Investigating Degenerate Code and Gene Dependency in the GAuGE System

Miguel Nicolau  
C.S.I.S. Department  
University of Limerick  
Ireland  
Miguel.Nicolau@ul.ie

Anne Auger  
CERMICS – ENPC  
Cité Descartes, 77455 Marne-La-Vallée  
France  
auger@cermics.enpc.fr

Conor Ryan  
C.S.I.S. Department  
University of Limerick  
Ireland  
Conor.Ryan@ul.ie

Abstract

This paper explores the topics of gene dependency and degenerate code, and their combined effect in the GAuGE system, a recently introduced position-independent genetic algorithm. To do so, a simulation was used to calculate the average position specifications of all genotype individuals, and the effect of degenerate code on those averages. The results obtained so far suggest that the introduction of degenerate code loosens the dependency between the position coding genes in each individual.

1 Introduction

The GAuGE system (Genetic Algorithms using Grammatical Evolution) is a recently introduced position independent genetic algorithm [13], which encodes both a position and a value for each locus in the phenotype string. It suffers from neither under nor over specification, due to its use of a genotype-to-phenotype mapping process; that is, a position specification is mapped onto a set of available positions in the resulting phenotype string, ensuring that a fixed-length genotype string will always generate a single value for each locus in the phenotype string.

Two of the main biologically inspired features present in GAuGE are gene dependency and code degeneracy. This work is an investigation into both features, and their combined effect on the position specifications of the genotype strings, in the initial generation of individuals. Experiments conducted suggest that an increase of degeneracy leads to a weaker gene dependency across the genotype, which in turn generates individuals which are less biased in what concerns the locus of specifications in the genotype string.

This paper is organised as follows. Section 2 gives a brief presentation of the Grammatical Evolution system, and Section 3 presents the main features of the GAuGE system. Section 4 presents and analyses the experiments conducted, and Section 5 draws some conclusions and possible future work directions.

2 Grammatical Evolution

Grammatical Evolution (GE) [12, 9, 11] is an automatic programming system that uses grammars to generate code written in any language. It uses a genetic algorithm to generate variable-length binary strings, which are interpreted as a series of integer values (each value being encoded with 8 bits). These values are then used to choose productions from the rules of a given grammar, to map a start symbol onto a program (the phenotype), consisting solely of terminal symbols extracted from the grammar. This genotype-to-phenotype mapping, based on the analogous process in molecular biology, provides a division between the search space (binary strings) and the solution space (evolved programs) [1].

Another interesting feature of GE is the functional dependency across the genes on each genotype string. Indeed, the function of a gene is dependent on those genes preceding it, as previous choices of grammar productions will dictate which symbol is to be mapped next, and therefore which rule is to be applied. This creates a dependency across the genotype string, ranging from left to right.

Finally, the use of degenerate code plays an important role in GE: by using the mod operator to map an integer to the choices of a grammar rule, neutral mutations [5] can take place, as different integers can choose the same production from a given rule. This means that the genotype can be modified without changing the phenotype, allowing variety at the genotypic level.
3 GAuGE

GAuGE has an approach to evolutionary computation that is similar to that of GE, and is based on most of the same biologically inspired features. Each member of the population is a fixed-length binary string, and a genotype-to-phenotype mapping process is used to create another fixed-length string as the phenotype result.

As an example of the mapping process, consider the following individual:

\[
genotype = 00010101000001000000100100001011
\]

In a process similar to GE, the mapping process will interpret this string as a series of integer values. In the case of GAuGE, these will be a sequence of (position, value) specifications; if we choose 7 bits as the position field size \(pfs\), and 1 bit as the value field size \(vfs\), this string will be interpreted as

\[
X = (x_i)_{0 \leq i \leq 2^6-1} = ((10,1)(2,0)(4,1)(5,1)),
\]

where each integer with an even index \(x_{i+2}\) specifies a position in the phenotype string, and each integer with an odd index \(x_{i+1}\) a value for that position (thus effectively encoding a phenotype string of length \(l = 4\)).

The construction of the phenotype from these pairs then works as follows. Taking the first pair, \((10,1)\), the position specified \(x_0 = 10\) is modied by the number of free positions in the phenotype string \(4\), giving the desired position \(dpos(x_0) = 10\%4 = 2\). As all positions are available in the phenotype string at the start, then the real position to be taken in that string is the same, which means that \(rpos(x_0) = 2\), i.e. the third position on the phenotype string, where the value specified \(x_1 = 1\) is placed:

\[
phenotype = ? ? 1 ?
\]

For the next pair, \((2,0)\), its position specification is moded by the number of remaining slots in the phenotype (now only 3), giving the desired position \(dpos(x_2) = 2\%3 = 2\), the third position available in the phenotype string; this is effectively the fourth real position in that string, so \(rpos(x_2) = 3\), where the value specified \(x_3 = 0\) is placed:

\[
phenotype = ? ? 1 0
\]

The mapping process continues until all pairs have been interpreted. This process clearly demonstrates the gene dependency across the genotype string, as the phenotype position each pair encodes depends on previous specifications.

Previous work has used similar approaches and techniques as the ones employed in GAuGE. Work by Bean [2] with the Random Keys Genetic Algorithm (RKGA) hinted that a tight linkage between genes would result in both a smoother transition between parents and offspring when genetic operators are applied, and an error-free mapping to a sequence of ordinal numbers. More recently, Harik [3] has applied the principles of functional dependency in the Linkage Learning Genetic Algorithm (LGA), in which a chromosome is expressed as a circular list of genes, with the functionality of a gene being dependent on a chosen interpretation point, and the genes between that point and the current gene.

4 Gene Dependency and Degenerate Code

The work presented here is focused on the degenerate code and gene dependency features of GAuGE. The effect of degeneracy in the performance of GE has been studied previously [8], and recent work has investigated the gene dependency in the position specification of GAuGE [6], and its extension to the value field [7].

This work is aimed at investigating the effect of both features on each other, at the start of a run. To do so, a Monte-Carlo simulation [4] was used to calculate the average real positions specified by a genotype string. The principle of the Monte Carlo simulation is as follows: in order to calculate the average real positions specified \(E(rpos(x_{i+2}))\) by an individual \(X = ((x_0,x_1),\ldots,(x_{i+2-2},x_{i+2-1}))\), where \(x_{i+2} \in \{0,1\}^pfs\), with \(pfs\) the size of the position field (in bits), a population of \(N\) individuals (with \(N\) large) is generated. The \(N\) individuals are denoted \(X^N = ((x_{i+2}^0,x_{i+2}^1),\ldots,(x_{i+2}^{N-2},x_{i+2}^{N-1}))\). Each mean of the real position specification \(E(rpos(x_{i+2}))\) is approximated by

\[
\frac{1}{N} \sum_{j=1}^{N} rpos(x_{i+2}^j).
\]

A consequence of the central limit theorem is that there is a confidence interval for this approximation. More precisely, for the 99% interval confidence, there is a probability equal to 0.99 that the true values of the average real positions are within the interval

\[
\left[ E\left(\frac{1}{N} \sum_{j=1}^{N} rpos(x_{i+2}^j) - \alpha_i(N)\right), E\left(\frac{1}{N} \sum_{j=1}^{N} rpos(x_{i+2}^j) + \alpha_i(N)\right) \right] \]

where \(\alpha_i(N)\) is the upper bound of the confidence interval for \(E(rpos(x_{i+2}))\).
Table 1: Experimental setup. 7 bits were used for the position field, as this is the minimum number of bits required to encode 128 positions.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem length (l)</td>
<td>128</td>
</tr>
<tr>
<td>Population size (N)</td>
<td>100000</td>
</tr>
<tr>
<td>Position field size</td>
<td>7 bits</td>
</tr>
<tr>
<td>Value field size</td>
<td>1 bit</td>
</tr>
</tbody>
</table>

where \( \alpha_1(N) \) is equal to

\[
2.58 \sqrt{\frac{1}{N} \sum_{j=1}^{N} r_{pos}(x_{j+2})^2 - \left( \frac{1}{N} \sum_{j=1}^{N} r_{pos}(x_{j+2}) \right)^2}
\]

(2)

The values used in the simulations are given in Table 1. Figure 1 shows a plot of those averages, along with the 99% interval.

The graph illustrates the effect of gene dependency across the genotype, from the left to the right side. As each position specification along the genotype is modded by smaller values (128, 127, ...), the real position it specifies is smaller on average; however, as more positions are taken in the first half of the phenotype string, the free positions on the second half of that string get chosen more often, which explains the increase in the real positions specified at around gene 32. This balancing phenomenon continues on, creating a ripple effect [11] across the various position specifications.

### 4.1 Introducing degeneracy

The introduction of degeneracy has an interesting effect. By using a larger number of bits for the position field, the range of numbers each field can specify is increased (e.g., using 14 bits, the range will be 0-16383, rather than 0-127 with 7 bits per gene); when modded by the number of available positions (128, 127, ...), this reduces significantly the differences between the position specified by each field, effectively flattening the ripples.

Figure 2 illustrates this effect: by gradually increasing the number of bits used to encode each position field, the differences in the average real positions specified are reduced, as is the slight increasing average position specified towards the end of the genotype string.

### 5 Conclusions

This paper presents the start of an investigation into the combined effect of degeneracy and gene dependency in the GAUGE system. Experiments conducted have shown how increasing the degeneracy in the position field leads to a decrease in the gene dependency effect, in the initial generation. Future work should concentrate on accurate measurements of gene dependency, and the effect of both features on the performance of the system.

### References

Figure 2: Effect of degeneracy on the average position specification per gene. The x-axis shows each gene in the genotype string, the y-axis (depth) shows the number of bits used for the position specification, and the z-axis (vertical) shows the average position specification of that gene (results averaged over 100000 randomly created individuals)


