# Compressed Introns in a Linkage Learning Genetic Algorithm

**Fernando G. Lobo**
Dept. Environmental Engineering
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
Quinta da Torre
2825 Monte da Caparica, Portugal
fgl@mail.fct.unl.pt

**Kalyanmoy Deb**
Dept. Mechanical Engineering
Indian Institute of Technology
Kanpur, PIN 208016, India
deb@iitk.ernet.in

**David E. Goldberg**
Dept. General Engineering
University of Illinois at Urbana-Champaign
117 Transportation Building
104 South Mathews Avenue
Urbana, IL 61801
deg@uiuc.edu

**Georges R. Harik**
Member Technical Staff
Silicon Graphics
Mountain View, CA
gharik@engr.sgi.com

**Liwei Wang**
Dept. General Engineering
University of Illinois at Urbana-Champaign
117 Transportation Building
104 South Mathews Avenue
Urbana, IL 61801
lwang@illigal.ge.uiuc.edu

## ABSTRACT

**Over the last 10 years, many efforts have been made to design a competent genetic algorithm. This paper revisits and extends the latest of such efforts—the linkage learning genetic algorithm. Specifically, it introduces an efficient mechanism for representing the non-coding material. Recent investigations suggest that this new method is crucial for solving a large class of hard optimization problems.**

## 1 Introduction

The simple genetic algorithm (SGA) has been applied successfully in a variety of applications, including medical, financial, and all kinds of engineering problems. Its power comes from its ability to combine good pieces (building blocks) from different solutions and assemble them into a single super solution. But despite their success, there are still problems whose solution can be constructed by the juxtaposition of building blocks, and yet the SGA fails. The reason behind this failure is well understood and is due to the so-called *linkage problem*.

Before applying a genetic algorithm to a given problem, a solution to the problem has to be encoded into a chromosome structure. There are many ways in which this can be done, and some encodings are better than others. The general guideline is to try to put together in the chromosome features of the problem that are somehow spatially or physically related. These smart encodings are probably the main reason why genetic algorithms have had so much success in a variety of applications. By putting together in the chromosome things that are related, the GA user is hopefully seeding the GA with a good encoding to start with (good linkage information). If the encoding is indeed good, that is, if things that are related are close together in the chromosome, then most likely the GA will perform well. On the other hand, if things that are related are encoded far away from each other, then most likely the GA will perform poorly. The dependence of GA's performance on an appropriate encoding is a big obstacle to the GA's robustness, especially because in most problems there is not enough knowledge of what might be a good encoding.

Early on, Holland (1975) observed that the GA gives preference to the building blocks (BBs) whose genes are close together—*tightly linked genes*. However, he also noted that the genes that constitute a BB could be far away from each other—*loosely linked genes*. Holland suggested that these widely separated genes could come closer and closer by means of a reordering operator called inversion. His idea was to let the GA evolve a good ordering of the genes at the same that it evolves a good solution. Unfortunately, subsequent work has shown that inversion is not fast enough and eventually the population converges before inversion has time to reorder the genes (Goldberg & Bridges, 1990). Following Holland's work, Goldberg realized that tight linkage is essential for having good GA performance. With this in mind, he initiated the research in messy GAs (Goldberg, Korb, & Deb, 1989) (Goldberg et al., 1993), which in turn have spawned other GAs that pay serious attention to linkage learning (Kargupta, 1996; Harik, 1997). The goal is to design a GA that can solve problems of bounded difficulty quickly, reliably, and accurately—*a competent GA*. Designing such an algorithm has been a difficult and challenging task, and although a lot has been achieved, there is still some progress to be made.

This paper extends the latest of these research efforts, the linkage learning genetic algorithm (Harik, 1997), with a technique that seems promising for solving a large class of hard optimization problems.

The next two sections provide background material needed to understand this paper. Section 2 explains the limitations of SGAs and section 3 reviews the linkage learning genetic algorithm (LLGA). Then we describe a technique that doesn't change the behavior of the LLGA, but makes it computationally tractable for problems that require a large number of non-coding genes (more on this later). Finally, section 5 presents an empirical study of the time complexity of the LLGA on problems with exponentially scaled building blocks.

## 2 The limitations of SGAs

Genetic algorithms are successful when they can propagate building blocks from generation to generation and combine them into a single solution. Recall that a building block is an association of genes (bits in a binary coded GA) that as a whole give a high contribution to the fitness of an individual. A building block is atomic in the sense that we can't expect the GA to discover it by combining sub-parts of it. BBs are something that have to be discovered as a whole and be propagated as unbreakable units. Unfortunately the simple GA can only do so when the genes that constitute a BB are located near each other.

Within the SGA, each gene is coded at a fixed position in the chromosome. If the genes that constitute a BB are located near each other, the SGA has no trouble propagating it. After all, Holland's schema theory (Holland, 1975) says that short, highly fit similarities will tend to be propagated to subsequent generations. But what if the BB is not short? What if the BB is spread out along the chromosome? The answer is: SGAs fail because the BBs from the different individuals cannot be combined (mixed) in a reasonable amount of time. Thierens and Goldberg (1993) showed that the time needed to solve such problems grows exponentially as the problem size increases.

Ideally the BBs would be coded so that its genes are close together in the chromosome. But which genes constitute a BB is unknown beforehand. By using a fixed encoding, SGAs are not able to evolve a good ordering of the genes and their performance become dependent of whether the initial encoding is good or bad. This is a limitation and a severe obstacle to the GA's robustness.

### 2.1 Test problems

In order to test new GA mechanisms, it is useful to create artificial problems with a crisp definition of building blocks—problems of bounded difficulty. These problems are constituted by a number of non-overlapping building blocks of a maximum size $k$. The overall fitness function is simply the sum of the fitnesses of the individual building blocks. These artificially constructed problems are often studied by GA researchers with the hope that they bound the difficulty of a large class of real-world problems. Bounded difficult problems can be divided into subclasses:

- Problems with uniformly scaled building blocks.
- Problems with exponentially scaled building blocks.
- A mixture of the above two types.

As the name suggests, uniformly scaled BBs are BBs that give the same contribution to the fitness function. Exponentially scaled means that the fitness of each BB is scaled by some constant power. Some can argue that these types of problems are not representative of the problems encountered in real world applications. This is partially true and it is very unlikely that a real world problem has such crisp definitions. Nonetheless, we argue that most problems have some kind of BB-structure. After all, if problems had no structure at all, then a genetic algorithm would be no better than a random search method. The success that GAs have had in various applications supports the hypothesis that problems do have a BB-like structure, possibly with some degree of interaction among BBs.

Our first goal is to design a GA that can efficiently solve problems with no interactions among BBs. Once we have such a GA, the next step will be to study the effects of having interactions among BBs. This last topic is sometimes called crosstalk. A *competent GA* should be able to solve any of these problems quickly, reliably, and accurately. Such a GA doesn't fully exist yet, but we are closer than ever. Specifically, the linkage learning genetic algorithm (Harik, 1997) excels in problems with exponentially scaled BBs, but it is not as strong when the BBs are uniformly scaled. The next section reviews this algorithm.

## 3 The linkage learning GA

This section gives a brief overview of the mechanics and operation of the linkage learning genetic algorithm (LLGA). For a more detailed description and analysis you should refer to Harik's dissertation (Harik, 1997).

On problems with exponentially scaled building blocks, the LLGA autonomously rearranges the genes in the chromosome so that the genes that are related come closer in the chromosome—become tight. Once the linkage is tight, the LLGA has no trouble propagating the BBs. On these problems, the LLGA efficiently learns the linkage of each BB, one after the other. It first learns the linkage of the most important BB, and once this is formed, it moves on to next most important BB, and so on. It does so without ever losing diversity on the other genes due to a probabilistic expression mechanism. The remainder of this section describes the representation of the chromosome, the probabilistic expression mechanism, and the crossover operator.

### 3.1 Representation

The LLGA represents each gene by a (locus,allele) pair. There is no under-specification, but over-specification is always present. Every gene has always both allele values represented in the chromosome. One of them is expressed and sent to the fitness function according to a probabilistic scheme. Having the complement allele for every gene ensures that diversity is never lost. In addition, a chromosome also contains non-coding genes (sometimes called introns in the evolutionary computation community). These genes give no contribution to the fitness function, and their purpose is to facilitate the propagation of BBs and the formation of linkage. Other researchers have also investigated the effects of including non-coding material in the chromosome structures (Levenick, 1991; Levenick, 1995; Wu & Lindsay, 1995). These studies are motivated by the existence of non-coding DNA in biological systems. For example, all living creatures contain large amounts of non-coding DNA. Specifically, about 97% of the human DNA is non-coding.

It is convenient to visualize the chromosome as a circular list of genes. Figure 1 gives an example of such a chromosome. The example is for a 3-bit problem. Genes 1, 2, and 3 are the coding genes. Genes 4 and 5 are non-coding genes. Each chromosome has an interpretation point. Starting from this point, the genes are expressed by traversing the chromosome circle in a clockwise manner and recording the first occurrence of each gene. In the example, we would get (3,1) (2,0) (1,1). These 3 genes would be sent to the fitness function. Genes number 4 and 5 are not sent to the fitness function because they are non-coding. Note that for every gene, there is always at least one copy with the complement allele. Harik named this expression mechanism as probabilistic expression (PE) because each chromosome can express a set of different solutions depending on the location of the interpretation point. Harik extended his concept of PE and created an extended probabilistic expression (EPE-$n$) mechanism. In this general scheme, each gene has 1 copy of the expressed allele and at most $n$ copies of the unexpressed allele (Harik, 1997). Throughout this paper we use the EPE-2 mechanism, 1 copy of the expressed allele and at most 2 copies of the unexpressed allele. Harik (1997) has shown that this later scheme allows for effective linkage learning to occur. Next, we illustrate the crossover operator of the LLGA.

## 3.2 Crossover

The crossover operator works as follows. First, two chromosomes are randomly chosen from the population. One is named the recipient and the other is named the donor. A segment of the donor's chromosome is chosen randomly and is injected at a randomly chosen point (the grafting point) of the recipient. Then, the recipient chromosome is traversed in a clockwise manner and extra genes are deleted so that the resulting chromosome is a legal EPE-2 chromosome. Figure 2 illustrates the 3 steps of the crossover operator. In order to produce 2 offspring, the parent chromosomes exchange their roles of donor and recipient. When the selection rate is properly chosen, this crossover operator brings the genes that constitute a BB closer together. Details of why this happens are explained in Harik's dissertation as well as in (Harik & Goldberg, 1996). Once the linkage is tight, the BB has a very low chance of being disrupted and it is easily propagated to the other population members.

The generation loop of the LLGA is similar to that of the SGA. Selection and crossover are applied generation after generation. Unlike the messy GA, there's no explicit primordial and juxtapositional phases. Harik showed that the LLGA efficiently solve a large class of hard problems that are difficult for the SGA—problems with exponentially scaled BBs. However, he also recognized that the LLGA didn't perform as well on uniformly scaled problems as it did on exponentially scaled ones. To attack this problem, a new interpretation scheme has been recently investigated (Wang, 1997). The main idea of this new interpretation scheme is to let the building blocks form in different parts of the chromosome circle. Preliminary results suggest that these kinds of problems can be solved efficiently, but in order to do so, the LLGA seems to require a very large number of non-coding genes. This number appears to grow exponentially with the problem size. The next section introduces a technique that is capable of dealing with this exponential growth.

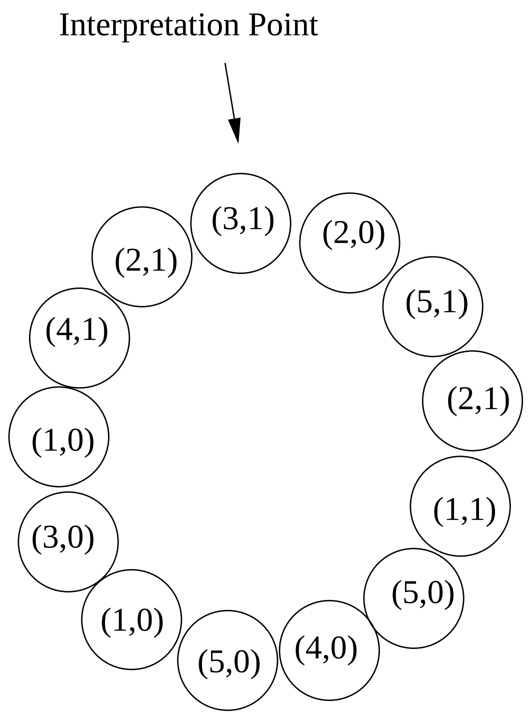

**Figure 1 Example of a chromosome in the LLGA. Genes 1, 2, 3 are coding genes. Genes 4 and 5 are non-coding. Starting from the interpretation point and traversing the chromosome in a clockwise direction, the chromosome would be expressed as (3,1) (2,0) (1,1). Note the existence of additional genes with the complement alleles.**

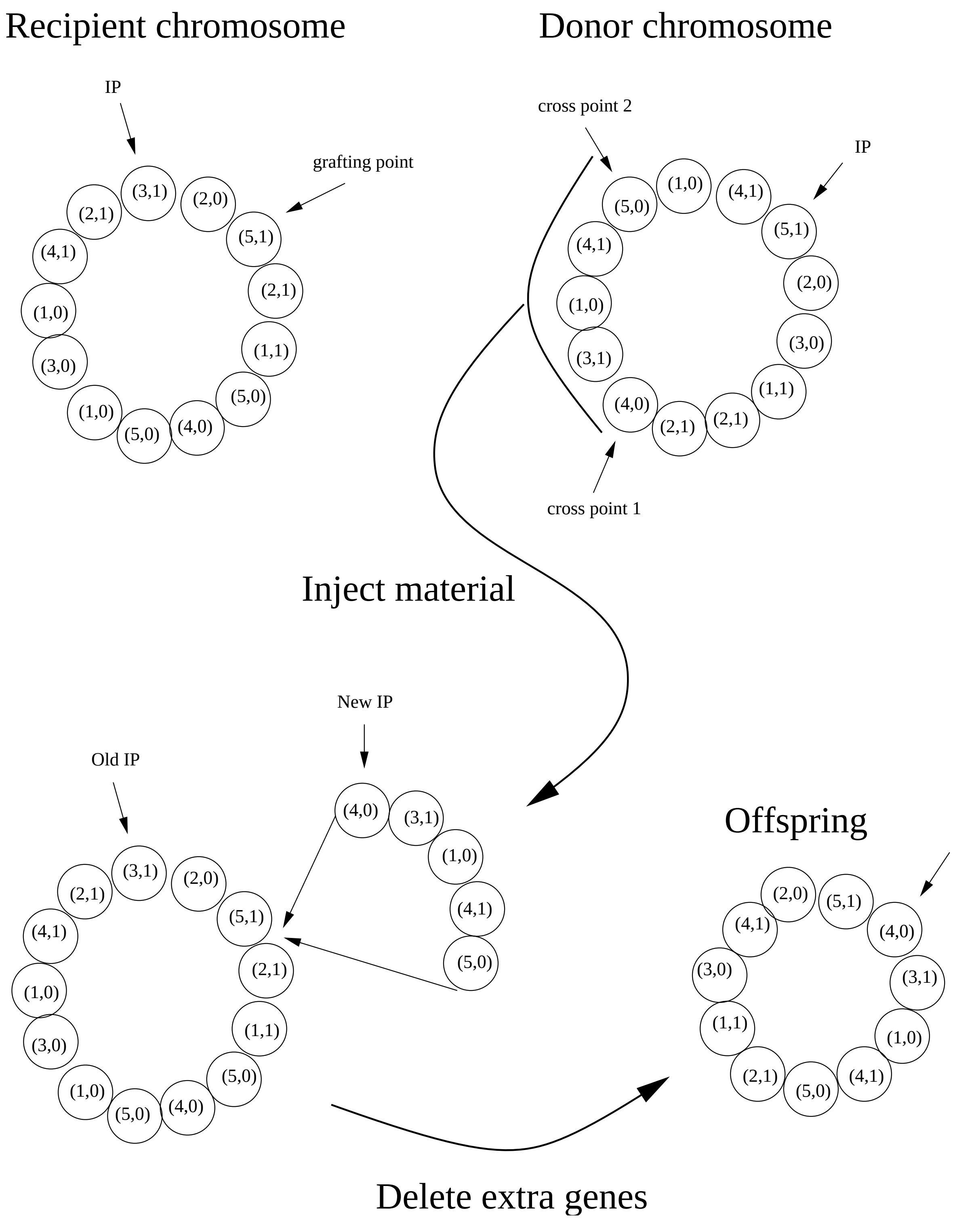

**Figure 2** **The crossover operator of the LLGA. Step 1: Pick a grafting point in the recipient, and pick a random segment from the donor. Step 2: Inject the random segment at the grafting point of the recipient. Step 3: Traverse the recipient and delete extra genes in order to make a legal EPE-2 chromosome. (Note: the choice of the cross and grafting points is not totally random. In the actual implementation, we make sure that they always fall in a non-coding gene. In this example, genes 4 and 5 are the non-coding genes).**

## 4 Simulating the LLGA with compressed introns

To cope with the exponential growth of non-coding material, we introduce a technique that compresses the non-coding genes, thus making it tractable from a computational perspective. The idea is simple. Instead of keeping the non-coding genes individually, we just keep the size of the non-coding segment (the number of consecutive non-coding genes) between each pair of coding genes. The storage requirement becomes $O(number\ of\ coding\ genes)$. The example in figure 3 illustrates the compressed representation.

With the compressed representation, the identity of the non-coding genes is lost. But this shouldn't be a problem because these genes give no contribution to the fitness function. Their purpose in the chromosome is to provide spacings to help the formation of linkage. This representation is just a trick to be able to cope with problems that require a large number of non-coding genes. But we also need to be able to simulate the LLGA's crossover operator as if it had the discrete coding. The expression mechanism remains the same, but the deletion of the extra non-coding genes needs to be simulated. An empirical analysis of the LLGA shows that very few genes are deleted from the grafted material. This suggests the implementation of the following deletion scheme:

1. Let $n$ be the number of non-coding genes in the grafted material.

2. Don't delete any genes from the grafted material.

3. Delete $n$ non-coding genes uniformly from the rest of the chromosome.

Experiments with the compressed representation and the new deletion procedure were performed. The modified version mimics the original LLGA. Figure 4 is a replication of the 19 exponentially scaled BB problem that is reported in Harik's dissertation (pp 89-90). Figure 5 shows the exact same problem being solved with the compressed representation. The graphs show the evolution of convergence and linkage of each BB. The linkage measure was defined by Harik (1997). It is a numerical value that indicates how tight a BB is. The tighter the BB, the closer the linkage is to 1.

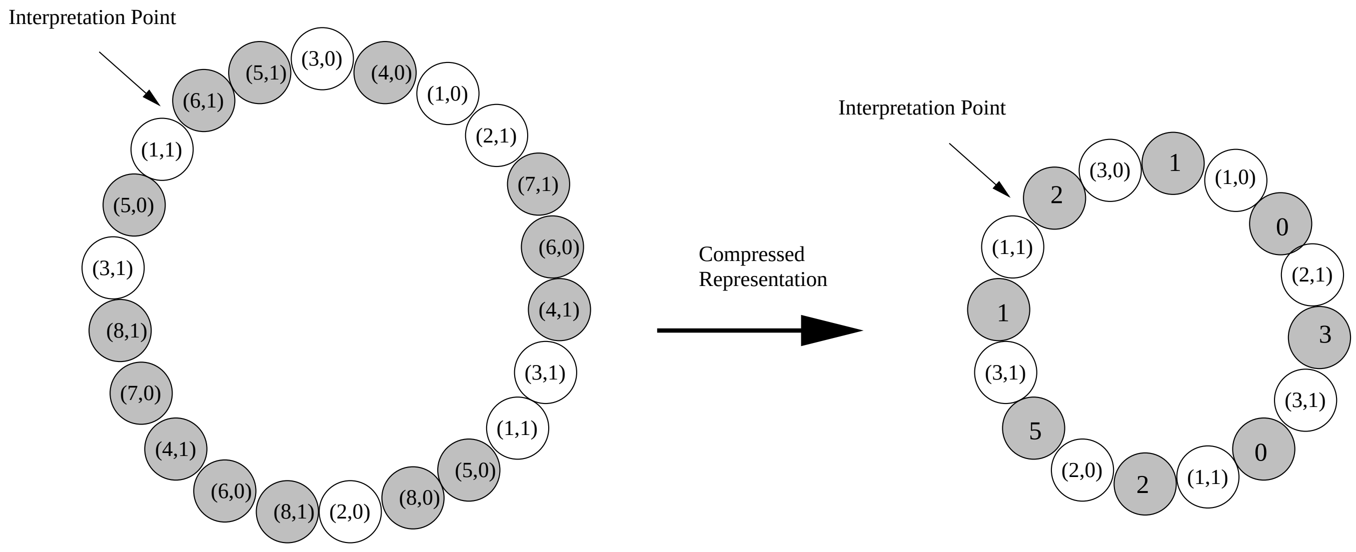

**Figure 3 Discrete (left) and compressed (right) representation of non-coding genes. In this example, genes 1, 2, 3 are coding genes. Genes 4, 5, 6, 7, 8 are non-coding and are shown in grey for easy visualization.**

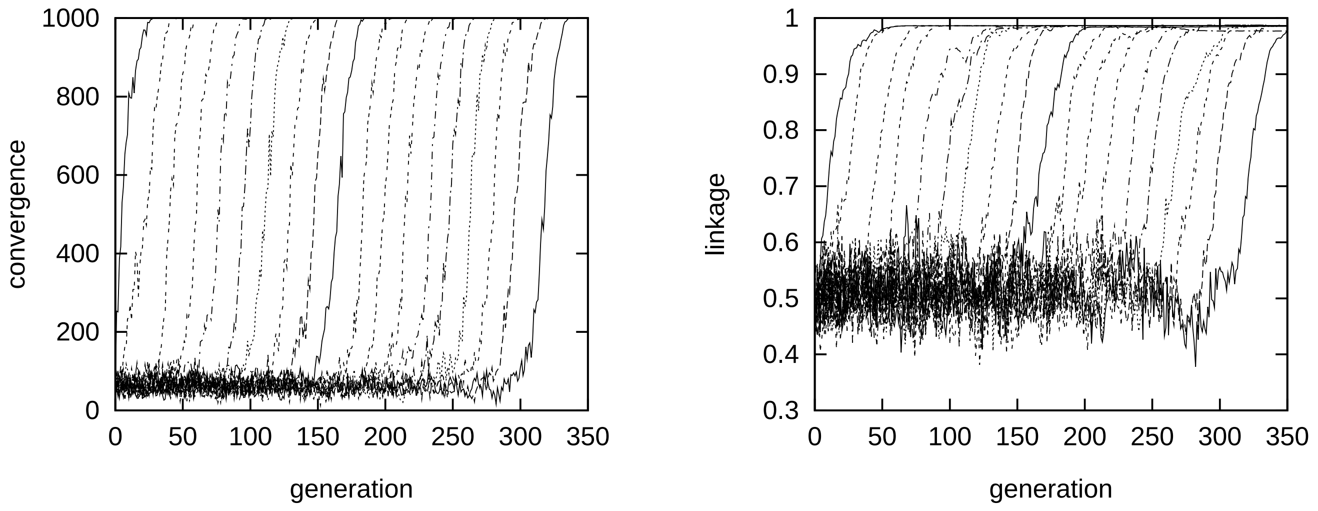

**Figure 4** The LLGA with discrete representation of introns. The left graph plot the convergence of each BB versus generation. The right graph plot the evolution of linkage for each BB. Each graph has 19 lines corresponding to the 19 BBs of the problem. Each BB is a 4-bit trap function with deceptive-to-optimal ratio of 0.6. The number of introns is 150, the population size is 1000, and tournament selection of size 4 is used.

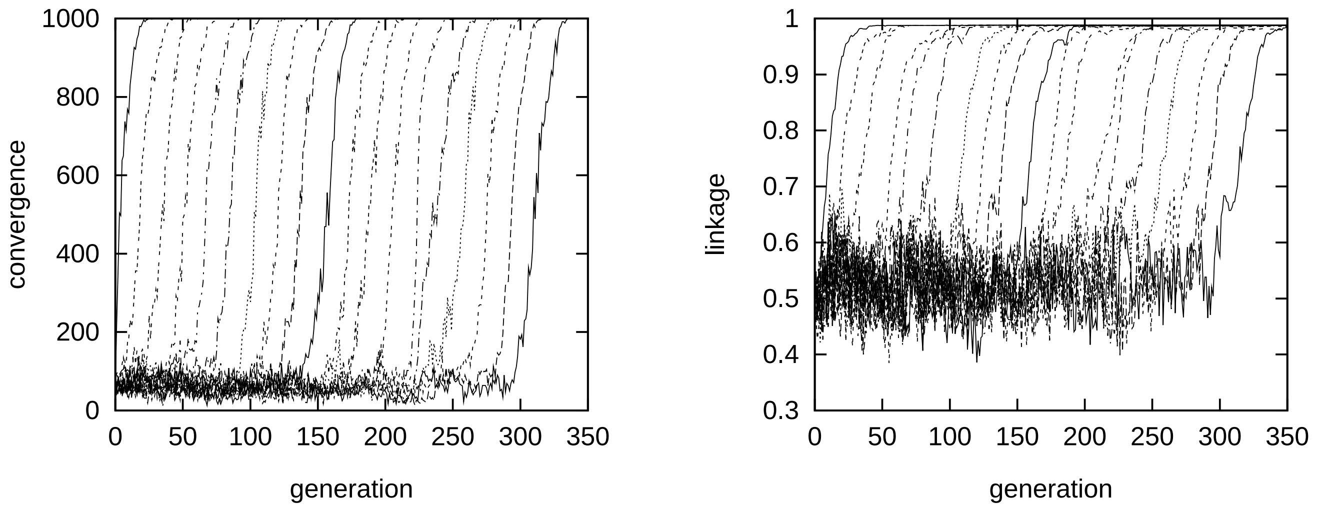

**Figure 5** The LLGA with compressed representation of introns. The parameters used are exactly the same as the ones described in the caption of figure 4.

## 5 Time complexity of the LLGA on exponentially scaled problems.

This section presents an empirical study of the time complexity of the LLGA on exponentially scaled problems. All simulations in this section were performed with the compressed representation of introns. We conduct experiments on problems with 20, 50, 75, and 100 building blocks. Each consists of a 4-bit trap function with a deceptive-to-optimal ratio of 0.8 (Deb & Goldberg, 1993). We record the minimum population size needed to solve the problem to optimality at least 4 out of 5 independent runs. For all these problems, the LLGA takes about the same time to solve each BB (around 17 generations each). For 100 BBs, it takes near 1700 generations to solve all of them. The pattern is the same as the one observed in figure 5. The LLGA learns the linkage of each BB, one after the other. The table below shows the minimum population size, the average number of generations, and the average number of function evaluations needed to get the optimal solution in at least 4 out of 5 independent runs. All experiments were performed using tournament selection of size 4. For the different problem lengths, we increase the number of introns linearly. The numbers were 150, 375, 562, and 750 respectively.

| Prob. length | Pop. size | Generations | Func. Eval. |
|---|---|---|---|
| 80 | 300 | 343 | 102,900 |
| 200 | 400 | 876 | 350,400 |
| 300 | 450 | 1289 | 580,050 |
| 400 | 550 | 1722 | 947,100 |

The number of generations needed to solve a larger problem increases linearly, but the population size increases very slowly. Figure 6 plots the number of function evaluations needed to solve the problem to optimality for different problem lengths. Taking the logarithm of the coordinates of the endpoints, we observe that the curve has a slope of approximately 1.4, which indicates that the time complexity of the LLGA is almost linear for exponentially scaled problems. It is clearly sub-quadratic. For these type of problems the LLGA has a clear advantage over the SGA, even when the SGA has good linkage information. The SGA has trouble with these problems because genetic drift occurs on the less salient BBs (Thierens, Goldberg, & Pereira, 1998). On the contrary, the probabilistic expression mechanism keeps diversity alive and therefore seems to eliminate the drift problem. The initial supply of BBs dominates the population sizing. Correct decision-making is secondary since there's almost no noise coming from the other partitions (Goldberg, Deb, & Clark, 1992), (Harik et al., 1997). The GA can correctly decide between two individuals based on the outcome of the competition in one BB alone.

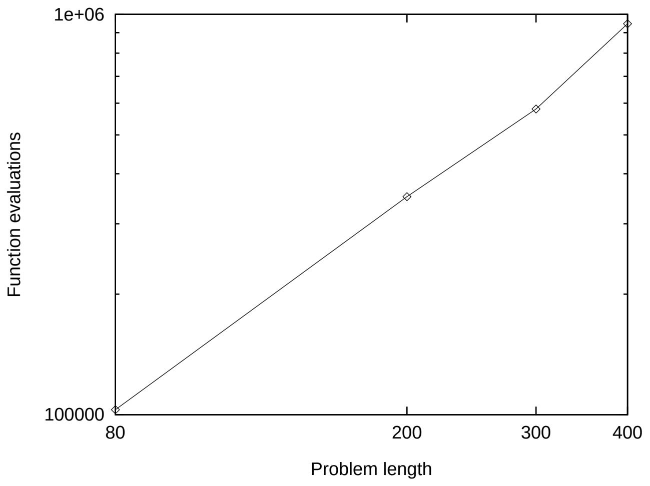

**Figure 6 Number of function evaluations needed to solve the problem to optimality at least 4 out of 5 runs.**

## 6 Conclusions

This paper revisited the linkage-learning genetic algorithm and showed that its time complexity is sub-quadratic for exponentially scaled problems. Aside from that, this work represents a small step towards a competent GA. The efficient representation of the non-coding material is likely to be crucial for the solution of uniformly scaled problems within the linkage learning GA framework. Investigations in this direction have already begun and will be reported soon.

For the past 20 years, many applications have had success using SGA-like technology. But it is possible to do better. Competent GAs are coming soon, and with them, hard optimization problems will be able to enjoy success as well.

## Acknowledgments

This work was done while the first two authors were visiting the Illinois Genetic Algorithms Laboratory.

This study was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants F49620-94-1-0103, F49620-95-1-0338, and F49620-97-1-0050. The U.S. Government is authorized to reproduce and distribute reprints for Governmental 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 or the U. S. Government.

Fernando Lobo was sponsored by "Junta Nacional de Investigação Ciêntífica e Tecnológica (Portugal)" through scholarship PRAXIS BD4020/94.

We also thank the reviewers for their useful comments.

## Bibliography

Deb, K., & Goldberg, D. E. (1993). Analyzing deception in trap functions. In Whitley, L. D. (Ed.), *Foundations of Genetic Algorithms 2* (pp. 93–108). San Mateo, CA: Morgan Kaufmann.

Goldberg, D. E., & Bridges, C. L. (1990). An analysis of a reordering operator on a GA-hard problem. *Biological Cybernetics*, *62*, 397–405.

Goldberg, D. E., Deb, K., & Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, *6*, 333–362.

Goldberg, D. E., Deb, K., Kargupta, H., & Harik, G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 56–64). San Mateo, CA: Morgan Kaufmann.

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, *3*(5), 493–530.

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

Harik, G. R., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In Bäck, T. (Ed.), *Proceedings of the Fourth International Conference on Evolutionary Computation* (pp. 7–12). New York: IEEE Press.

Harik, G. R., & Goldberg, D. E. (1996). Learning linkage. In Belew, R. K., & Vose, M. D. (Eds.), *Foundations of Genetic Algorithms 4* (pp. 247–262). San Francisco, CA: Morgan Kaufmann.

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

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

Levenick, J. R. (1991). Inserting introns improves genetic algorithm success rate: Taking a cue from biology. In Belew, R. K., & Booker, L. B. (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 123–127). San Mateo, CA: Morgan Kaufmann.

Levenick, J. R. (1995). Metabits: Generic endogenous crossover control. In Eschelman, L. (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 88–95). San Francisco, CA: Morgan Kaufmann.

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 38–45). San Mateo, CA: Morgan Kaufmann.

Thierens, D., Goldberg, D. E., & Pereira, A. G. (1998). Domino convergence, drift, and the temporal-salience structure of problems. *Proceedings of the Fifth International Conference on Evolutionary Computation*. in Press.

Wang, L. (1997). Personal communication.

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