Advances in Complex Systems, Vol. 5, No.4, 389-408, 2002

GENETIC REPRESENTATION AND GENETIC NEUTRALITY IN GENE EXPRESSION PROGRAMMING

CÂNDIDA FERREIRA
Gepsoft, 37 The Ridings, Bristol BS13 8NU, UK
candidaf@gepsoft.com

Received 1 February 2002
Revised 30 August 2002
Accepted 6 September 2002

The neutral theory of molecular evolution states that the accumulation of neutral mutations in the genome is fundamental for evolution to occur. The genetic representation of gene expression programming, an artificial genotype/phenotype system, not only allows the existence of non-coding regions in the genome where neutral mutations can accumulate but also allows the controlled manipulation of both the number and the extent of these non-coding regions. Therefore, gene expression programming is an ideal artificial system where the neutral theory of evolution can be tested in order to gain some insights into the workings of artificial evolutionary systems. The results presented in this work show beyond any doubt that the existence of neutral regions in the genome is fundamental for evolution to occur efficiently.

Keywords: Genetic neutrality, Gene expression programming, Evolutionary computation

1. Introduction

The evolving entities of many artificial evolutionary systems are simple replicators which survive by virtue of their own properties. The chromosomes of genetic algorithms (GAs) [9] or the parse trees of genetic programming (GP) [12] are examples of simple replicators. The properties of such replicators are the properties of the individual and no development takes place in these systems. This is very different from natural evolutionary systems in which replicators survive by virtue of causal effects on the phenotype [6, 13]. In these systems, replicators and phenotypes are autonomous entities with different functions and properties. Also important is that, in replicator/phenotype systems, a certain degree of development is already present.

No matter how complex the genotype/phenotype system, the development always starts with translation, the transfer of information from the genotype to something else. This "something else" can take different forms like, for instance, the natural RNA or proteins. However, all the immediate products of expression share an important property: their class membership is intrinsic, i.e., the genes of a particular class always produce valid structures of the respective class. Indeed, genotype/phenotype systems with imperfect translation mechanisms are not known to exist in nature. Only in artificial evolutionary systems can these systems be found [3, 18]. In summary, a truly functional genotype/phenotype system must allow a perfect mapping so that evolution could occur smoothly and efficiently.

The automatic evolution of computer programs can also be done smoothly and efficiently provided a genuine genotype/phenotype mapping is created. The creation of such a mapping requires some creative thinking because proteins and computer programs are very different things. Thankfully, computer programs are much easier to understand than proteins and it is not necessary to know, for instance, the rules that determine the three-dimensional structure of proteins to create a simple genotype/phenotype system capable of evolving computer programs. What are, then, the fundamental properties common to the DNA/protein system and to an artificial system especially designed to evolve efficiently computer programs? Obviously, the first is the creation of the genome/program dyad; and second, no matter what, the genome must always produce valid programs. And how can that be accomplished?

Turning to nature for inspiration can help. How does the DNA/protein system cope with complexity? Is the information somehow fragmented in the genome? Maybe the fragmentation of the genome in genes can also be useful in a simple artificial evolutionary system? And what about expression in nature? Is all the information encoded in the genome always expressed? How is it possible to differentiate the information that gets to be expressed from the silenced one? Why is differentiation important? Might this also be of any use in artificial evolutionary systems? Although the answers to all these questions are still being sought, what is known is that, in nature, genomes are vastly redundant, with lots and lots of so called junk DNA which is never expressed: highly repetitive sequences, introns, pseudogenes, and so forth. So, most probably, the introduction of junk sequences in an artificial genome can also be useful.

The genetic representation used in gene expression programming (GEP) explores both the fragmentation of the genome in genes and the existence of junk sequences or non-coding regions in the genome [7]. As Kimura hypothesized [11], the accumulation of neutral mutations plays an important role in evolution. And the non-coding regions of GEP are ideal places for the accumulation of neutral mutations. Thus, in this work, the importance of neutral regions in the genome and, consequently, the importance of neutral mutations in evolution is analyzed using the fully functional genotype/phenotype system of gene expression programming.

2. Genetic Algorithms with Tree Representations

All genetic algorithms use populations of individuals, select individuals according to fitness, and introduce genetic variation using one or more genetic operators (see, e.g., [14]). In recent years different systems have been developed so that these powerful algorithms inspired in natural evolution could be applied to a wide spectrum of problem domains (see, e.g., [14] for a review of recent work on genetic algorithms and [4] for a review of recent work on genetic programming).

Structurally, genetic algorithms can be subdivided in three fundamental groups. i) Genetic algorithms with individuals consisting of linear chromosomes of fixed length devoid of complex expression. In these systems, replicators (chromosomes) survive by virtue of their own properties. The algorithm invented by Holland [9] belongs to this group, and is known as genetic algorithm or GA. ii) Genetic algorithms with individuals consisting of ramified structures of different sizes and shapes and, therefore, capable of assuming a richer number of functionalities. In these systems, replicators (ramified structures) also survive by virtue of their own properties. The algorithm invented by Cramer [5] and later developed by Koza [12] belongs to this group and is known as genetic programming or GP. iii) And genetic algorithms with individuals encoded as linear chromosomes of fixed length which are afterwards expressed as ramified structures of different sizes and shapes. In these systems, replicators (chromosomes) survive by virtue of causal effects on the phenotype (ramified structures). The algorithm invented by myself [7] belongs to this group and is known as gene expression programming or GEP.

This classification not only stresses the fundamental differences between the different kinds of genetic algorithms but also shows clearly the kinship between GEP and GP as both are engaged in the evolution of computer programs as ramified structures. In the next section the fundamental differences between GEP and GP are briefly highlighted.

2.1. *Genetic programming*

As simple replicators, the ramified structures or trees of GP are tied up in their own complexity. On the one hand, bigger, more complex structures become less and less flexible and, therefore, the evolution of more complex structures becomes extremely difficult as these structures cannot be fragmented into smaller, manageable sub-trees. On the other hand, the introduction of genetic variation can only be done at the tree level and, therefore, must be done carefully so that valid structures are created. For instance, the tree-specific crossover illustrated in Figure 1 is practically the only source of genetic variation used in GP for it allows the exchanging of sub-trees and, therefore, always produces valid structures. But the implementation of other operators, like the equivalent of the high-performing point mutation [8], is unproductive as most mutations result in syntactically incorrect structures (Figure 2). Obviously, the implementation of other operators such as transposition or inversion raises similar difficulties and the search space in GP remains vastly unexplored.

2.2. *Gene expression programming*

The phenotype of GEP individuals consists of the same kind of ramified structures used in genetic programming. However, these complex entities are encoded in simpler, linear structures of fixed length – the chromosomes. Thus, there are two main players in GEP: the chromosomes and the ramified structures or expression trees (ETs), being the latter the expression of the genetic information encoded in the former. The transfer of information from the chromosomes to the ETs is called translation. This translation implies obviously a kind of code and a set of rules. The genetic code is very simple: a one-to-one relationship

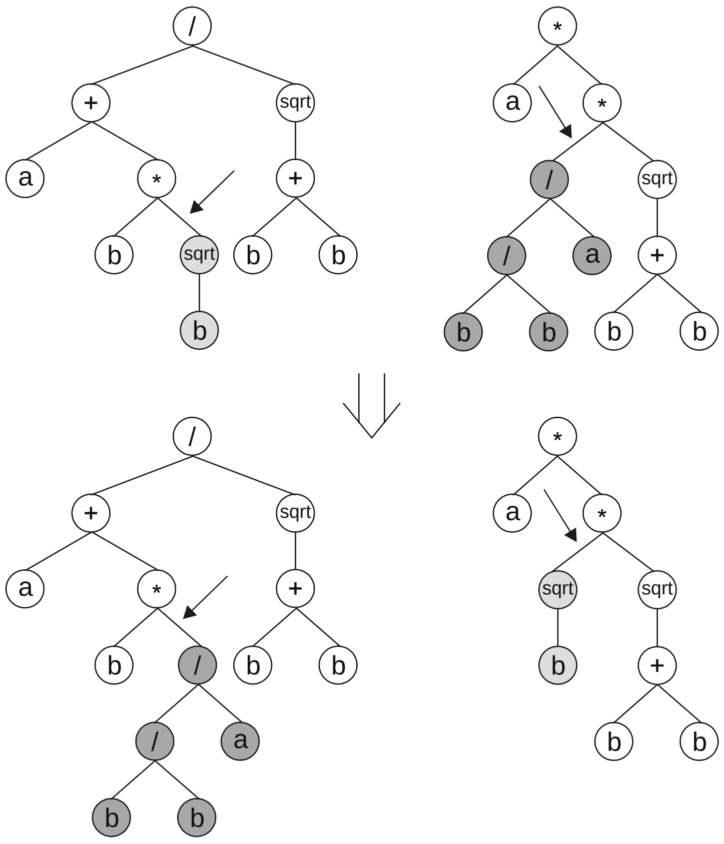


Fig. 1. Tree-specific crossover in genetic programming. Sub-trees in parents are selected and exchanged, forming two new daughter trees.

between the symbols of the chromosome and the functions or terminals they represent. The rules are also very simple: they determine the spatial organization of the functions and terminals in the ETs and the type of interaction between sub-ETs in multigenic individuals.

In GEP there are therefore two languages: the language of genes and the language of ETs and, in this simple replicator/phenotype system, knowing the sequence or structure of one, is knowing the other. In nature, although the inference of the sequence of proteins given the sequence of genes and vice versa is possible, practically nothing is known about the rules that determine the three-dimensional structure of proteins. But in GEP, thanks to the simple rules that determine the structure of ETs and their interactions, it is possible to infer immediately the phenotype given the sequence of a gene, and vice versa. This bilingual and unequivocal system is called *Karva* language. The details of this language are summarized below.

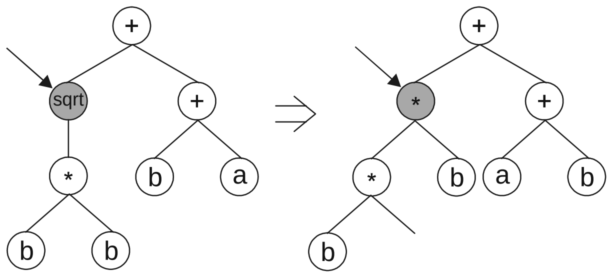


Fig. 2. Illustration of an hypothetical event of point mutation in genetic programming. Note that the daughter tree is an invalid structure.

2.2.1. *Open reading frames and genes*

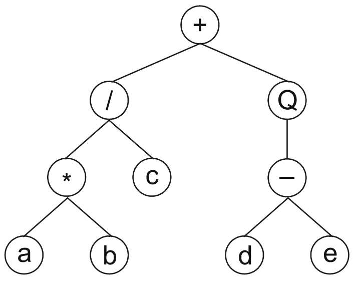
In GEP, the genome or chromosome consists of a linear, symbolic string of fixed length composed of one or more genes. As will next be shown, despite their fixed length, GEP chromosomes code for ETs with different sizes and shapes.

The structural organization of GEP genes is better understood in terms of open reading frames (ORFs). In biology, an ORF or coding sequence of a gene begins with the “start” codon, continues with the amino acid codons, and ends at a termination codon. However, a gene is more than the respective ORF, with sequences upstream of the start codon and sequences downstream of the stop codon. Although in GEP the start site is always the first position of a gene, the termination point does not always coincide with the gene last position. It is common for GEP genes to have non-coding regions downstream of the termination point. (For now these non-coding regions will not be considered because they do not interfere with the product of expression.)

Consider, for example, the algebraic expression:

$$\frac{a \cdot b}{c} + \sqrt{d - e} \tag{2.1}$$

It can also be represented as a diagram or ET:



where “Q” represents the square root function.

This kind of diagram representation is in fact the phenotype of GEP chromosomes, where the genotype is easily inferred from the phenotype as follows:

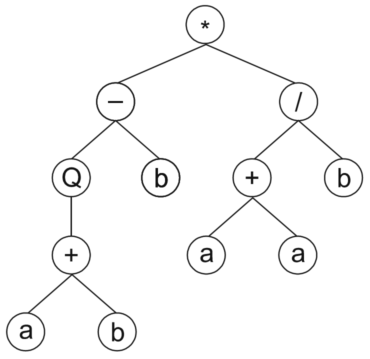
```
0123456789
+/Q*c-abde
```
(2.2)

which is the straightforward reading of the ET from left to right and from top to bottom (exactly as we read a page of text). The expression (2.2) is an ORF, starting at “+” (position 0) and terminating at “e” (position 9). These ORFs are called K-expressions (from Karva notation).

Consider another ORF, the following K-expression:

```
012345678901
*-/Qb+b+aaab
```
(2.3)

Its expression as an ET is also very simple and straightforward. To express correctly the ORF, the rules governing the spatial distribution of functions and terminals must be followed. The start position (position 0) in the ORF corresponds to the root of the ET. Then, below each function are attached as many branches as there are arguments to that function. The assemblage is complete when a baseline composed only of terminals (the variables or constants used in a problem) is formed. So, for the K-expression (2.3) above, the following ET is formed:



Looking at the structure of GEP ORFs only, it is difficult or even impossible to see the advantages of such a representation, except perhaps for its simplicity and elegance. However, when ORFs are analyzed in the context of a gene, the advantages of this representation become obvious. As stated previously, GEP chromosomes have fixed length, and they are composed of one or more genes of equal length. Therefore the length of a gene is also fixed. Thus, in GEP, what varies is not the length of genes but the length of the ORFs. Indeed, the length of an ORF may be equal to or less than the length of the gene. In the first case, the termination point coincides with the end of the gene and, in the latter, the termination point is somewhere upstream of the end of the gene.

In summary, GEP genes usually contain non-coding regions downstream of the termination point and, because of this apparently trivial fact, the genome of GEP individuals can be easily modified using any genetic operator. This means that, for the first time in evolutionary computation, a truly functional genotype/phenotype systems is created in which the search space can be thoroughly explored by a myriad of genetic operators, among which the high-performing point mutation [7, 8]. In fact, the chromosomal organization of GEP individuals allows the easy implementation of genetic operators which always produce valid structures and, therefore, no repair whatsoever is necessary.

The section proceeds with the study of the structural organization of GEP genes in order to show how they invariably code for syntactically correct programs and why they allow the unconstrained application of any genetic operator.

2.2.2. *Structural organization of genes*

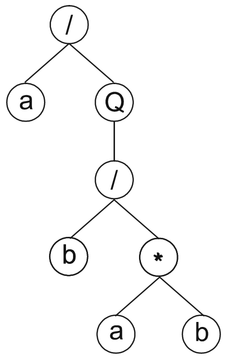
GEP genes are composed of a head and a tail. The head contains symbols that represent both functions and terminals, whereas the tail contains only terminals. For each problem, the length of the head h is chosen, whereas the length of the tail t is a function of h and maximum arity n, and is evaluated by the equation:

$$t = h\,(n\text{-}1) + 1 \qquad (2.4)$$

Consider a gene for which the set of functions consists of F = {Q, *, /, -, +} and the set of terminals T = {a, b}. In this case, n = 2 and if we chose an h = 15, then t = 16. Thus, the length of the gene g is 15 + 16 = 31. One such gene is shown below (the tail is shown in bold):

```
0123456789012345678901234567890
/aQ/b*ab/Qa*b*-ababaababbabbbba
```
(2.5)

It codes for the following ET:

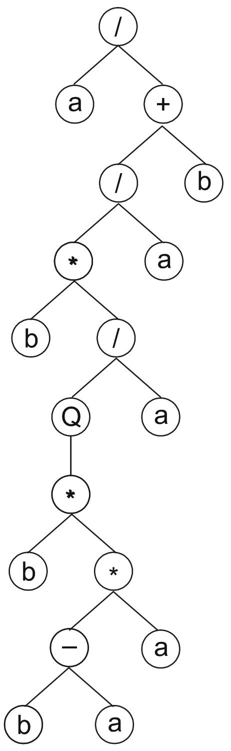


In this case, the ORF ends at position 7, whereas the gene ends at position 30.

Suppose now a mutation occurred at position 2, changing the “Q” into “+”. Then the following gene is obtained:

```
0123456789012345678901234567890
/a+/b*ab/Qa*b*-ababaababbabbbba
```
(2.6)

And its expression gives:

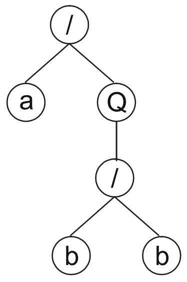


In this case, the termination point shifts 10 positions to the right (position 17).

Obviously the opposite might also happen, and the ORF is shortened. For example, consider again gene (2.5) above, and suppose a mutation occurred at position 5, changing the “*” into “b”, obtaining:

```
0123456789012345678901234567890
/aQ/bbab/Qa*b*-ababaababbabbbba
```
(2.7)

Its expression results in the following ET:



In this case, the ORF ends at position 5, shortening the parental ET in two nodes.

It is worth noticing that any mutation occurring downstream of the termination point of a gene is neutral in effect. For instance, the substitution in chromosome (2.7) of “/” at position 8 by another symbol, be it a function or a terminal, or the substitution of “b” at position 16 by another terminal, are examples of neutral mutations. Note that mutations on the tails also occur, but here a terminal can only be replaced by another terminal in order to maintain the structural organization of genes and therefore guarantee their correct expression.

So, despite their fixed length, GEP genes have the potential to code for ETs of different sizes and shapes, being the simplest composed of only one node (when the first element of a gene is a terminal) and the biggest composed of as many nodes as the length of the gene (when all the elements of the head are functions with maximum arity).

It is evident from the examples above, that any modification made in the genome, no matter how profound, always results in a syntactically correct ET as long as the structural organization of genes is maintained. Indeed, the implementation of high-performing genetic operators in GEP is a child’s play, and Ferreira [7] describes seven: point mutation, IS (from insertion sequence element) and RIS (from root IS element) transposition, two-point and one-point recombination, gene transposition and gene recombination.

2.2.3. *Multigenic chromosomes*

GEP chromosomes are usually composed of more than one gene of equal length. For each problem or run, the number of genes, as well as the length of the head, are *a priori* chosen. Each gene codes for a sub-ET and the sub-ETs interact with one another forming a more complex multi-subunit ET.

Consider, for example, the following chromosome with length 45, composed of three genes (the tails are shown in bold):

```
012345678901234012345678901234012345678901234
Q/*b+Qababaabaa-abQ/*+bababbab**-*bb/babaaaab          (2.8)
```

It has three ORFs, and each ORF codes for a sub-ET (Figure 3). Position 0 marks the start of each gene. The end of each ORF, though, is only evident upon construction of the respective sub-ET. As shown in Figure 3, the first ORF ends at position 8 (sub-ET$_1$); the second ORF ends at position 2 (sub-ET$_2$); and the last ORF ends at position 10 (sub-ET$_3$). Thus, GEP chromosomes contain several ORFs, each ORF coding for a structurally and functionally unique sub-ET. Depending on the problem at hand, these sub-ETs may be selected individually according to their respective fitness (for example, in problems with multiple outputs), or they may form a more complex, multi-subunit ET where individual sub-ETs interact with one another by a particular kind of posttranslational interaction or linking. For instance, when the sub-ETs are algebraic or Boolean expressions, any algebraic or Boolean function with more than one argument can be used to link the sub-ETs in a final, multi-subunit ET. The functions most chosen are addition or multiplication for algebraic sub-ETs, and OR or IF for Boolean sub-ETs.

a)

```
012345678901234012345678901234012345678901234
Q/*b+Qababaabaa-abQ/*+bababbab**-*bb/babaaaab
```

b)

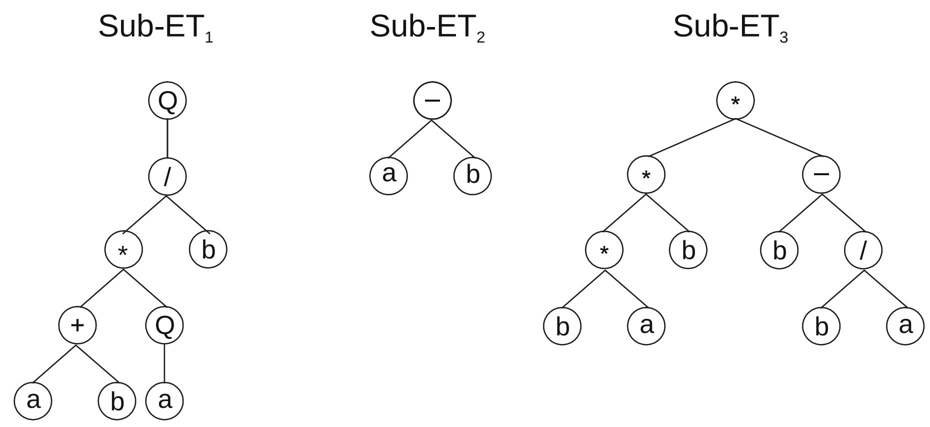


Fig. 3. Expression of GEP genes as sub-ETs. **a)** A three-genic chromosome with the tails shown in bold. Position 0 marks the start of each gene. **b)** The sub-ETs codified by each gene.

Figure 4 illustrates the linking of three sub-ETs by addition. Note that the final ET could be linearly encoded as the following K-expression:

```
012345678901234567890123456789012
++*+-Q*bQ-ba*-*/b/aba*ba/aa+baaab                    (2.9)
```

However, to evolve solutions to complex problems, it is more effective the use of multigenic chromosomes, for they permit the modular construction of complex, hierarchical structures, where each gene codes for a small building block. These small building blocks are separated from each other, and thus can evolve independently. Not surprisingly, these multigenic systems are much more efficient than unigenic ones. Indeed, GEP is effectively a hierarchical invention system capable of discovering simple blocks and using them to form more complex structures.

3. Analyzing the Importance of Genetic Neutrality Using Gene Expression Programming

In order to analyze the importance of genetic neutrality in evolutionary systems, two simple, exactly solved test problems were chosen. These problems can be solved using both unigenic and multigenic systems. On the one hand, the extent of non-coding regions in unigenic systems can be easily increased by increasing the gene length. On the other, in multigenic systems the number of non-coding regions can be increased by increasing the number of genes.

a) `0123456789012340123456789012340123456789012340`

`+bQ**b+`**`bababbbb`**`--b/ba/`**`aaababab`**`*Q*a*-/`**`abaaaaab`**

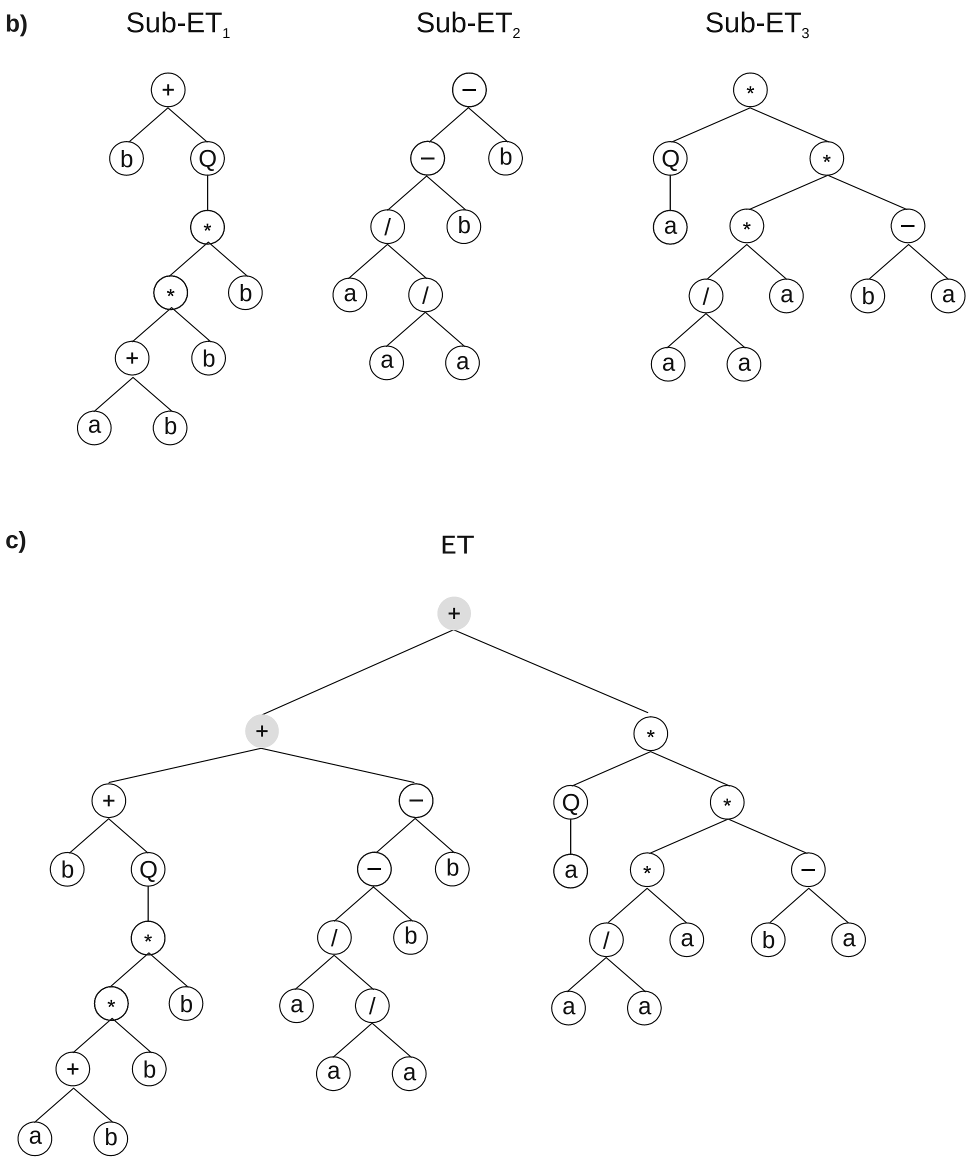


Fig. 4. Expression of multigenic chromosomes as multi-subunit expression trees. **a**) A three-genic chromosome with the tails shown in bold. **b**) The sub-ETs codified by each gene. **c**) The result of posttranslational linking with addition. The linking functions are shown in gray.

3.1. *General settings*

Two problems of symbolic regression were chosen for this analysis. The first is the simple test function:

$$y = a^3 + a^2 + a + 1 \tag{3.1}$$

and the second is the more difficult test sequence:

$$a_n = 4n^4 + 3n^3 + 2n^2 + n \tag{3.2}$$

where *n* consists of the nonnegative integers.

These problems were chosen for three reasons: first, although not trivial, they can be exactly solved by the algorithm and therefore provide an accurate measure of performance in terms of success rate; second, they require relatively small populations and relatively short evolutionary times, making the task feasible; and third, they can be exactly solved using both unigenic and multigenic systems and therefore provide two different approaches for the analysis of genetic neutrality.

For the test function (3.1), a set of 10 random fitness cases chosen from the interval [-10, 10] was used; the fitness function was based on the relative error with a selection range of 25% and a precision error of 0.01%, giving maximum fitness $f_{\max} = 250$ [7]; and population sizes *P* of 30 and evolutionary times *G* of 50 generations were chosen.

For the sequence induction problem, the first 10 positive integers *n* and their corresponding a_n term were used as fitness cases; the fitness function was also based on the relative error with a selection range of 25% and maximum precision (0% error), giving $f_{\max} = 250$; the population size and the evolutionary time were increased, respectively, to 50 and 100 as this problem is slightly more difficult than the function finding one.

In all the experiments, the function set F = {+, -, *, /} and the terminal set T consisted only of the independent variable which was represented by *a*, giving T = {a}; genetic modification was introduced using a mutation rate of 0.03, an IS and RIS transposition rates of 0.1 using three transposons of lengths 1, 2, and 3, and two-point and one-point recombination rates of 0.3; in multigenic systems gene recombination and gene transposition were also used as sources of genetic modification, both at rates of 0.1 and the linking was made by addition; the selection was made by roulette-wheel sampling coupled with simple elitism and the success rate was evaluated over 100 independent runs.

3.2. *Genetic neutrality in unigenic systems*

The importance of genetic neutrality in unigenic systems can be easily analyzed in GEP by increasing the gene length (Figure 5). After finding the most compact organization which allows the discovery of a perfect, extremely parsimonious solution to the problem at hand, any increase in gene length can lead to the evolution of perfect, less parsimonious solutions in which both neutral blocks and non-coding regions might appear.

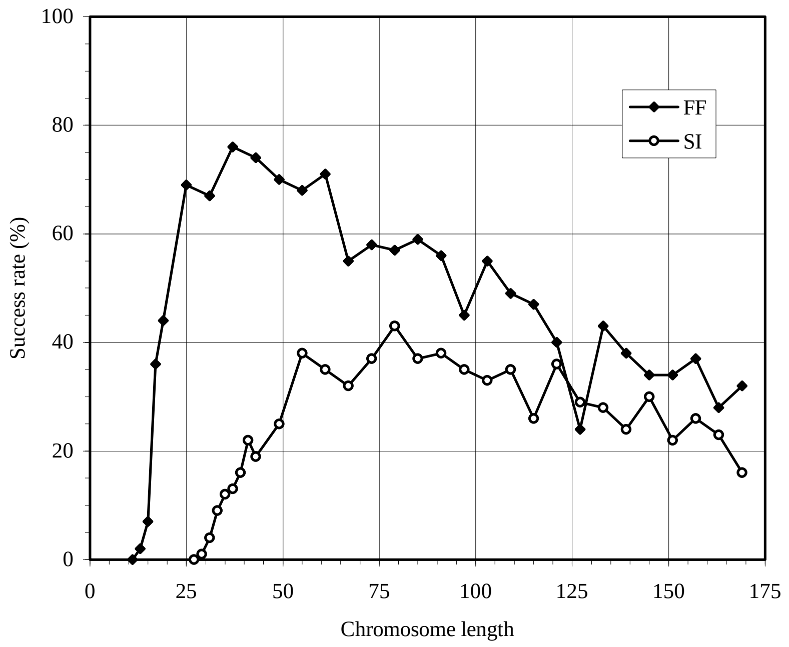


Fig. 5. Variation of success rate with chromosome length for the function finding (FF) and sequence induction (SI) problems.

For problems exactly solved by the algorithm, the most compact organization can be found in most cases. As shown in Figure 5, the test function (3.1) can be compactly encoded using an head length of 6 (corresponding to g = 13). One such solution composed of 13 nodes is shown below:

```
0123456789012
*++*//aaaaaaa
```

Note that, in this case, all the elements of the gene are expressed and therefore no noncoding regions exist.

The test sequence (3.2), however, requires more nodes for its correct and parsimonious expression using the chosen function set. As shown in Figure 5, an h = 14 (corresponding to g = 29) is the minimum head length necessary to solve this problem. The two perfect, parsimonious solutions shown below are expressed using, respectively, 25 and 23 nodes:

```
01234567890123456789012345678
+*aa+*++**++/+aaaaaaaaaaaaaaa

01234567890123456789012345678
**a+a*a++/+/+/aaaaaaaaaaaaaaa
```

Note that the first gene has a small non-coding region composed of four elements, whereas the second has a larger non-coding region with six elements.

As figure 5 emphasizes, the most compact organizations are not the most efficient. For instance, in the function finding problem, the success rate obtained for the most compact organization (*g* = 13) is only 2% whereas the highest success rate obtained for a chromosome length of 37 is 76%. In the sequence induction problem, an identical behavior is observed with a success rate of only 1% for the most compact organization (*g* = 29) and 43% for the best chromosome length (*g* = 79). Therefore, a certain amount of redundancy is fundamental for evolution to occur efficiently. Indeed, in both examples, a plateau was found where the system evolves best. Note also that highly redundant systems adapt, nonetheless, considerably better than highly compact systems, showing that evolutionary systems can cope fairly well with genetic redundancy. For instance, in the function finding experiment, the most redundant system with a chromosome length of 169, has a success rate of 32%, considerably higher than the 2% obtained for the most compact organization with *g* = 13. The same is observed in the sequence induction experiment, where the success rate of the most redundant organization (*g* = 169) is 16% compared to 1% for the most compact organization (*g* = 29).

The structural analysis of compact organizations and less compact ones can also be useful for understanding the role of redundancy in evolution. For instance, the three following perfect solutions to the function (3.1) were discovered using, respectively, head lengths of 6, 18, and 48 (only the K-expressions are shown):

```
(a) 0123456789012
    *++*//aaaaaaa

(b) 0123456789012345678901234567890
    +-*+/+a*/**a*/a++-aaaaaaaaa

(c) 0123456789012345678901234567890123456789...
    /+aa++*+aa+*-a--+-*+*+/**a*a-*--+*/-/-/a...

    ...0123456789012345678901234567890123456
    ...aa//-/*-aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Note that the first solution with an *h* = 6 is expressed using 13 nodes, and therefore the entire gene was used for its expression; the second solution with an *h* = 18 is expressed using 31 nodes, and therefore has a non-coding region with 6 elements; and the last solution with an *h* = 48 uses for its expression only 77 of the 97 elements and therefore has a non-coding region with a length of 20. Note also that not only the length of the non-coding region increases from the most compact to the less compact (0, 6, and 20, respectively) but also increases the number of redundant or neutral motifs. For instance, two neutral motifs using a total of 14 nodes can be found on the medium compact solution and seven neutral motifs involving 38 nodes can be found on the less compact one. This phenomenon is known as code bloat in genetic programming and many have argued about its evolutionary function [1, 2, 15]. Like all kinds of genetic redundancy, neutral motifs are most probably beneficial whenever used in good measure. This can be rigorously determined using GEP, although such an analysis would require lots of time. However, what was learned from the analysis shown in Figure 5 (and also from Figure 6 in the next section) and what is known about the evolution of proteins or technological artifacts suggest an important role for neutral regions in evolution. Their nonexistence or their excess results most probably in an inefficient evolution whereas their existence in good measure, as shown here, is beneficial.

3.3. *Genetic neutrality in multigenic systems*

Another way of analyzing genetic neutrality in GEP, consists of increasing the number of genes. For that a compact, unigenic system capable of exactly solving the problem at hand is chosen as the starting point. Thus, the starting point for the function finding problem is a unigenic system with h = 6 (a chromosome length c of 13), and h = 14 (c = 14) for the sequence induction problem (see Figure 5).

As shown in Figure 6, the results obtained for multigenic systems further reinforce the importance of neutrality in evolution. Note that, in both experiments, the efficiency of the system increases dramatically with the introduction of a second gene and that compact unigenic systems are much less efficient than less compact, multigenic ones. For instance, in the function finding problem, the compact, unigenic system (c = 13) has a success rate

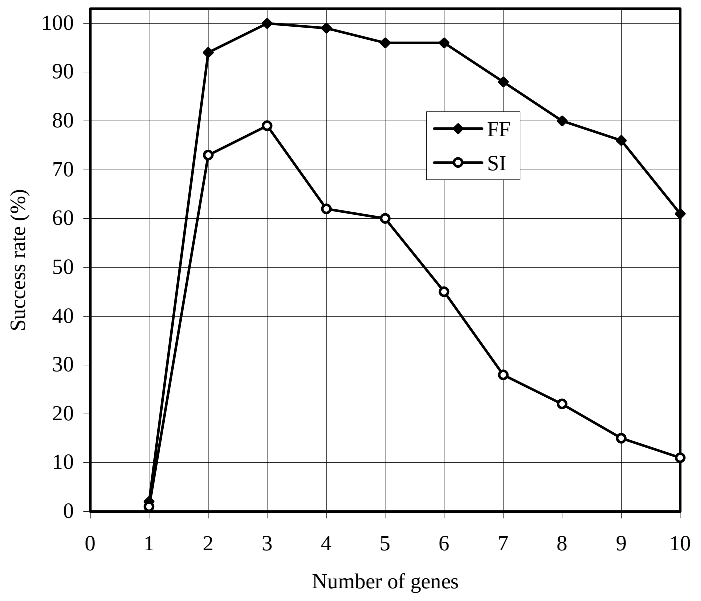


Fig. 6. Variation of success rate with number of genes for the function finding (FF) and sequence induction (SI) problems.

of 2% whereas the success rate in the two-genic system (c = 26) is 94%. In the sequence induction problem, the success rate in the unigenic system (c = 29) is only 1%, whereas in the less compact two-genic system (c = 58) is 73%. Again, a plateau is observed where systems are most efficient, showing that a certain amount of redundancy is fundamental for evolution to occur efficiently. Indeed, it is intuitively understood that a certain room to experiment, not only by forming new building blocks but also by rejecting existing ones, is essential to come up with something new and useful. If there is no room to play, it is only costly (if ever) that one comes up with a good solution to the problem at hand. Note also that highly redundant systems are more efficient than extremely compact ones. For instance, the 10-genic system used in the function finding problem (c = 130), has a success rate of 61% compared to only 2% obtained with the most compact organization (c = 13). The same behavior can be observed in the sequence induction problem where the highly redundant 10-genic system (c = 290) performs slightly better than the extremely compact one (c = 29) (1% and 11%, respectively).

The comparison of Figures 5 and 6 also shows that multigenic systems are considerably better than unigenic ones. For instance, in the function finding problem, from two-genic to six-genic systems (corresponding to chromosome lengths c = 26 through c = 78) the success rates are in all cases above 94% and the best has a success rate of 100% (see Figure 6), whereas the best chromosome length in the unigenic system (c = 37 and h = 18) achieved only 76% (see Figure 5). The same phenomenon can be observed in the sequence induction problem in which from two-genic to five-genic systems (corresponding to c = 58 through c = 145) the success rates are above 60% and the best has a success rate of 79% (see Figure 6) whereas the best chromosome length in the unigenic system (c = 79 and h = 39) achieved only 43% (see Figure 5).

The structural analysis of compact solutions and less compact ones can also provide some insights into the role of redundancy in evolution. For instance, the following three perfect solutions to the sequence induction problem were obtained, respectively, for systems with only one gene, three, and five genes:

```
(a) 01234567890123456789012345678
    **a+a*a++/+/+/aaaaaaaaaaaaaaa

(b) 01234567890123456789012345678
    **a+/*+a//++//aaaaaaaaaaaaaaa
    /*a+/*+a//++//aaaaaaaaaaaaaaa
    -*a+/-*a//++aaaaaaaaaaaaaaaaa

(c) 01234567890123456789012345678
    -aa+*///-+aa/*aaaaaaaaaaaaaaa
    --/*a/+/**/-/-aaaaaaaaaaaaaaa
    +/-*-*a-/a+*a-aaaaaaaaaaaaaaa
    aa/a/*a+*+/+/+aaaaaaaaaaaaaaa
    /-/*aa++**/+/+aaaaaaaaaaaaaaa
```

Note that not only does the total length of the non-coding regions increase from the most compact to the less compact but also does the number of neutral motifs. Specifically, the length of the non-coding region in the unigenic solution is only six and no neutral motifs are present; for the three-genic solution, the total length of the non-coding regions is 16 and three neutral motifs involving a total of 12 nodes can be counted; for the five-genic solution, the non-coding regions encompass already 66 elements and there are six neutral motifs involving a total of 31 nodes.

4. Conclusions

The neutral theory of evolution was tested using the artificial genotype/phenotype system of gene expression programming. GEP provides an ideal framework to conduct such an analysis for three main reasons. First, GEP is a simple artificial life system with a truly functional genotype/phenotype mapping and therefore can provide valuable insights into the workings of any genotype/phenotype system. Second, the number and extent of neutral regions in the genome can be tightly controlled either by increasing the number of genes or the gene length. And third, the high efficiency of the algorithm allows not only the execution of thousands of runs in minutes but also the undertaking of non-trivial tasks with which to make the analysis. Indeed, previous discussions on the importance of neutrality in genetic programming are inconclusive, with some researchers claiming an important role for introns (as neutral motifs are called in GP) and others claiming that introns are an hindrance and must be avoided [1, 2, 15, 17, 20, 21, 22]. How can these contradictory results be explained? Either the conclusions were made using a non-representative number of runs on a trivial task or the simple replicator system of GP is governed by other, still unknown rules. At least for RNA molecules, it has been shown that neutral mutations play an important role in evolution, allowing the diffusion of populations along neutral genotypic networks [10, 16, 19].

In this work, a total of four experiments involving thousands of runs each were made. These experiments show that extremely compact systems with little or no room for neutral regions are significantly less efficient than moderately redundant systems where a considerable part of the genome is engaged in doing “nothing” either by being part of non-coding regions at the end of the ORFs or by encoding neutral motifs that contribute nothing to the individual program. Furthermore, it was also shown that highly redundant systems are, nonetheless, more efficient than extremely compact ones, suggesting that evolutionary systems can cope very well with excessive redundancy. And finally, it was also shown that multigenic systems are more efficient than unigenic ones, suggesting that the fragmentation of the genetic information into smaller units such as genes allows the evolution of more complex programs composed of smaller sub-programs.

Because of their clarity, the results presented in this work are extremely useful for understanding the role of genetic neutrality both in artificial and natural evolution. As shown here, there are two different kinds of neutral regions in GEP: the neutral motifs within the ORFs and the non-coding regions at the end of the ORFs. In simple replicator systems such as GP or GAs, only the former exists whereas in genotype/phenotype systems such as the DNA/protein system or GEP, both kinds exist. And the presence of non-coding regions in genotype/phenotype systems is certainly entangled with the higher efficiency of these systems. For instance, introns in DNA are believed to be excellent targets for crossover, allowing the recombination of different building blocks without their disruption (e.g., [13]). The non-coding regions of GEP can also be used for this purpose and, indeed, whenever the crossover points are chosen within these regions, entire ORFs are exchanged. Furthermore, the non-coding regions of GEP genes are ideal places for the accumulation of neutral mutations that can be later activated and integrated into coding regions. This is an excellent source of genetic variation and certainly contributes to the increase in performance observed in redundant systems.

But, at least in GEP, the non-coding regions play another, much more fundamental role: they allow the modification of the genome by numerous high performing genetic operators. And here by “high performing” I mean genetic operators that always produce valid structures. This problem of valid structures applies only to artificial evolutionary systems for in nature there is no such thing as an invalid protein. How and why the DNA/protein system got this way is not known, but certainly there were selection pressures to get rid of imperfect genotype/phenotype mappings. The fact that the non-coding regions of GEP allow the creation of a perfect genotype/phenotype mapping, further reinforces the importance of neutrality in evolution, as a good mapping is essential for the crossing of the phenotype threshold in evolution.

On the other hand, the reason why neutral motifs within structures, be they parse trees or proteins, can boost evolution, is not so easy to understand, although I think this is another manifestation of the same phenomenon of recombining and testing smaller building blocks. In this case, the building blocks are not entire genes with clear boundaries, but smaller domains within genes. Indeed, in nature, most proteins have numerous variants in which different amino acid substitutions occurred. These amino acid substitutions occur mostly outside the crucial domains of proteins such as the active sites of enzymes and, therefore, the protein variants work equally well or show slight differences in functionality. At the molecular level, these variants constitute the real genetic diversity, that is, the raw material of evolution. The neutral motifs of GEP or GP play exactly the same function, allowing the recombination and testing of different building blocks and, at the same time, allowing the creation of neutral variants that can ultimately diverge and give rise to better adapted structures.

References

[1] Angeline, P. J., Genetic Programming and Emergent Intelligence. In K. E. Kinnear Jr., ed., *Advances in Genetic Programming,* chapter 4, pages 75-98, MIT Press, Cambridge, MA, 1994.

[2] Angeline, P. J., Two Self-adaptive Crossover Operators for Genetic Programming. In P. J. Angeline and K. E. Kinnear, eds., *Advances in Genetic Programming 2,* chapter 5, pages 89-110, MIT Press, Cambridge, MA, 1996.

[3] Banzhaf, W., Genotype-Phenotype-Mapping and Neutral Variation – A Case Study in Genetic Programming. In Y. Davidor, H.-P. Schwefel, and R. Männer, eds., *Parallel Problem Solving from Nature III*, *Lecture Notes in Computer Science*, 866: 322-332, Springer-Verlag, 1994.

[4] Banzhaf, W., P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, 1998.

[5] Cramer, N. L., A Representation for the Adaptive Generation of Simple Sequential Programs. In J. J. Grefenstette, ed., *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Erlbaum, 1985.

[6] Dawkins, R., *River out of Eden*. Weidenfeld and Nicolson, 1995.

[7] Ferreira, C., 2001. Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. *Complex Systems*, 13 (2): 87-129.

[8] Ferreira, C., Mutation, Transposition, and Recombination: An Analysis of the Evolutionary Dynamics. In H. J. Caulfield, S.-H. Chen, H.-D. Cheng, R. Duro, V. Honavar, E. E. Kerre, M. Lu, M. G. Romay, T. K. Shih, D. Ventura, P. P. Wang, Y. Yang, eds., *Proceedings of the 6th Joint Conference on Information Sciences, 4th International Workshop on Frontiers in Evolutionary Algorithms*, pages 614-617, Research Triangle Park, North Carolina, USA, 2002.

[9] Holland, J. H., *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975 (second edition: MIT Press, 1992).

[10] Huynen, M., P. F. Stadler, and W. Fontana, 1996. Smoothness Within Ruggedness: The Role of Neutrality in Adaptation. *Proc. Natl. Acad. Sci. (USA)*, 93: 397-401.

[11] Kimura, M., *The Neutral Theory of Molecular Evolution*, Cambridge University Press, Cambridge, UK, 1983.

[12] Koza, J. R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge, MA: MIT Press, 1992.

[13] Maynard Smith, J. and E. Szathmáry, *The Major Transitions in Evolution*, W. H. Freeman, 1995.

[14] Mitchell, M., *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[15] Nordin, P., F. Francone, and W. Banzhaf, Explicitly Defined Introns and Destructive Crossover in Genetic Programming. In J. P. Rosca, ed., *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 6-22, Tahoe City, CA, 1995.

[16] Reidys, C., P. F. Stadler, and P. Schuster, 1997. Generic Properties of Combinatory Maps: Neutral Networks of RNA Secondary Structures. *Bull. Math. Biol.*, 59: 339-397.

[17] Rosca, J. P., Analysis of Complexity Drift in Genetic Programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 286-294, Stanford University, CA, Morgan Kaufmann, San Francisco, CA, 1997.

[18] Ryan, C., J. J. Collins, and M. O'Neill, Grammatical Evolution: Evolving Programs for an Arbitrary Language. In *Proceedings of the First European Workshop on Genetic Programming, Lecture Notes in Computer Science*, 1391: 83-95, Springer-Verlag, 1998.

[19] Schuster, P., W. Fontana, P. F. Stadler, and I. L. Hofacker, 1994. From Sequences to Shapes and Back: A Case Study in RNA Secondary Structures. *Proc. Roy. Soc. Lond. B*, 255: 279-284.

[20] Soule, T., J. A. Foster, and J. Dickinson, Code Growth in Genetic Programming. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, eds., *Genetic Programming*

1996: Proceedings of the First Annual Conference, pages 215-223, Stanford University, CA, MIT Press, Cambridge, MA, 1996.

[21] Soule, T. and J. A. Foster, Code Size and Depth Flows in Genetic Programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 313-320, Stanford University, CA, Morgan Kaufmann, San Francisco, CA, 1997.

[22] Wineberg, M. and F. Oppacher, The Benefits of Computing with Introns. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, eds., *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 410-415, Stanford University, CA, MIT Press, Cambridge, MA, 1996.