) the segment between the promoter and the grafting point, (2) the segment chosen from the donor, and (3) the rest of the recipient.

3.3 Linkage Learning Mechanisms

With the integration of PE and the exchange crossover operator, the LLGA is capable of solving difficult problems without prior knowledge of good linkage. Traditional GAs 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 LLGA, two key mechanisms of linkage learning: *linkage skew* and *linkage shift* have been identified and analyzed (Harik & Goldberg, 1996). Linkage skew occurs when an optimal building block is transferred from the donor to the recipient. Linkage shift occurs when an optimal building block resides in the recipient and survives an injection. Both linkage skew and linkage shift make the building block's linkage tighter. With these two mechanisms, the linkage of building blocks can evolve, and tightly linked building blocks are formed during the process.

3.3.1 Quantifying linkage

For studying the linkage learning process, a proposed definition for quantifying linkage (Harik & Goldberg, 1996) is adopted, which is the sum of the square of the inter-gene distances of a building block, considering the chromosome to be a circle of circumference 1. Figure 3 shows an example for calculating the linkage of a three-gene building block. The definition is appropriate in that linkage in such a definition specifies a measure directly proportional to the probability for a building block to be preserved under the exchange crossover operator.

3.3.2 Linkage skew

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 an inferior one does. The effect of linkage skew was found to make linkage distributions move toward higher linkages by eliminating less fit individuals. Linkage skew does not make the linkage of a building block of any particular individual tighter. Instead, it drives the whole linkage distribution to a higher state.

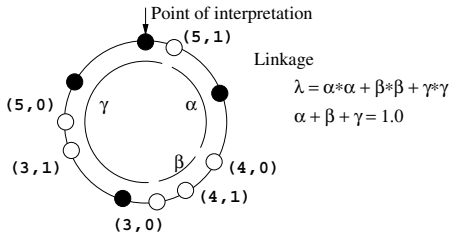


Figure 3: Calculation for the linkage of a three-gene building block. The linkage is defined as the sum of the square of the inter-gene distances of a building block, considering the chromosome to be a circle of circumference 1. The linkage is calculated according to the expressed genes, and in this example, the distance between the three genes are α, β, and γ. Thus, by definition, $\alpha+\beta+\gamma = 1$, and the linkage $\lambda = \alpha*\alpha+\beta*\beta+\gamma*\gamma$.

3.3.3 Linkage shift

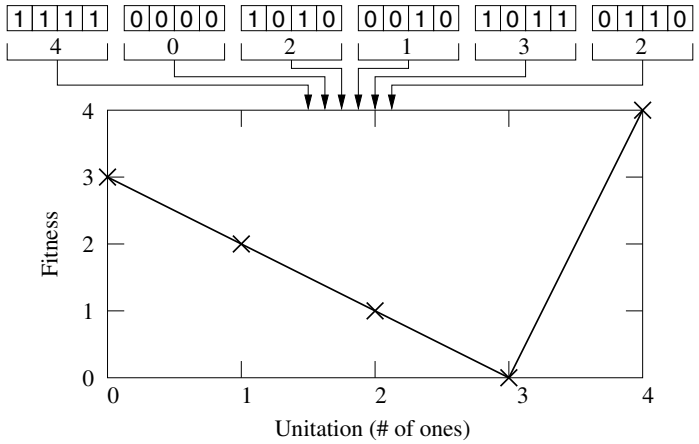
Linkage shift is the second linkage learning mechanism. It occurs when an optimal building block resides in the recipient and survives a crossover event. For the optimal building block to survive, there cannot be any gene contributing to a transferred deceptive building block. Linkage shift gets a building block tighter in an individual with deletion of duplicate genetic material caused by injection of exchange crossover.

3.4 Difficulty Faced by the LLGA

We study the LLGA on problems containing multiple building blocks in two forms—the uniformly scaled problem and the exponentially scaled problem. When the problem is composed of only a short, single building block, the mechanisms of the LLGA work as expected. However, when the problem consists of multiple building blocks, the success or failure depends on how these building blocks are scaled. As reported by Harik (1997), when the building blocks of a problem are exponentially scaled, the LLGA can solve the problem in a linear time function of the number of building blocks. However, when the building blocks are uniformly scaled, the LLGA either needs a population size that grows exponentially with the problem size based on Harik's results or takes exponential time to converge according to our recent study. Therefore, the LLGA takes linear or exponential time depending on the scaling of the building blocks of the problem. It is currently unknown why the behavior of the LLGA seems inconsistent when solving multiple building blocks of different scalings. This research tries to understand the difficulty faced by the LLGA and tries to provide explanations to the seemingly inconsistent behavior.

4 Convergence Time for the LLGA

In this section, we develop the convergence time model for the LLGA step by step. We start by describing the settings of all the experiments for observing the behavior of the LLGA and verifying the theoretical results. The first step is to empirically identify the sequential behavior of the LLGA from the macro view. The next step is to introduce the tightness time model for a single building block based on the two linkage learning mechanisms from the micro view. Finally, after extending the tightness time model to establish the connection between the models from the macro view and the micro view, a convergence time model for the LLGA is constructed by integrating all these models.



Uniformly Scaled: 4 x 1 + 3 x 1 + 1 x 1 + 2 x 1 + 0 x 1 + 1 x 1 = **11**

Exponentially Scaled: 4 x 1 + 3 x 2 + 1 x 4 + 2 x 8 + 0 x 16 + 1 x 32 = **106**

Figure 4: 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.

4.1 Experiment Settings

The experiment settings for all the experiments in this study is described in this section. First, we introduce the test function for understanding the LLGA, and then, the algorithm configuration and parameters, including the selection operator and the population size, are presented.

4.1.1 Test function

In this study, the order-4 trap function (Ackley, 1987; Deb & Goldberg, 1993) is adopted as a subproblem for constructing all the test problems. 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 domain into two sets. One of them leads to a global optimum, and the other leads to a local optimum. In particular, Figure 4 shows the order-4 trap function used in this study. Note that for this 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), the order-4 trap is fully deceptive because $r = 0.75 \geq r_{min} = (k-1)/(2k-3) = 0.6$, where $k = 4$ for order-4 trap functions.

Moreover, as also shown in Figure 4, to construct larger test problems, we mainly consider two scaling methods to combine those elementary subproblems, which are also considered as building blocks in our framework. One way is to scale them uniformly, and the other exponentially. These 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. In the remainder of this paper, the scaling factor of the exponentially scaled building blocks is 5.0.

Table 1: Parameters for population sizing based on the gambler's ruin model.

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

4.1.2 General parameters

In this paper, tournament selection without replacement is used. Harik (1997) proposed using a high selection pressure to drive both search and linkage learning processes. However, according to Chen and Goldberg (2002), the selection pressure required is not necessarily as high as Harik proposed. As a consequence, we set the tournament size to 3 throughout this paper.

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 size. We employ the gambler's ruin model (Harik, Cantuú-Paz, Goldberg, & Miller, 1999) for population sizing, which can be approximated with the following formula:

$$\text{population size } n = -2^{k-1}\ln(\alpha)\frac{\sigma_{bb}\sqrt{\pi(m-1)}}{d},$$

where k is the length of a single building block, α is the failure probability we are willing to tolerate, σ_{bb} is the standard deviation of the fitness of a building block, m is the total number of building blocks, and d is the signal, which is the fitness difference between the global optimum and the local optimum of the building block.

Since the tournament size is set to 3, the signal is adjusted with the equation (Harik, Cantú-Paz, Goldberg, & Miller, 1997) $d' = d + \Phi^{-1}(1/s)\sigma_{bb}$, where s is the tournament size, and $\Phi^{-1}(1/s)$ is the ordinate of a unit normal distribution where the CDF equals $1/s$. Table 1 summarizes the parameters for calculating population size according to the order-4 trap functions. Table 2 lists all population sizes used for the test problems composed of different numbers of building blocks. Note that the gambler's ruin model for population-sizing was developed based on the model of random walk and verified with the problems composed of uniformly scaled building blocks. However, in this paper, we employ this population-sizing model for both uniformly scaled and exponentially scaled building blocks. The rationale is that the population sizes for uniformly scaled building blocks can be considered an upper bound of that for exponentially scaled building blocks. According to the temporal-salience structure (Thierens,

Table 2: Population sizes for the test problems composed of different BB numbers.

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

Goldberg, & Pereira, 1998) of the exponentially scaled problems, GAs work on the most salient part of the chromosome, then on the next most salient part, and so on. Thus, at a given time, the population size required to solve an exponentially scaled problem should be smaller than that required to solve a uniformly scaled problem of the same number of building blocks. Therefore, for observation and comparison purposes, we use this population-sizing model for both kinds of problems in this work.

Other parameters are set as follows. The crossover rate is 1.0 such that the crossover event always happens. The maximum number of generation is 10000. The number of promoters is set to $2m$, where m is the number of building blocks. 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 study are averaged over 50 independent runs of experiments unless mentioned otherwise.

4.2 Macro View: Sequential Behavior

As mentioned in section 3.4, the LLGA seems to have an inconsistent behavior when solving the problem having multiple building blocks with different scaling. Therefore, the first step in this study is to identify consistent underlying working behavior if it exists. Experiments for the exponentially scaled problem and the uniformly scaled problem are conducted respectively for observing the working behavior of the LLGA. As expected, the empirical results reveal a consistent, sequential behavior.

4.2.1 Exponentially scaled building blocks

Harik (1997) had identified that the exponentially scaled problems are solved by the LLGA sequentially. The building blocks get tightly linked and are solved one by one. In the following, we conduct our own experiments to verify this sequential behavior.

Time to converge First, we use the LLGA to solve exponentially scaled problems composed of different numbers of building blocks. We vary the number of building blocks from 4 to 15 and record the number of generations when the following condition is true for 20 consecutive generations: the difference between the number of building blocks solved by the generational best individual and the average number of building blocks solved by all individuals is less than 0.0001. The result is shown in Figure 5. As we can see, the time for the LLGA to converge when solving exponentially scaled problems increases linearly as the number of building blocks increases.

Building block propagation Next 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 the LLGA run and observe the convergence time under different settings and configurations. To be exact, Table 3 lists all the experiments conducted for this purpose. Figure 6 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.

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

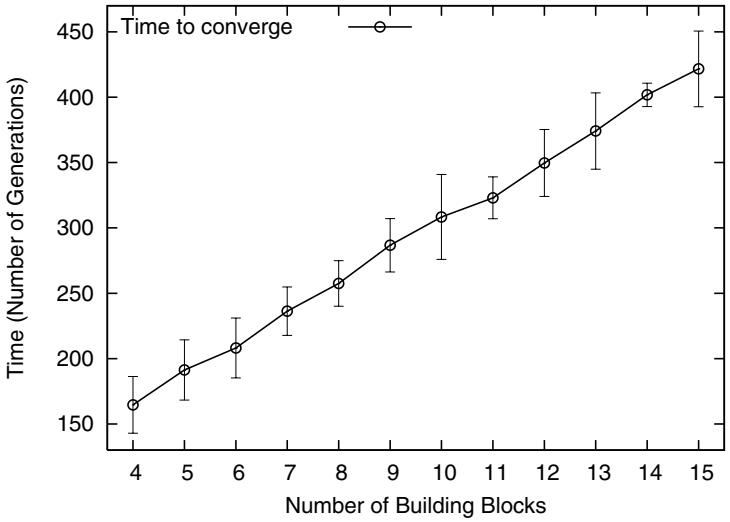


Figure 5: Time for the LLGA to converge when solving exponentially scaled building blocks. The convergence time grows linearly with the number of building blocks.

should remain constant. Thus, for every g generations, one building block is solved and effectively disappears from the scope of the LLGA. Note that the "first building block" here does not necessarily mean the building block of the highest number, the salient building block, or any particular building block. The "first building block" refers to the building block achieving certain linkage first among all building blocks. The results are shown in Figure 7. The time for the first building block to converge seems to hold constant when solving the problem consisting of different numbers of building blocks.

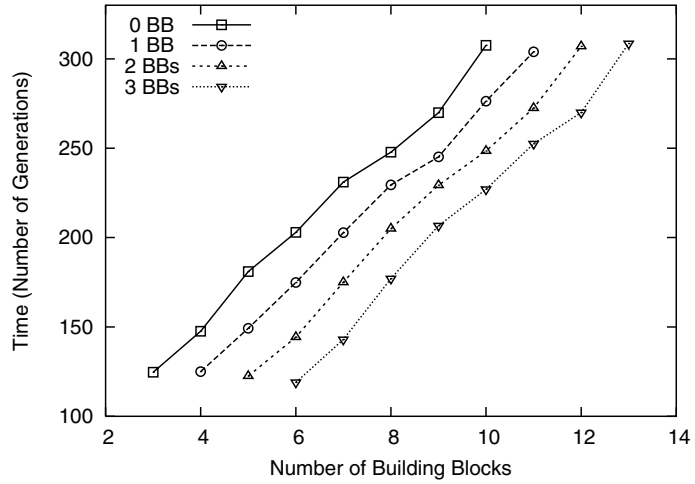
4.2.2 Uniformly scaled building blocks

After observing the LLGA's behavior on solving exponentially scaled building blocks, we now turn to uniformly scaled building blocks to observe the experimental results.

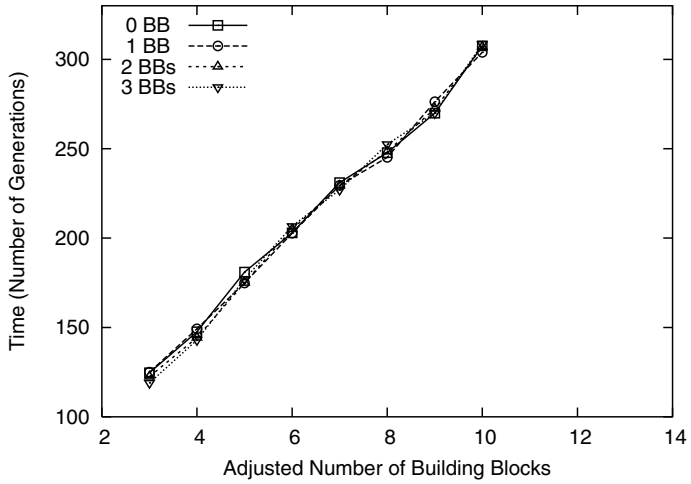
Time to converge As we did in section 4.2.1, we first use the LLGA to solve uniformly scaled problems. We vary the number of building blocks from 4 to 15 and record the number of generations when convergence. Figure 8 shows the results of the experiments. The convergence time to solve uniformly scaled problems grows exponentially with the number of building blocks. If we consider the overall complexity, these results do not contradict those reported by Harik (1997), because the overall complexity grows exponentially in both cases.

Table 3: Experiments for observing the building block propagation.

Number of pre-solved BBs	Range of numbers of BBs
0	$3, 4, \cdots, 10$
1	$4, 7, \cdots, 11$
2	$5, 8, \cdots, 12$
3	$6, 9, \cdots, 13$



(a)



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

Figure 6: Convergence time for solving exponentially scaled building blocks with some building blocks pre-solved. It shows that the time needed to solve $m+j$ building blocks when j building blocks are solved equals the time needed to solve m building blocks.

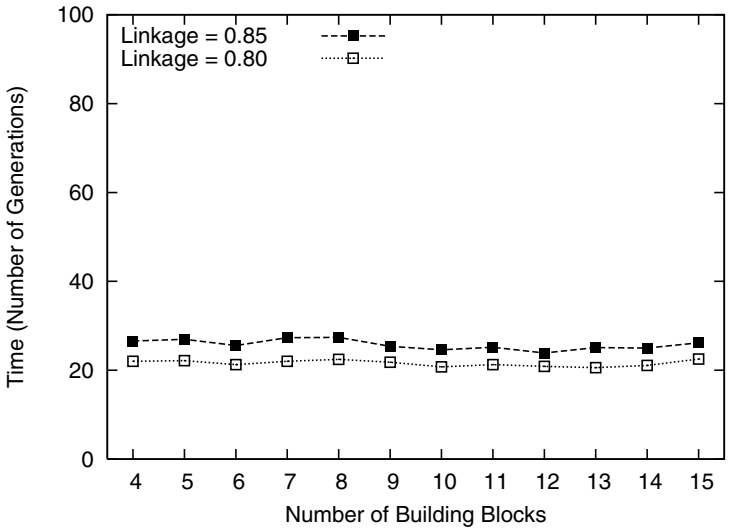


Figure 7: Time to tighten the first building block holds constant for different numbers of building blocks when solving exponentially scaled problems.

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 3 as we did in section 4.2.1. Figure 9 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 relatively more important role than they do in solving exponentially problems. However, the results of this experiment show that building block propagation also works well when solving uniformly scaled building blocks.

Time to tighten the first building block According to the experimental results from previous experiments, we might expect that for uniformly scaled building blocks, the time for the first building block to be tightened grows exponentially. Because building block propagation also works for uniformly scaled problems, 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 10 shows the time for the LLGA to tighten the first building block. 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.

These results not only ensure the building block propagation but also imply that if there are m unsolved building blocks, when a building block is solved, the situation or configuration is equivalent to when 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 LLGA.

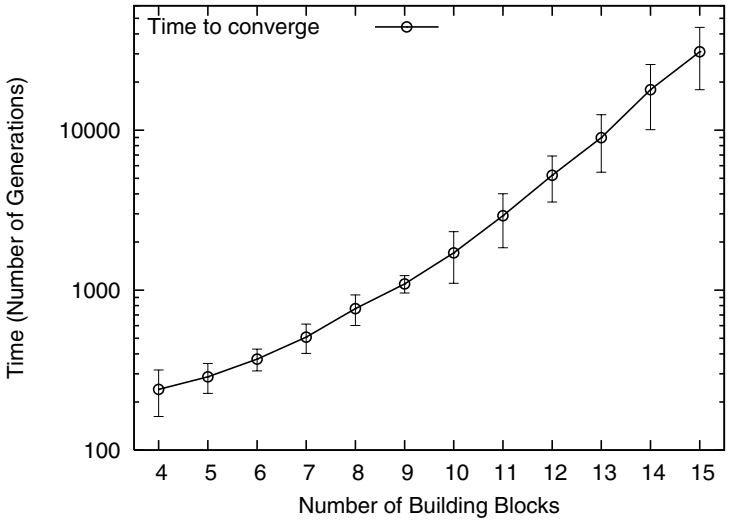


Figure 8: Time for the LLGA to converge when solving problems of uniformly scaled building blocks. The convergence time grows exponentially with the number of building blocks. Straight lines on a semi-log scale are indicative of exponential growth.

4.2.3 Sequential behavior

Based on the experimental results obtained in the previous sections, we propose a simple model, called the *first-building-block model*, to describe the sequential behavior of the LLGA. 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 of all, we define the function t_{fbb} as

$$t_{\text{fbb}}(m) = \text{time to tighten the first building block in a problem consisting of } m \text{ building blocks} \tag{1}$$

and the function t_{c} for the convergence time as

$$t_{\text{c}}(m) = \text{convergence time for solving a problem consisting of } m \text{ building blocks.} \tag{2}$$

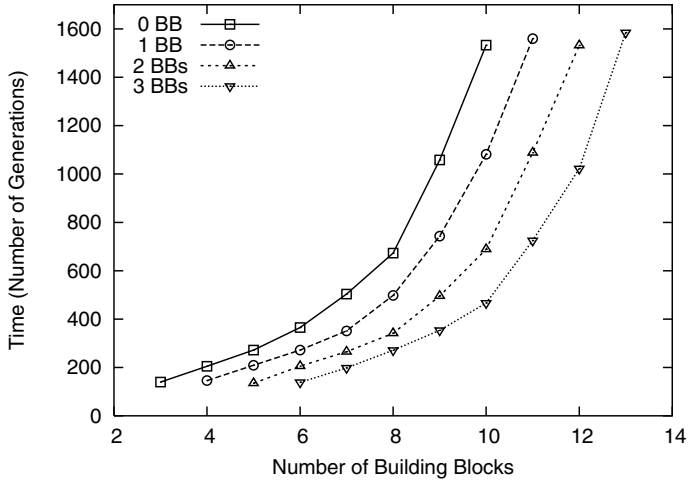
By our assumption of the first-building-block model, we can express t_{c} as

$$t_{\text{c}}(m) = \sum_{i=1}^{m} t_{\text{fbb}}(i) + t_{\text{c}0} \ , \tag{3}$$

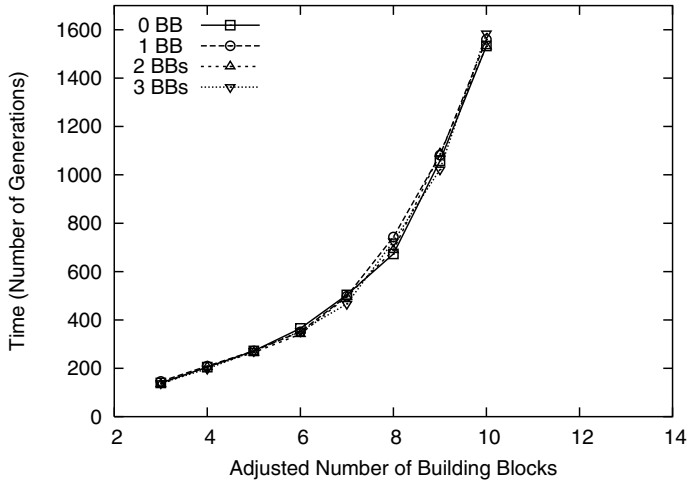
where $t_{\text{c}0}$ is a constant. The sequential behavior is therefore established through the first-building-block model. By rewriting t_{c} as

$$t_{\text{c}}(m) = \sum_{i=i_0+1}^{m} t_{\text{fbb}}(i) + t_{\text{c}}(i_0) \ , \tag{4}$$

where i_0 is the least number of building blocks of the available experimental results for convergence time, the model can be empirically verified with the obtained experimental results which are shown in Figures 11 and 12.



(a)



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

Figure 9: Convergence time for solving uniformly scaled building blocks with some building blocks pre-solved. It shows that the time needed to solve $m + j$ building blocks when j building blocks are solved equals the time needed to solve m building blocks.

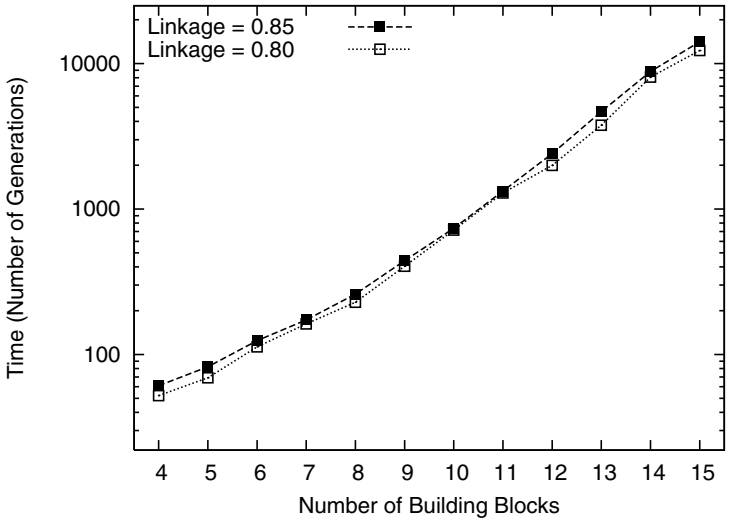


Figure 10: Time to tighten the first building block grows exponentially with the number of building blocks when solving uniformly scaled problems.

4.3 Micro View: Tightness Time

The next step toward the convergence time model for the whole linkage learning process is to derive the *tightness time*, the linkage learning time for a single building block to achieve a specified linkage. Based on the models for linkage skew and linkage shift, Chen and Goldberg (2003) proposed a tightness time model for a single building block obtained by extending and integrating the two fundamental models as

$$t_\ell(\lambda) = \frac{\log(1-\lambda) - \log(1-\overline{\Lambda_0})}{c_s \log(1-c)} , \tag{5}$$

where $t_\ell(\lambda)$ is the tightness time for a given linkage λ, $\overline{\Lambda_0}$ is the mean of the initial linkage distribution, $c = 2/(k+2)(k+3)$, k is the order of the building block, and $c_s \approx 2$ is determined empirically.

Furthermore, given the initial linkage distribution, $\overline{\Lambda_0}$ remains constant during the whole process. For simplicity, we can define $\epsilon = 1 - \lambda$ and $\overline{\epsilon_0} = 1 - \overline{\Lambda_0}$. Also, $c \approx \frac{2}{k^2}$ when $k \to \infty$. Hence, Equation (5) can be rewritten as a function of ϵ as

$$t'_\ell(\epsilon) = \frac{k^2}{2c_s} \log \frac{\epsilon}{\overline{\epsilon_0}} . \tag{6}$$

Equation (6) 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. In addition, tightness time is proportional to the logarithm of the desired linkage.

4.4 Convergence Time

After identifying the sequential behavior of the LLGA in the 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

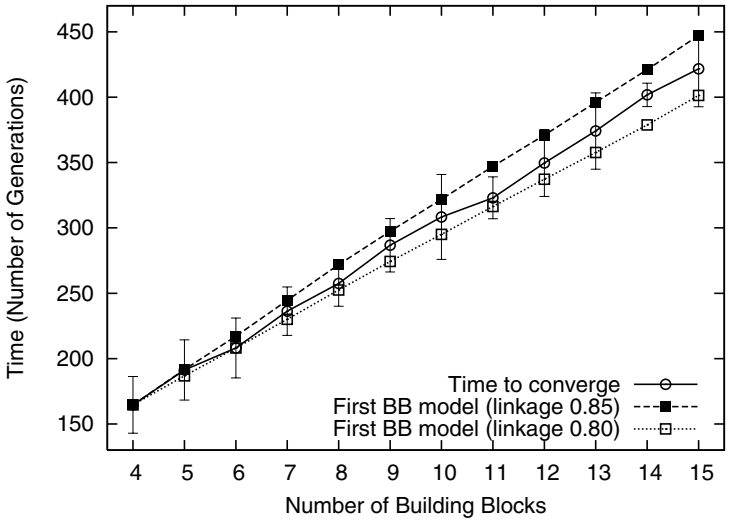


Figure 11: For exponentially scaled building blocks, the first-building-block model agrees with the experimental results.

how multiple building blocks affect the tightness time for the first building block when the building blocks are uniformly scaled. Therefore, we will now identify the effect of multiple building blocks on the tightness time for a single building block and then construct the convergence time model for the LLGA by integrating the models from the macro view, the micro view, and the connection in between.

4.4.1 From one building block to m building blocks

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

Genetic material from the donor First, we consider the genetic material cut from the donor during a crossover event. When the exchange crossover operates, a donor and a recipient are selected from the population. Based on the linkage learning mechanisms, which were identified under the condition that there is only one building block in the problem to be solved, we assume that (1) a segment from the donor containing only one complete building block (and other incomplete building blocks) contributes linkage learning and (2) the m building blocks are uniformly distributed in individuals in the middle linkage learning process. On average, there are $m/2$ building blocks transferred from the donor to the recipient. There are $\binom{m}{m/2}$ possible conditions to choose $m/2$ out of m building blocks. Therefore, the probability for the donor segment containing only one complete building block out of m building blocks is $\binom{m}{1}/\binom{m}{m/2}$.

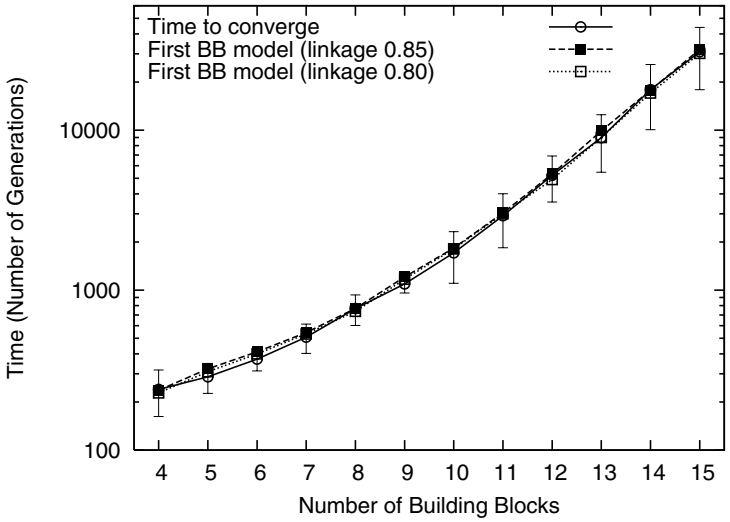


Figure 12: For uniformly scaled building blocks, the first-building-block model agrees with the experimental results.

Genetic material of the recipient In addition to the genetic material from the donor, $p_{\ell\ell}(m)$ also depends on the genetic material of the recipient. If the building block in question is disrupted by the grafting point, there is no linkage learning event. Based on the calculation of random linkage proposed by Harik and Goldberg (1996), the probability for a building block of order k to reside on one of the two segments is described as $2/(k+1)$. 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.

Tightness time for m uniformly scaled building blocks As a result, the probability of a linkage learning event is

$$p_{\ell\ell}(m) = \frac{2}{k+1}\left[\binom{m}{1} \Big/ \binom{m}{\frac{m}{2}}\right] . \tag{7}$$

By combining the tightness time model for a single building block and the probability of a linkage learning event, we get the tightness time model for m uniformly scaled building blocks as

$$\begin{aligned} t(m,\epsilon) &= t_\ell(\epsilon)\frac{1}{p_{\ell\ell}(m)} ; \\ &= t_\ell(\epsilon)\frac{k+1}{2}\left[\binom{m}{\frac{m}{2}} \Big/ \binom{m}{1}\right] . \end{aligned} \tag{8}$$

By using the Stirling approximation $m! \approx (m/e)^m\sqrt{2\pi m}$, we obtain

$$\begin{aligned} t(m,\epsilon) &= t_\ell(\epsilon)\left(\frac{k+1}{2}\right)\frac{\sqrt{2}}{\sqrt{\pi}}\frac{2^m}{m\sqrt{m}} ; \\ &= t_\ell(\epsilon)\,\frac{k+1}{\sqrt{2\pi}}\,\frac{2^m}{m\sqrt{m}} . \end{aligned} \tag{9}$$

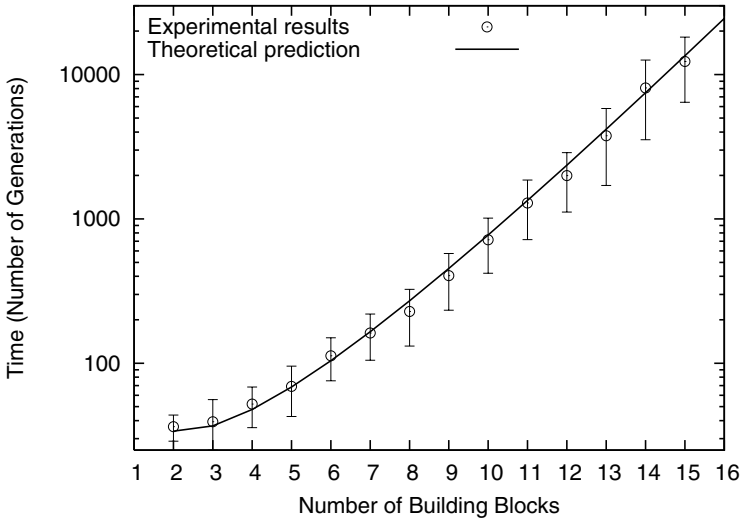


Figure 13: Tightness time for the first building block achieving $\lambda = 0.80$ among multiple uniformly scaled building blocks.

Figure 13 shows that the experimental results agree with the tightness time model for $\lambda = 0.80$ when solving multiple uniformly scaled building blocks.

4.4.2 Model for convergence time

Finally, by integrating the results from the sequential behavior (Equation (3)), the tightness time model (Equation (6)), and the connection in between (Equation (9)), we can obtain the LLGA convergence time model for some desired linkage as

$$\begin{aligned} t_{\mathrm{C}}(m,\epsilon) &= \sum_{i=1}^{m} t(i,\epsilon) + t_{\mathrm{C}0} \ ; \\ &= \sum_{i=1}^{m} \left(t_{\ell}(\epsilon)\, \frac{k+1}{\sqrt{2\pi}}\, \frac{2^i}{i\sqrt{i}} \right) + t_{\mathrm{C}0} \ ; \\ &= \sum_{i=1}^{m} \left(\left(\frac{k^2}{2c_s} \log \frac{\epsilon}{\epsilon_0} \right) \frac{k+1}{\sqrt{2\pi}}\, \frac{2^i}{i\sqrt{i}} \right) + t_{\mathrm{C}0} \ ; \\ &= \left(\frac{k^2(k+1)}{2c_s\sqrt{2\pi}} \log \frac{\epsilon}{\epsilon_0} \right) \sum_{i=1}^{m} \frac{2^i}{i\sqrt{i}} + t_{\mathrm{C}0} \ , \end{aligned} \tag{10}$$

where c_s and $t_{\mathrm{C}0}$ are constants, m is the number of uniformly scaled building blocks, k is the order of the single building block, $\epsilon = 1 - \lambda$, and λ is the desired linkage. As shown in Figure 14, the results agree with the proposed convergence time model.

5 Summary and Conclusions

We started with a brief survey of the existing genetic linkage learning techniques for genetic algorithms. These methods were classified into three groups: (1) perturbation-based methods, (2) linkage adaptation techniques, and (3) probabilistic model builders,

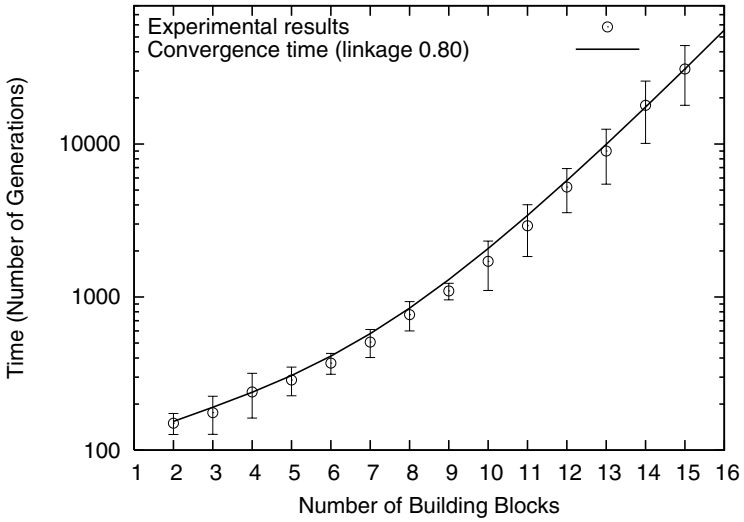


Figure 14: Convergence time for the LLGA when $\lambda = 0.80$ on the problem composed of multiple uniformly scaled building blocks.

according the way these techniques detect and process genetic linkage. Then, key elements of the LLGA were introduced, including chromosome representation, exchange crossover, and the linkage learning mechanisms. The sequential behavior was observed when the LLGA were solving both exponentially scaled problems and uniformly scaled problems. The first-building-block model was proposed accordingly from the macro view based on the sequential behavior. By extending and integrating the linkage learning mechanisms, the tightness time model for a single building block was developed from the micro view. Establishing the connection between these models, a convergence time model for the LLGA was constructed and empirically verified. The proposed convergence time model explains why the LLGA requires exponential time to solve uniformly scaled problems and gives us an insight into how the LLGA operates.

In this paper, we identified a consistent, sequential behavior of the LLGA. It was previously believed that when solving a uniformly scaled problem, the LLGA works on all building blocks simultaneously, while when solving an exponentially scaled problem, the LLGA works on the salient building block, the second most salient building block, and so on. By identifying the sequential behavior of the LLGA, we gain better understanding about how the LLGA works—one building block at a time. The difference is that for exponentially scaled building blocks, the salient building block is tightened first with a very high probability due to the selectional advantage, but for uniformly scaled building blocks, each building block has the same probability of being tightened first. Recognizing the sequential behavior might shed light on a better design of the LLGA to perform well on both exponentially and uniformly scaled problems.

The proposed convergence time model indicates that the time required by the LLGA to solve a uniformly scaled problem grows exponentially in terms of the number of building blocks. The exponential growth of time, according to the analysis, mainly comes from the competition among the building blocks of equal salience. The decrease of the probability of crossover events increases the linkage learning time correspondingly. Therefore, based on the model, the possible ways to improve the LLGA's per-

formance on uniformly scaled problems include (1) effectively reducing the number of building blocks simultaneously processed by the LLGA and (2) employing certain mechanisms or procedures to make the LLGA process building blocks sequentially. These two ways may be promising directions to improve the LLGA's performance.

More work along this line still needs to be done including both theoretical and practical aspects. For the theoretical aspect, more exact and sophisticated models for the linkage learning process are needed for understanding the nature of genetic linkage learning. On the other hand, for the practical aspect, new representations, mechanisms, or procedures should be developed and tested for improving the performance of the LLGA as discussed.

Acknowledgments

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants F49620-00-0163 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 U.S. Government.

References

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

Andre, D., & Teller, A. (1996). A study in program response and the negative effects of introns in genetic programming. In *Proceedings of the First Annual Conference on Genetic Programming (GP 96)* (pp. 12–20). MIT Press.

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

Baluja, S., & Davies, S. (1997). Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 30–38). Morgan Kaufmann Publishers, Inc.

Bonet, J. S. D., Isbell, C., & Viola, P. (1996). MIMIC: Finding optima by estimating probability densities. *Advances in Neural Information Processing Systems*, *9*, 424–430.

Bosman, P. A., & Thierens, D. (2001). Advancing continuous IDEAs with mixture distributions and factorization selection metrics. In *Proceedings of the Optimization by Building and Using Probabilistic Models Workshop at GECCO-2001* (pp. 208–202). GECCO.

Chen, Y.-p., & Goldberg, D. E. (2002). Introducing start expression genes to the linkage learning genetic algorithm. *Lecture Notes in Computer Science (LNCS)*, *2439*, 351–360. (Also IlliGAL Report No. 2002007).

Chen, Y.-p., & Goldberg, D. E. (2003). Tightness time for the linkage learning genetic algorithm. *Lecture Notes in Computer Science*, *2723*, 837–849. (Also IlliGAL Report No. 2003002).

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

Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards*, *71B*, 233–240.

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., Deb, K., Kargupta, H., & Harik, G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)* (pp. 56–64). Morgan Kaufmann Publishers, Inc. (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).

Harik, G., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation* (pp. 7–12). IEEE Publishers.

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. In *Foundations of Genetic Algorithms 4* (pp. 247–262). Morgan Kaufmann Publishers, Inc. (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.

Harik, G. R., Lobo, F. G., & Goldberg, D. E. (1999). The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation*, *3*(4), 287–297.

Heckerman, D., Geiger, D., & Chickering, M. (1994). *Learning Bayesian networks* (Tech. Rep. No. MSR-TR-94-09). Redmond, WA: Microsoft Research.

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

Iba, H., & Terao, M. (2000). Controlling effective introns for multi-agent learning by genetic programming. In *Proceedings of Genetic and Evolutionary Computation Conference 2000 (GECCO-2000)* (pp. 419–426). Morgan Kaufmann Publishers, Inc.

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

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. In *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA-91)* (pp. 123–127). Morgan Kaufmann Publishers, Inc.

Levenick, J. R. (1995). Metabits: Generic endogenous crossover control. In *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA-95)* (pp. 88–95). Morgan Kaufmann Publishers, Inc.

Marascuilo, L. A., & McSweeney, M. (1977). *Nonparametric and distribution-free methods for the social sciences*. CA: Brooks/Cole Publishing Company.

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. *Lecture Notes in Computer Science*, *1141*, 178–187.

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

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

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. (1999a). Identifying linkage groups by nonlinearity/non-monotonicity detection. In *Proceedings of Genetic and Evolutionary Computation Conference 1999 (GECCO-99)* (pp. 433–440). Morgan Kaufmann Publishers, Inc.

Munetomo, M., & Goldberg, D. E. (1999b). Linkage identification by non-monotonicity detectio for overlapping functions. *Evolutionary Computation*, *7*(4), 377–398. (Also IlliGAL Report No. 99005).

Nordin, P., Francone, F., & Banzhaf, W. (1996, October). Explicitly defined introns and destructive crossover in genetic programming. In Angeline, P. J., & Kinnear, Jr., K. E. (Eds.), *Advances in Genetic Programming*, Volume 2 (Chapter 6, pp. 111–134). MIT Press.

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).

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

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

Smith, J. (2002). On appropriate adaptation levels for the learning of gene linkage. *Genetic Programming and Evolvable Machines*, *3*(2), 129–155.

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

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. In *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)* (pp. 38–45). Morgan Kaufmann Publishers, Inc.

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

Wineberg, M., & Oppacher, F. (1996). The benefits of computing with introns. In *Proceedings of the First Annual Conference on Genetic Programming (GP 96)* (pp. 410–415). MIT Press.

Wu, A. S., Lindsay, R. K., & Smith, M. D. (1994). Studies on the effect of non-coding segments on the genetic algorithm. In *Proceedings of the Sixth IEEE Conference on Tools with Artificial Intelligence* (pp. 744–747). IEEE Publishers.