View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by DIAS Access to Institutional Repository

Zero is not a four letter word : Studies in the evolution of language.

M. Nicolau[1], C. Ryan[1], and Christopher R. Stephens[2,3]

[1] Department of Computer Science and Information Systems
University of Limerick, Limerick, Ireland
[2] Dublin Institute for Advanced Studies
10 Burlington Road, Dublin 4, Ireland
[3] Instituto de Ciencias Nucleares, UNAM
Circuito Exterior, A. Postal 70-543
México D.F. 04510

Abstract. We examine a model genetic system that has features of both genetic programming and genetic regulatory networks, to show how various forms of degeneracy in the genotype-phenotype map can induce complex and subtle behaviour in the dynamics that lead to enhanced evolutionary robustness and can be fruitfully described in terms of an elementary algorithmic "language".

DIAS preprint DIAS-STP-04-19
To be published in EUROGP2005

1 Introduction

Evolutionary algorithms (EAs) have been applied with a remarkable degree of success to a large variety of problems. However, this is often done with little or no understanding of the dynamics of the system, and practitioners often find themselves unable to explain why a particular tweak has apparently improved their system. Features such as degeneracy and neutral evolution are generally accepted to aid evolution, but little detailed work, particularly in Genetic Programming (GP), though see [1] for a notable exception, has been carried out to investigate how GP exploits these features. This paper presents a study in evolutionary dynamics, demonstrating how a detailed analysis detects complex and subtle structure formation. Phenomena such as competing conventions, robustness and redundancy are examined and we demonstrate how it is natural to describe these phenomena in the framework of natural language.

In section 2 we describe the representation of our system, showing how it has all the salient features of GP systems, as well as a Genotype-Phenotype map (GPM) inspired by genetic regulatory networks. We also describe the different types of degeneracy inherent in the system. Section 3 takes an in depth look at a representative run, illustrating some surprising strategies that are being adopted by the system. The paper concludes with a summary and discusses areas in which

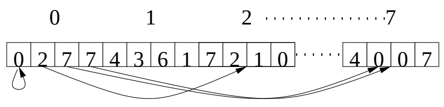

Fig. 1. An example of the switchboard gene in operation. Each codon in the switchboard gene (in the current version, always located in the first position) acts as an index into the *entire* genome. Notice how this particular switchboard gene indexes itself.

our system can be used to develop a deeper understanding of bloat, degeneracy and neutrality, all of which are crucial to the development of better and more robust EAs.

2 Representation

The representation we use is based on two existing systems; Grammatical Evolution (GE) [2] and that of [3], where a GPM inspired by gene expression models and the phenomenon of cellular division was used. Both exhibit genetic redundancy: GE via a *degenerate* mapping scheme, where many *codons* map to the same item, while in [3] the redundancy was at the gene expression level, in that only a certain proportion of genes from an individual needed to be expressed to produce a fully specified phenotype.

The underlying representation is similar to GE in that we employ a binary string representation. However, in our case, we use a fixed length string and, typically, shorter codons. The genotype consists of N_g genes, with each gene consisting of N_c codons, where each codon takes a symbolic value taken from an alphabet of size N_a. In the experiments described in this paper we consider $N_c = 4$ and $N_g = N_a = 8$, so each codon is described by three bits.

The first step is to *transcribe* the genome from the binary representation to eight genes of four codons each. We use a *gene expression* approach where each codon is indexed with a value from 0 to 7, and these values can subsequently be used to inhibit or promote the genes. This inhibition/promotion is controlled by the first gene, which is known as the *switchboard* gene, and is always expressed initially. Each of the four codons that appear in the switchboard is used to index a gene which is then activated as indicated in Figure 1. Then, in a manner not entirely dissimilar to cellular growth, the initial structure is replaced with a new one, consisting of four genes. Notice that a gene can be activated more than once, and the switchboard can also activate *itself.* Notice also that this can be an iterative process, that is, the four genes can be looked upon as containing a list of indices into another *sixteen* genes, which can, in turn, be used to index another 64 genes, and so on. Figure 2 shows how the example from Figure 1 is remapped. In this paper, we only examine individuals produced in a single growth iteration.

Once the activated genes have been identified, the system reverts to a standard GE type mapping, with each codon being used to make a choice in a

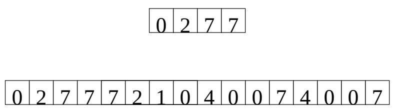

Fig. 2. An example of remapping using the genome from Figure 1. The system starts with the switchboard gene, before replacing it with the genes it indexes. In this case, it indexes *itself* initially.

grammar. Consider the grammar below, with each production rule numbered. As the codons are represented by three bits, each decodes to a value from 0 to 7.

```
<e> :: = tanh(<e>)      (0)
         tanh(<e>)      (1)
         tanh(<e>)      (2)
         tanh(<e>)      (3)
         add(<e>,<e>)   (4)
         add(<e>,<e>)   (5)
         X              (6)
         0              (7)
```

This is referred to as a *closed* grammar [2], that is, there is only one non-terminal and, hence, just a single context.

Consider the following individual, which has already been reduced to its activated genes:

```
4567 1623 0021 4401
```

Each codon will *always* make the same choice. Thus, codon 0 will always perform the mapping `e -> tanh(e)`, while codon 7 will always perform the mapping `e -> X`. The mapping steps are as follows:

```
4 -> add(e,e)
5 -> add(add(e,e),e)
6 -> add(add(X,e),e)
7 -> add(add(X,0),e)
1 -> add(add(X,0),tanh(e))
6 -> add(add(X,0),tanh(X))
```

Notice that only six codons are required for the mapping, and that the remaining codons are ignored. In the case where an individual has used all of its codons and still has a non-terminal remaining, it is given a fitness of zero. This straightforward mapping scheme permits us to use a more convenient notational representation to facilitate human interpretation of codons. That is, a shorthand for each codon value can be inserted as follows: 0, 1, 2, 3 → `h`; 4, 5 → +; 6 → `X` and 7 → `0`. Thus, the individual above can be rewritten as:

```
++X0 hXhh hhhh ++hh
```

This shorthand notation is a more coarse description than one would get when using the codon values themselves, however, as will be seen later, we are interested in the choices made, rather than the actual codon values. We refer to genes described in this way as *words*.

One can fruitfully think of the GE mapping [4] as identifying three categories of production rules : *Producers*, *Consumers* and *Neutrals*. Producers are those rules which increase the number of non-terminals in the expression being mapped, e.g. `<e> ::= add(<e>,<e>)`, while Consumers reduce the number of non-terminals, e.g. `<e> ::= X`, and Neutrals leave the number unchanged, e.g. `<e> ::= tanh(<e>)`.

2.1 Degeneracy

A key feature of our model is the hierarchy of levels at which degeneracy occurs. The most primitive level is that in which the codons are mapped. Codons are required to have a value between 0 and 7 because there are eight production rules in the grammar. At the bit level, there are three bits per codon, which map into this range. However, if there were more bits per codon, the range would be larger, and the values produced would have to be transformed to the meaningful range. This type of degeneracy is common to all GE type systems [7] [8].

Degeneracy at the mapping level occurs when there is more than one way to select a particular production rule. In the grammar used here, degeneracy exists for `tanh(e)` and `add(e,e)`. This type of degeneracy is present in most systems that employ grammars, which include the GE-like systems referenced above, but also systems such as [9] [10].

A further level of degeneracy exists due to the manner in which "words" are generated by activating only a subset of genes, this subset ranging from one to four, as the switchboard gene could, in theory, point to itself four times. It is quite possible, as is demonstrated in section 3.1, that a gene or a subset of its codons will be identical, but only one of them will be activated. This type of degeneracy is particular to the gene expression inspired GPM we used here.

Further degeneracy still can occur via the fact that not all activated genes/words will necessarily be expressed when mapping an individual. In our example, only the first two were expressed, which means the latter two can be replaced with *any* other words from that individual without affecting the output. Thus, using schema notation, one could describe the above "sentence" as

```
++X0 hX** **** ****
```

Notice that the second word is not made up of four distinct codons/letters. More generally, the last word *used* can vary in length from one to four letters.

The final type of degeneracy that can occur is present at the expression level. This is where different expressions evaluate to the same thing. For example, each of the following will all produce the same value.

- $add(add(X,0),add(X,0))$
- $add(add(0,X),add(X,0))$

– $add(add(X, 0), add(0, X))$
– $add(add(0, X), add(0, X))$

All automatic programming systems experience this type of degeneracy. In this work we differentiate between genes that actually contribute to the mapping and those that were activated but don't contribute as *expressed* genes.

2.2 Neutrality

Degeneracy leads to the existence of neutral networks [5], where individuals from different areas in the search space have the same fitness. Whether such individuals are connected by the genetic operators used depends on what these operators are. Operators are neutral when the individual they produce is genetically different to the individual that they operated on, but phenotypically the same. Point mutation can be neutral at several of the levels above, e.g. changing a codon value such that it still selects the same rule as before, changing a value on the switchboard gene so that it generates the same word as before, but from a different gene, etc. It is also possible to perform neutral crossover. Again, each of the above types of degeneracy can be exploited with this operator. For example, crossover might only effect non-activated genes, both parents might have a copy of a required gene etc.

2.3 Gene expression as a language

The kinds of words that one would expect to be produced by this grammar depends on the fitness function, and it is likely that certain letters and words will be selected against, or that certain combinations of letters, or amounts of letters will be selected for or against.

Consider the function $f(X) = 4X$. This can be described by using a number of different sentences, and the fitness function provides a measure of how well an arbitrary sentence describes the target function. One possible solution (showing only the expressed genes) is :

```
+++X XXX*
```

Recall that $*$ is a don't-care symbol, and indicates that the mapping has finished before that codon is needed. This particular sentence maps to a minimal solution of $(+(+XX)(+XX))$. Another solution, which maps to the same phenotype is :

```
+X+X +XX*
```

Notice how these sentences are fundamentally different because their first words differ. This does not suggest a lack of robustness, however, as one would not expect a single evolving population to balance two such different solutions at the same time for very long, although, as described below, it is possible for a number of distinct optimal solutions to appear throughout a run, often competing with each other for dominance of the population, until one becomes extinct.

The role of modularity in most complex problem solving systems, including nature, cannot be over estimated. Difficult problems are often best solved by decomposing them into a set of smaller ones, each of which can be solved more easily than the whole. Similarly, simple modules or strategies which can be reused several times, either on different problems or while solving a single problem are likely to be preferred over more complex ones. Consider the individual:

```
+++X +++X X000 X000
```

This maps to $(+(+X(+(XX))(+00))(+0X))$, using 13 codons, and can be reduced to $4X$.

3 Results

We applied the system to the problem of performing symbolic regression on the function $f(X) = 4X$. By normal GP standards, this is a trivial problem which one would expect to appear in a reasonably sized population with a good initialisation scheme. However, we are concerned with making a detailed analysis of the dynamics, and a simple function like this keeps the analysis tractable.

In total 30 runs were conducted, all of which discovered an optimal solution. Typically, the solutions initially consisted of three or four expressed genes, but shorter solutions almost always appeared, reducing the length to usually two or sometimes three genes. Repeated activation of the same gene was ubiquitous. There are several ways in which an optimal solution can be represented, and, typically, each run discovered several of these. A population of 100 individuals was used with a mutation rate, implemented at the bit level, of 0.01. One-point recombination was used with probability 0.9 and restricted to occur only at the boundaries between genes. For selection, a rank-based method was used, where the ranking was applied only to individuals that successfully mapped onto syntactically correct expressions.

3.1 Description of Algorithmic Language

In this section we consider a detailed description of a particular run in order to show the complexity of the dynamics associated with the GPM, even in the case of our very simple search problem. With the production rules specified in section 2, starting off with a random population one finds, as expected, that the initial codon frequencies are approximately: $h = 50\%$, $+ = 25\%$, $X = 12.5\%$ and $0 = 12.5\%$. Additionally, these are the approximate frequencies for any genetic locus. Later on however, one sees structure begin to emerge - for instance, usage of codon h is significantly less than one might expect over the majority of the run, while usage of codon X is significantly greater, as it is more likely to play a useful role in fit solutions. With codon 0 there is a greatly increased usage over the middle part of the run, for a reason that will become apparent shortly.

As on the switchboard gene the distribution of codons is random, one also finds initially a random distribution of activated genes, i.e. that any gene is active

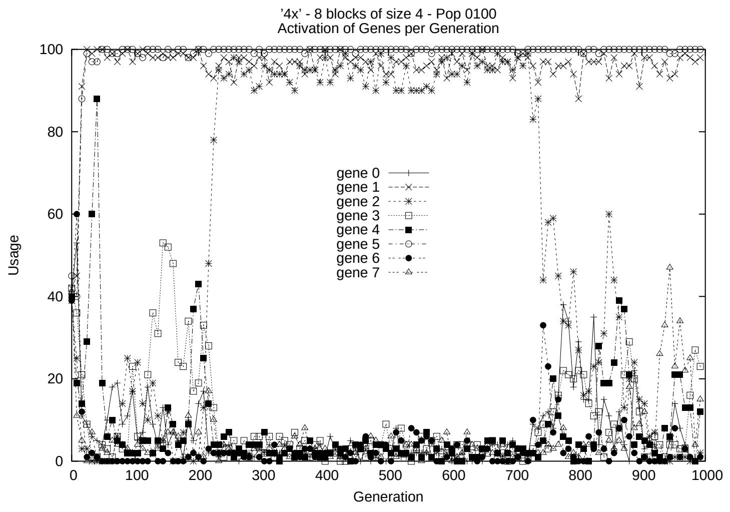

Fig. 3. Gene activity per generation. The graph plots the percentage of individuals per generation that have a particular gene activated.

on average in 50% of the population. Once again, later on however, gene activity patterns emerge with a great deal more structure, as can be seen in figure 3. One immediately sees that very early on in the evolution an ordering convention is arrived at whereby the content of the switchboard gene is largely fixed, with more than 90% of the population using codons 5 and 1, which activate genes 5 and 1 respectively. Interestingly, there is a redundant usage, whereby these codons are very frequently repeated in the switchboard. As we shall see this leads to enhanced robustness. At generation 25 more than 95% of switchboard genes in the population are of the form $5 * **$ while over 60% are of the form $55 * *$. Similarly, 92% possess $* * 1*$ and 83% possess $* * 11$.

Checking then the contents of genes 5 and 1: In figure 4 we show the different codon frequencies in gene 5 as a function of time. Before an optimal solution is found gene 5 consists of almost 50% of codons of type $+$, i.e. producers. This is about double what would be expected with a random distribution. Similarly, block 1 contained nearly 90% more codons of type X than would be expected by chance. Gene 5 is clearly the most important, or "core", gene as its codon content is so stable that there must be strong selection pressure in order to maintain this stability. Gene 1 by contrast showed more variation, though the important role played by codons X and 0 was evident. Note that when expressed, gene 5 precedes (i.e. is to the left of) gene 1 hence, as expected, we see the system puts more producers to the left and more terminals to the right, a conclusion that is completely consistent with the different codon frequencies seen in these genes. It is interesting to note that optimal solutions are detected from time to time but it is not until after generation 200 that they become established in the population.

It is also noteworthy that the fitter solutions tend to use only two different active genes - 5 and 1 - and three expressed ones - $551*$ - the final non-coding tail not being expressed. The most common phenotype is $f(X) = 2X + 2\tanh(X)$, that results from a combination of a repeated producing gene $5 = + + hX$ and a consuming gene $1 = XX00$.

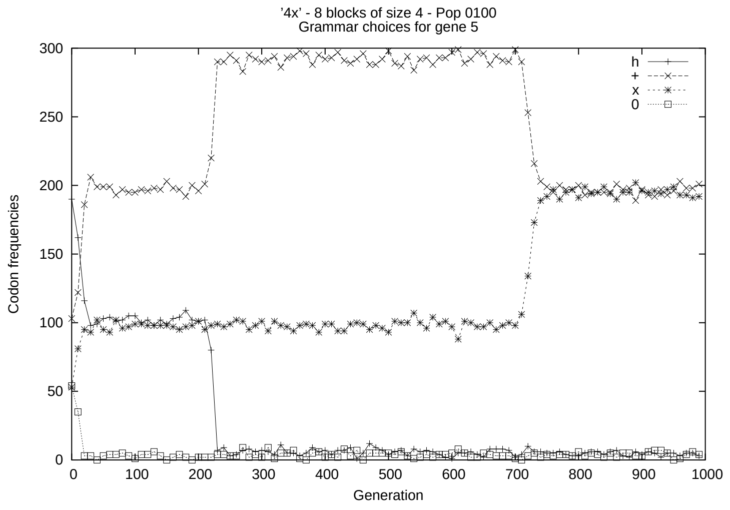

Fig. 4. Grammar choices for gene 5 per generation.

At generation 213 optimal solutions begin to be found that finally successfully propagate through the population. At this point the number of + codons in gene 5 increases by 50% while the number of terminal 0 codons in gene 1 doubles. The first solution found is of the form below. Notice that the switchboard block appears twice, once in decimal form to facilitate reading, and once in the normal form, $+ + hh$ in this case.

```
5533.++hh.XX00.0+h+.X000.++hX.+++X.++Xh.+Xhh
```

```
(+(+(+X(+(+(+X X ) 0) 0)) 0)) X) = 4X
```

which consists of 4 expressed genes but uses only 2 activated ones. Note that this founder does not activate gene 1 as the vast majority of the population do. It is based on the same switchboard template $55**$ of previous common suboptimal solutions but is achieved through a single point mutation of the 5 gene $++hX \rightarrow +++X$ which, repeated, combines with a totally consuming gene $3 = X000$ to give $4X$. 5533 however, is not the dominant switchboard gene, that role being kept by 5512, and therefore the dominant optimal solution had 5512 as a founder switchboard rather than 5533. Moreover, with this switchboard, in general all four genes 5, 5, 1 and 2 are expressed, as can be seen in the huge increase in use of gene 2 after generation 200 in figure 3.

Given that all these solutions are optimal one might expect that, on average, evolution preserves their structure, apart from the effect of neutral drift. However, this is not the case. For the next 500 generations this solution spreads and evolves. Early on, among the optimal solutions, in gene 1 over 60% of the individuals have a $+$ codon. Later on this percentage has dropped to zero! It is important to emphasize that there is no direct selection pressure for this, as we are talking about the structure of this gene only for optimal individuals. In terms of effective fitness [6] however, there is a clear explanation: any $+$ in gene 1 means that in order to maintain an optimal solution more codons from gene 2 must be expressed. As these are subject to mutational damage there is an effective selection pressure to make the solution more robust within the context of a repeated "core" gene, $+ + +X$, that needs a minimum of 5 more terminal codons. The elimination of the $+$ codon reduces the total number of codons that are expressed in the optimal solutions and therefore increases robustness against mutational damage.

At generation 704 a new optimal solution appears - still based on the switchboard gene 5512 but with a mutated core gene 5 of the form $+X + X$. An immediate advantage of this solution is that it uses fewer expressed codons and therefore will be more mutationally robust. However, another complementary and more subtle effect appears: $55 * *$ in the switchboard gene requires a final terminating 0 codon in the first position of the following gene (or some other codon combination, such as $hh0*$, that evaluates to zero) in order to provide an optimal solution. In the initial population, when the $+X + X$ solution is first found, this is provided by gene 1, associated with the switchboard $551*$, as more than 50% of the individuals that used the $+ + +X$ core gene already had a 0 codon in the first position there. However, of the 67 optimal individuals at generation 704 only 5 had a zero first position codon. Three hundred generations later, an examination of genes 4, 6 and 7, which are neither activated by the switchboard nor expressed, show that, from the 246 such genes associated with the 82 optimal individuals, 101, i.e. 41% have a 0 in the first position. This is almost three times the percentage (15%) one would expect if the distribution were random! What is the reason for this extraordinary self-organization - the origin of our somewhat tongue-in-cheek title? The answer is that this evolutionary strategy opens the road to a more stable switchboard gene as from $551*$ a mutation on the switchboard of the third codon to activate any gene that has a 0 codon in first position would result in an optimal string. In this sense many of the non-activated or expressed genes are acting as a genetic "reserve" to protect against potentially deleterious mutations of the switchboard gene.

As in [6] one can summarize the algorithmic "language" that has emerged. It is extremely interesting that this language evolves, in the sense that the system is continually finding "fitter" genes (the "words" of the language) and "fitter" ways of expressing them through the switchboard gene (the "syntax" of the language), where "fitter" means an effective fitness that also measures evolutionary robustness. In Table 1 we give a description of the algorithmic language that has emerged after 1000 generations.

Table 1. Description of the algorithmic language that has emerged by generation 1000

Words of the Language		
Gene	Codon content	Logic
Gene 5	$+X+X$	Gives possibility of at least 2X when expressed once and 4x when expressed twice
Gene 1	$0***$	Terminates the twice expressed $00**5$ gene with 0 codons that do not affect the 4X expression
Genes 4,6,7	$0***$	Genetic Reserve - predominantly not expressed but backup in case of mutation in third switchboard codon
Genes 2,3	$****$	"Non-coding" regions - either do not form part of the language or have unknown functionality
Syntax of the Language		
Switchboard gene	$551*$	Puts producing gene 5, expressed twice, before consuming gene 1 to given optimal ordering

The perspicacious reader might wonder as to why the system based itself on the optimal solution $(+X(+X(+X(+X\ 0))))$, which actually uses more expressed codons - 9 - than the minimal solution $(+X(+X(+X\ X)))$, which uses only 7 and therefore might be surmised to be more evolutionarily robust. The answer is, of course, that because the first solution uses a repeated building block - $+X+X$ - this block may be expressed twice by activating the same genetic locus twice. In this case the effective number of codons that are subject to mutational damage of $(+X(+X(+X(+X\ 0))))$ is only 5 compared to the 7 used by $(+X(+X(+X\ X)))$.

We have discussed at length the evolution of robustness and believe that the above description of the results of the experiment offers unequivocal evidence for it. However, one can determine more rigorous and quantitative, but also somewhat less revealing, measures. If one can think of one optimal solution as being more robust than another then this should imply that the more robust state's "neighbours" are on average fitter, where the notion of neighbour depends on the application of the operator we are thinking of robustness with respect to. For mutation, it is natural to use Hamming distance as a measure of neighbourhood. To this end, for every optimal solution we consider the average fitness of its 96 one-mutant neighbours. In figure 5 we see a graph of this quantity as a function of time. The key observation is: upon discovery of the solution $(+X(+X(+X(+X\ 0))))$ at generation 704, the system now has a solution that uses in the genotype only five distinct codons, other than the switchboard gene, whereas the previous solution, based on $+++X$, used 9 distinct codons. This implies that, all else being equal, as the $+++X$ solution has $(32-13)=19$

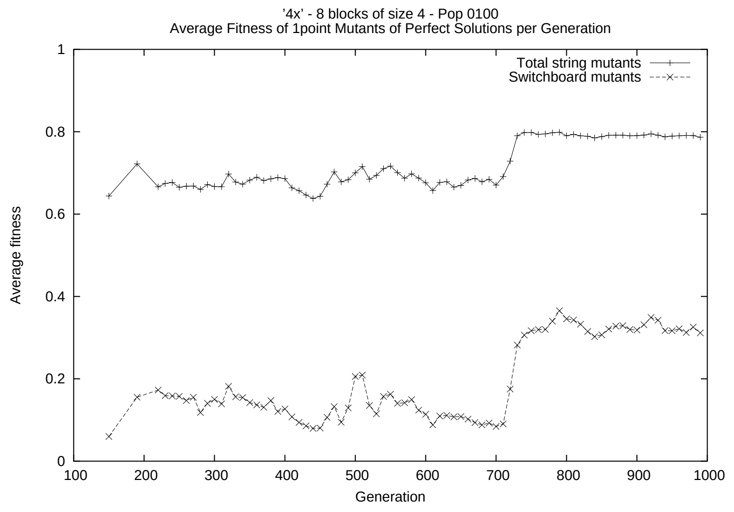

Fig. 5. Average fitness of 1-point mutants of perfect individuals, and of 1-point mutants of each perfect individual's switchboard.

non-expressed codons, the $+X+X$-based solution, which uses 4 less should have $4/19 \sim 20\%$ more optimal one-mutant neighbours which is roughly the increase seen in figure 5.

The evolution of robustness is even more pronounced if we examine the fitness of the one-mutant neigbours of optimal solutions where we consider only the switchboard gene, i.e. only 12 neighbours. In figure 5 we also see the temporal evolution of the average fitness of the one-mutant neighbours of the switchboard. We have also examined the evolution of the ratio of one-point mutants of both the entire genotype and the switchboard that generate optimal and valid solutions respectively. The behaviour is similar to that for fitness, showing marked increases when a more robust solution is found. Naturally, in the case of the switchboard, due to the important role this gene plays in generating the syntax, and the lack of neutral mutations due to the fact that all four codons are used to activate other genes, one notes that the average fitness of the one-mutant neighbours is small. This is a sign that the switchboard is more brittle than many of the other genes. However, it is noteworthy that even in the case where the optimal solution uses 4 expressed genes (before generation 704) there is still some degree of robustness. As the average number of optimal mutants can be as high as 13% this means that up to 2 switchboard codons could be mutated and still leave an optimal individual. This clearly argues for some degree of self-organization. After generation 704 the new class of optimal solution uses only three activated genes hence the fourth codon on the switchboard loses its importance as it is associated with a non-expressed gene. One immediately concludes that a minimum of 25% of the one-mutant neighbours should be optimal. However, it was observed that $27-28\%$ is the norm and hence, that once again, the system had evolved robustness above and beyond just finding a solution that uses fewer expressed codons.

While our conclusions thus far have been gleaned from an examination of a single (though representative) run, all our experiences with other runs suggested that the phenomena we observed are common across a large number of runs. Furthermore, we believe that the structures we have described are sufficiently complex and subtle that the probability that they occurred by chance is negligible. It is however legitimate to ask what happens over many runs. The problem with this is that many of the observed phenomena are contingent. Of course, fit genes in one run will tend to have a similarity with fit genes in another (one can't code $4X$ using a maximum of 12 production rules in too many different ways). However, the switchboard structure, and subtleties such as genetic reserve, will look quite different in different runs. Two basic related phenomena associated with robustness that can be seen over many runs are: the tendency to activate more than once the same gene - especially the core gene - and a tendency to evolve towards solutions that use fewer expressed genes, as can be appreciated in figure 6. There we see that solutions that use two expressed genes occur much more frequently than three-gene solutions, while four-gene solutions are rare indeed. This tendency towards solutions that use fewer expressed genes is the equivalent in this representation of the more familiar phenomena of bloat in standard GP. In both case there is a tendency for the system to evolve to a state where the ratio of coding material to non-coding material is minimized. In GP this is achieved principally by increasing the amount of non-coding material while here is it by minimizing the amount of coding material. Obviously, the payoff is enhanced evolutionary robustness via resistance to mutational damage.

4 Conclusions

In this paper we have investigated how the existence of a degenerate GPM in the form of a simple gene expression network, coupled with a GE-style grammar-based genetic interpreter, can lead to the evolution of robustness and the emergence of an algorithmic language as a result of the self-organization of the GPM.

For the sake of clarity we concentrated on a very simple fitness function and closed grammar. In distinction to previous work, we concentrated on an in depth analysis of a single run as we wished to give an idea of the tremendous subtlety and complexity of the phenomena that can occur, even in this simple situation. We saw that the manner in which an evolutionary system can build robustness can be very varied, from simply developing solutions that require fewer expressed genes, to influencing the content of non-coding parts of the genome and the pattern of gene expression, as was seen, for example, in the creation of genetic reserves. We saw and quantified a tendency to reduce the size of the effective coding region relative to the non-coding region - a phenomena that, in this sense, is analogous to bloat in GP. We saw that robustness can

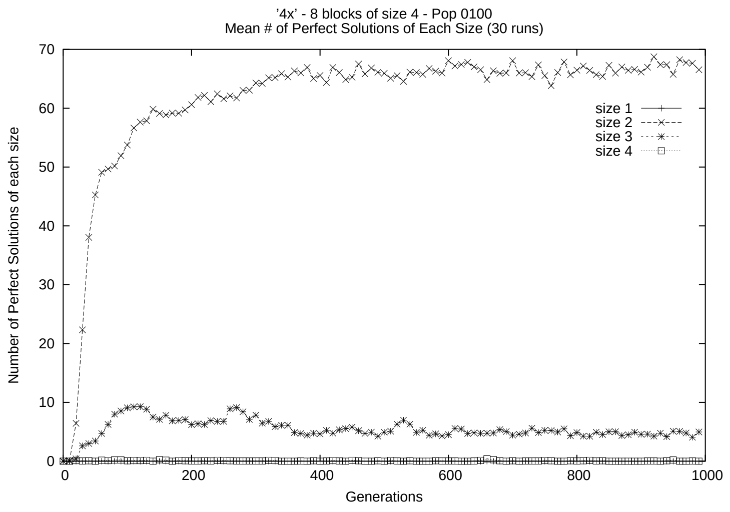

Fig. 6. Average size of perfect solutions per generation (30 runs).

evolve both continuously and in a more punctuated manner, as when passing between solutions with different numbers of expressed genes.

Our study was motivated by a desire to offer a phenomenological predictive framework and description, based on sound principles, of the evolution of robustness in the context of a genetic model that has some language-like features. There exists a formal mathematical framework in which to describe these phenomena - induced symmetry breaking of the genotype-phenotype map and effective fitness as a quantitative measure of this fitness [6]. We will return to a description within this framework at a later date. We believe that further studies of our model and framework will lead to a much deeper understanding of the phenomena of bloat, as well as help in the design of better genetic operators and therefore more competent EAs as advocated by [1]. A further motivation is that of [3] - to understand the origins and evolution of language.

Acknowledgements

CRS is grateful for financial support and hospitality from DIAS and the University of Limerick, a DGAPA sabatical fellowship and Conacyt grant 30422-E.

References

1. Soule, T.: Operator Choice and the Evolution of Robust Solutions. Genetic Programming Theory and Practice I, Kluwer Academic Publishers. (2003) 257-269
2. O'Neill, M. and Ryan, C.: Grammatical Evolution - Evolving programs in an arbitrary language. Kluwer Academic Publishers. (2003)

3. Angeles, O., Stephens, C.R., Waelbroeck, H.: Emergence of Algorithmic Language in Genetic Systems. BioSystems **47**. (1998) 129-147
4. Ryan, C., Keijzer, M., and Nicolau. M.: On the Avoidance of Fruitless Wraps in Grammatical Evolution. In: Cantu-Paz et al., (eds.): Genetic and Evolutionary Computation - GECCO 2003. Springer. (July 2003) 1752-1763
5. Van Nimwegen, E., Crutchfield, J.P., and Huynen, M.: Neutral Evolution of Mutational Robustness. Proc. Natl. Acad. Sci. USA 96. (1996) 9716-9720
6. Stephens, C.R. and Mora, J.: Effective Fitness as an Alternative Paradigm for Evolutionary Computation. Gen. Prog. Evol. Hardware 2. (2000) 7-32.
7. Paterson, N. and Livesy, M.: Evolving caching algorithms in C by genetic programming. In: Genetic Programming 1997: Proceedings of the Second Annual Conference. Morgan Kaufmann. (1997)
8. Keller, R. and Banzhaf, W. : Genetic Programming using Genotype-Phenotype Mapping from Linear Genomes into Linear Phenotypes. In: Genetic Programming 1996: Proceedings of the First Annual Conference. MIT Press. (1996)
9. Whigham, P. : Search Bias, Language Bias, and Genetic Programming In : Genetic Programming 1996. M.I.T. Press. (1996)
10. Wong, M. and Leung, K. Inductive Logic Programming Using Genetic Algorithms. I.I.A.S. (1994)