Grammar Defined Introns: An Investigation Into Grammars, Introns, and Bias in Grammatical Evolution.

Michael O'Neill **Conor Ryan** **Miguel Nicolau**
Dept. Of Computer Science & Information Systems
University of Limerick
Ireland.
{Michael.ONeill|Conor.Ryan|Miguel.Nicolau}@ul.ie

Abstract

We describe an investigation into the design of different grammars on Grammatical Evolution. As part of this investigation we introduce introns using the grammar as a mechanism by which they may be incorporated into Grammatical Evolution. We establish that a bias exists towards certain production rules for each non-terminal in the grammar, and propose alternative mechanisms by which this bias may be altered either through the use of introns, or by changing the degeneracy of the genetic code. The benefits of introns for Grammatical Evolution are demonstrated experimentally.

1 Introduction

Grammatical Evolution (GE) is an evolutionary algorithm that can evolve code in any language, using linear genomes [O'Neill & Ryan 2001] [Ryan C., Collins J.J. & O'Neill M. 1998]. We have previously presented results relating to an analysis of some of GE's distinctive features, such as its degenerate genetic code, wrapping operator and crossover [O'Neill & Ryan 1999b] [O'Neill & Ryan 1999a]. We now present the first results from an investigation into the role of the grammar in GE. Specifically, we introduce a mechanism by which introns can be incorporated into the genotypic representation through the grammar, and conduct an analysis on the effects of these grammar defined introns on the performance of GE. We also establish the existence of a bias towards the use of certain production rules for each non-terminal, dependent upon their ordering in the grammar, and propose a mechanism by which this bias can be altered as desired through the use of grammar defined introns.

We begin with a brief overview of GE, for a more complete description we refer the reader to [O'Neill & Ryan 2001]. Grammar defined introns are then introduced, followed by a description of the experimental approach adopted to test the effects of introns, before a discussion on bias and introns.

2 Grammatical Evolution

Unlike standard GP [Koza 1992], GE uses a variable length binary string to represent programs. Each individual contains in its codons (groups of 8 bits) the information to select production rules from a Backus Naur Form (BNF) grammar. BNF is a notation that represents a language in the form of production rules. It is comprised of a set of non-terminals that can be mapped to either elements of the set of terminals, or to elements of the set of non-terminals, according to the production rules. An excerpt from a BNF grammar is given below. These productions state that S can be replaced with any one of `expr`, `if-stmt`, or `loop`.

```
S ::= expr       (0)
    | if-stmt    (1)
    | loop       (2)
```

In order to select a rule in GE, the next codon value on the genome is generated and placed in the following formula:

$$Rule = \text{“}Codon\ Integer\ Value\text{''}$$

$$MOD$$

$$\text{“}Number\ of\ Rules\ for\ this\ non-terminal\text{''}$$

If the next codon integer value was 4, given that we have 3 rules to select from as in the above example, we get $4\ MOD\ 3\ =\ 1$. S will therefore be replaced with the non-terminal `if`-stmt.

Beginning from the left hand side of the genome, codon integer values are generated and used to select rules from the BNF grammar, until one of the following situations arise:

1. A complete program is generated. This occurs when all the non-terminals in the expression being mapped are transformed into elements from the terminal set of the BNF grammar.
2. The end of the genome is reached, in which case the *wrapping* operator is invoked. This results in the return of the genome reading frame to the left hand side of the genome once again. The reading of codons will then continue, unless an upper threshold representing the maximum number of wrapping events has occurred during this individual's mapping process.
3. In the event that a threshold on the number of wrapping events has occurred and the individual is still incompletely mapped, the mapping process is halted, and the individual is assigned the lowest possible fitness value.

GE uses a steady state replacement mechanism, such that two parents produce two children, the best of which replaces the worst individual in the current population if the child has a greater fitness. The standard genetic operators of point mutation, and crossover (one point) are adopted. It also employs a duplication operator that duplicates a random number of codons and inserts these into the penultimate codon position on the genome. A full description of GE can be found in [O'Neill & Ryan 2001].

3 Grammar Defined Introns

The benefit, or otherwise, of introns in evolutionary computation have been hotly debated for some time [Levenick 1991] [Altenberg 1994] [Angeline 1994] [Nordin & Banzhaf 1995] [Nordin, Francone & Banzhaf 1995] [Wu & Lindsay 1995] [Andre & Teller 1996] [Wineberg & Oppacher 1996] [Haynes 1996] [Wu & Lindsay 1996] [Lobo et al. 1998] [Smith & Harries 1998] [Luke 2000]. In the standard implementation of GE, introns can only occur at the end of a chromosome due to the nature of the mapping process. The role of an intron in the preservation of building blocks due to destructive crossover events is therefore minimised in GE. We wish to investigate the effects introns might have on the performance of GE and, as such, have devised a mechanism by which they may be incorporated into the system. We call this mechanism Grammar Defined Introns, whereby the grammar is used to incorporate introns into the genome. This is achieved by allowing codons to be skipped over during the mapping process, by using `introns` as a choice(s) for non-terminals.

For example, the following non-terminal uses an intron as a rule:

```
<line> :: =  <if-statement>      (A)
            |<op>                (B)
            |intron              (C)
```

When a codon evaluates to the `intron` rule being selected we simply skip over this codon, and the code undergoing the mapping is unchanged. In this case the non-terminal `<line>` would remain as `<line>` if the `intron` rule is selected, and the next codon is read.

4 Bias in Grammatical Evolution

When choosing a production rule to be applied to a non-terminal during the mapping process, there is a bias towards certain choices. The amount of bias depends on the number of choices that are to be made, and on the number of genetic codes that are used to represent each choice. Taking the example of the non-terminal `<op>`:

```
<op> :: =    left()        (A)
           | right()       (B)
           | move()        (C)
```

there are 3 possible mappings for `<op>` that can be made in this case. Given a 2-bit codon, there are 4 possible genetic codes representing these choices. This results in a strong bias towards the first choice with a probability of selection of 0.5 as opposed to 0.25 for both of the other rules, see Table 1.

Genetic Code	**Choice**
00	A
01	B
10	C
11	A

Choice	**Probability**
A	2/4
B	1/4
C	1/4

Table 1: Probabilities of selecting a production rule using 2-bit codons.

However, given an 3-bit codon the bias due to the probability of using any one rule is reduced, see Table 2.

Taking the case of an 8-bit codon as adopted in the standard GE implementation this bias is minimised even further, see Table 3.

In the case of there being two choices as in

```
(1) <code> :: =  <line>             (A)
                |<code><line>       (B)
```

there is no bias to either choice no matter how many codes exist.

One approach to alleviate the problem of bias was that used by [Paterson & Livesley], who duplicated certain rules. Unfortunately, that system was difficult to control, and not very

Genetic Code	Choice
000	A
001	B
010	C
011	A
100	B
101	C
110	A
111	B

Choice	Probability
A	3/8 (.375)
B	3/8 (.375)
C	2/8 (.25)

Table 2: Probabilities of selecting a production rule using 3-bit codons.

Choice	Probability
A	86/256 (.336)
B	85/256 (.332)
C	85/256 (.332)

Table 3: Probabilities of selecting a production rule using 8-bit codons.

successful at removing the bias. Another approach that GE can employ is to minimise the bias towards any one rule by increasing the size of the codon.

This paper will consider both the possibility of introducing and removing bias through the incorporation of introns.

5 Experimental Approach

The aim of this paper is to examine bias in the grammar and see if using introns and increasing codon size can be used to alter any bias effects that might be observed. We also wish to establish if introns may be useful to GE.

We conduct our experimentation on the Santa Fe ant trail problem. A tableau describing this problem and parameters can be seen in Table 4. The default grammar used for this problem is outlined below.

$$N = \{code, line, if - statement, op\}$$

$$T = \{left(), right(), move(), food\_ahead(),$$
$$else, if, \{, \}, (,)\}$$

$$S = < code >$$

And P can be represented as:

```
(A) <code> :: =  <line>                 (0)
                |<code><line>           (1)
```

```
(B) <line> :: =  <if-statement>     (0)
                |<op>               (1)

(C) <if-statement> :: = if(food_ahead()){
                           <line>
                        }
                        else{ <line> }

(D) <op> :: =    left()             (0)
               | right()            (1)
               | move()             (2)
```

To determine the effect of introns on the performance of GE, grammar defined introns were placed at various points in the grammar, and the cumulative frequency of success measured on the target problem.

For example, 100 runs were conducted where an intron was placed at position zero of Rule (A) as follows:

```
(A) <code> :: = intron              (0)
               |<line>              (1)
               |<code><line>        (2)
```

100 runs were then conducted with the intron placed at the other two remaining positions:

```
(A) <code> :: = <line>              (0)
               |intron              (1)
               |<code><line>        (2)
```

and,

```
(A) <code> :: = <line>              (0)
               |<code><line>        (1)
               |intron              (2)
```

The same approach was taken for the other two non-terminals involving a choice (i.e. Rules B and D).

To take into account the bias that might result from using a smaller codon size, we repeat the above experiments using a 2-bit codon instead of the 8-bits used normally.

6 Results

Cumulative frequencies of success for each of the experiments outlined in the previous section are given in Figures 1, 2, 3 and 4.

Figure 1 shows results for the insertion of an intron at the various positions of rule A. With the intron in position zero, a success rate superior to standard GE is achieved in the case of both 8-bit and 2-bit codons, with little difference between the 8-bit and 2-bit results. In the cases of positions one and two, it can be seen that the presence of the intron has the similar effect of improving success over standard GE. With the addi-

Objective :	Find a computer program to control an artificial ant so that it can find all 89 pieces of food located on the Santa Fe Trail.
Terminal Operators:	left(), right(), move(), food_ahead()
Fitness cases	One fitness case
Raw Fitness	Number of pieces of food before the ant times out with 615 operations.
Standardised Fitness	Total number of pieces of food less the raw fitness.
Hits	Same as raw fitness.
Wrapper	Standard productions to generate C functions
Parameters	$Population$ = 500, $Generations$ = 50 $pmut$ = 0.01, $pcross$ = 0.9

Table 4: Grammatical Evolution Tableau for the Santa Fe Trail

tion of an intron to Rule A, we change the number of choices from two to three, thus biasing the rule in position zero.

In the case that the intron is in position zero and therefore biased towards (stronger bias in the case of a 2-bit codon) we see a superior performance to standard GE, particularly in the case of a 2-bit codon.

These results would suggest that by inserting bias towards the choice of an intron we achieve an improved performance, comparing to what would otherwise be an unbiased rule choice. When 8-bit codons are adopted (reduction in bias towards the rule at position zero), the improvement in performance by placing an intron at position zero is less evident than in the case of 2-bit codons.

In the case of inserting an intron in positions 1 or 2, we are creating a bias towards the rule in position 0, i.e. $< code >::=< line >$. This also gave us superior performance comparing to standard GE. This seems to suggest by forcibly inserting a bias towards certain rules, we can guide the system to make its choices, thus improving the overall performance.

Similar results are observed for rule B, see Figure 2. The presence of introns generally enhances the performance over standard GE, with positive effects due to the insertion of bias either towards introns, or towards existing rules.

Insertion of an intron into rule D has the opposite effect to insertion into rules A and B, i.e. change from an uneven number of choices (3) to an even number (4), see Figure 3 and 4. With the addition of an intron, the bias towards any one of the production rules is removed. The results demonstrate that with the intron placed at all the positions other than position zero, a reduction in performance over standard GE with 8-bit codons is observed. The change in success rate when placed in position zero appears to be less evident in the case of 8-bit codons, but much larger for 2-bit codons.

6.1 Discussion

These results suggest that it is quite possible for a grammar to implicitly contain bias. This, in turn, can have severe implications for the type and quality of individuals explored by the system.

Previous results [O'Neill & Ryan 1999a] have shown that when degeneracy was removed from the system, the performance dropped dramatically. Indeed, Figures 2 to 4 illustrate just how poorly the 2 bit representation (minimal degeneracy) fares.

While it wasn't clear from earlier work exactly why a degenerate encoding was better, these results suggest that degeneracy acts to remove bias from the search. The performance of the 2 bit representation with bias removed approaches that of the 8 bit representation, but on no occasion does it outperform the 8 bit with bias removed. This suggests that degeneracy is doing more than counteracting bias.

Finally, it is clear from the results that sometimes the removal of bias towards a grammar production rule will not improve performance. This in turn suggests that bias in grammars can guide the system to better choices, thus improving the search for a solution.

These findings are, however, limited to the problem domain examined, and as such, further investigations will be required to determine their generality.

7 Conclusions & Future Work

A technique called Grammar Defined Introns is introduced to incorporate introns into GE. Following a discussion on the bias that exists towards certain production rules of the BNF grammar, we demonstrate that the creation of bias has positive effects in the case of the problem domain and grammar examined here. In particular, bias towards introns has been shown to have beneficial effects, thus suggesting that introns

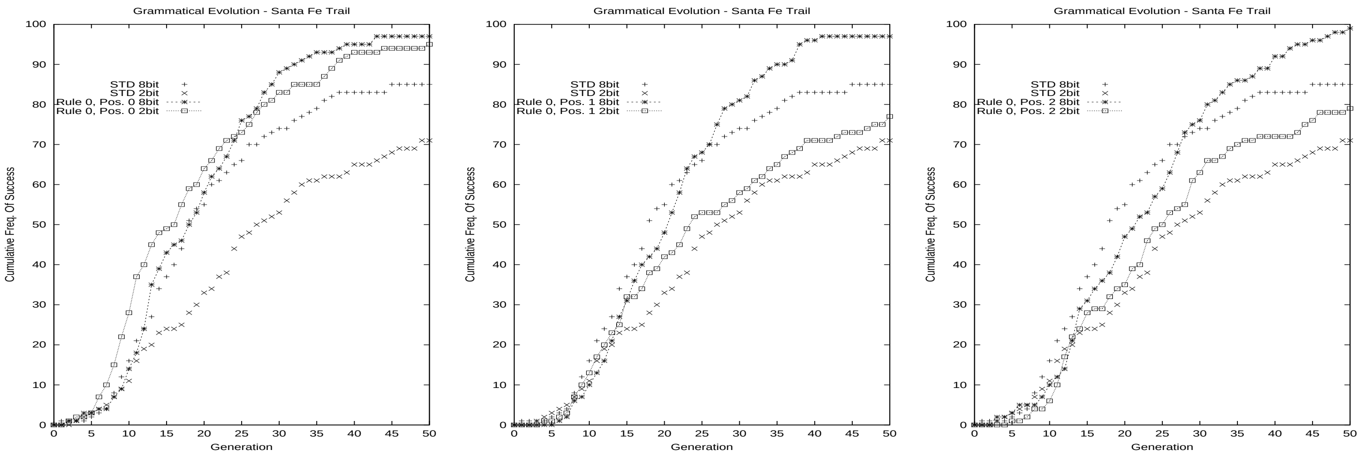

Figure 1: The effects of inserting introns for each choice on the first non-terminal `code`

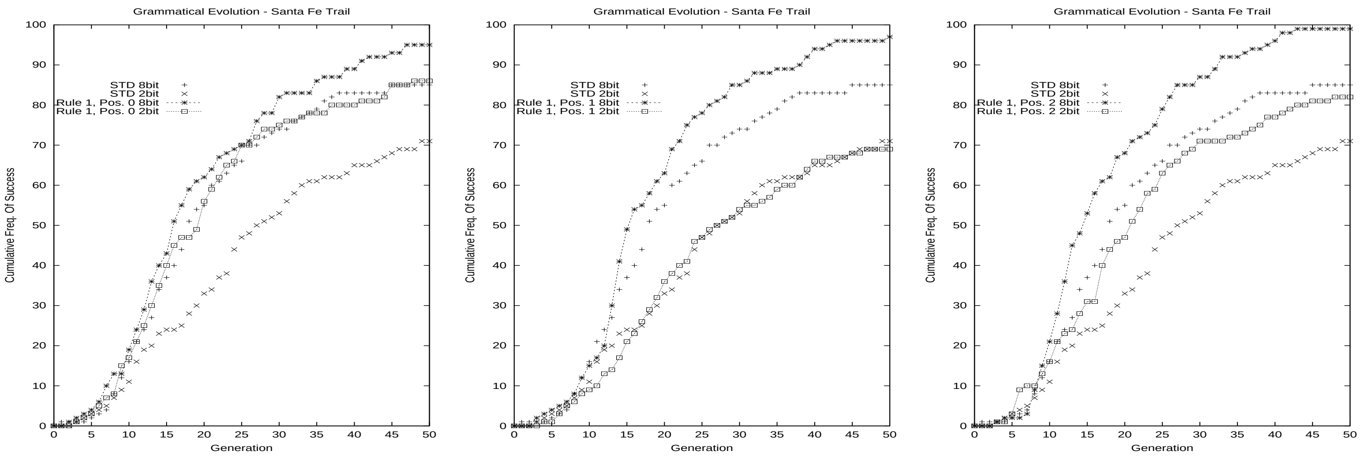

Figure 2: The effects of inserting introns for each choice on the second non-terminal `line`

have a useful role to play in their own right, i.e. in addition to their ability to alter bias towards other production rules.

We show that degeneracy can remove the effect of bias, and that, in many cases, using a degenerate code can outperform a tweaked insertion of introns. In certain cases, a combination of Grammar Defined Introns and degenerate code produces the best performance.

The effect of counteracting bias can be dramatic, and this suggests that much care should be taken in the design of a grammar. Future work will consider the possibility of ideal numbers of productions, and also examine the effects of removing/introducing bias on other problems.

Acknowledgment

The authors wish to thank Maarten Keijzer and Mike Cattolico for the many conversations that helped to form the foundations of this work.

References

[Altenberg 1994] Altenberg L. 1994. The evolution of evolvability in genetic programming. In Kenneth E. Kinnear, Jr., Ed., *Advances in Genetic Programming*. MIT Press, 1994.

[Andre & Teller 1996] Andre D., Teller A. 1996. A Study in Program Response and the Negative Effects of Introns in Genetic Programming. In *Proceedings of Genetic Programming 1996: Proceedings of the First Annual Conference*, John R. Koza, David E. Goldberg, David B. Fogel, & Rick L. Riolo, Eds. Stanford, USA 1996, pp 12-20.

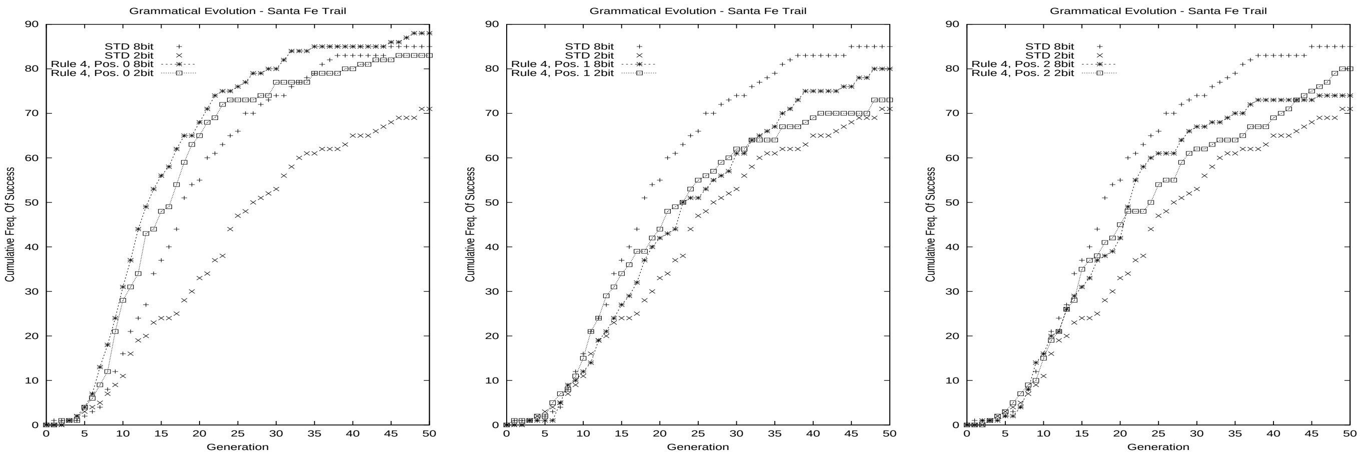

Figure 3: The effects of inserting introns for the first three choices on the fourth non-terminal `op`

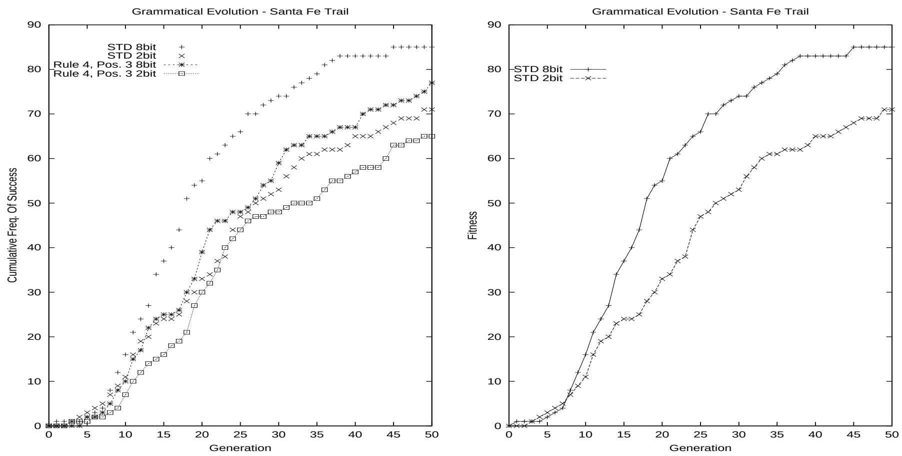

Figure 4: (Left) The effects of inserting introns for the fourth choice on the fourth non-terminal `op` (Right) Results for 2-bit and 8-bit codons using the standard grammar

[Angeline 1994] Angeline P.J. 1994. Genetic Programming and Emergent Intelligence. In Kenneth E. Kinnear, Jr., Ed., *Advances in Genetic Programming*, MIT Press, pp 75-98.

[Goldberg 1989] Goldberg, David E. 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley.

[Koza 1992] Koza, J. 1992. *Genetic Programming*. MIT Press.

[Haynes 1996] Haynes T. 1996. Duplication of Coding Segments in Genetic Programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR, pp 344-349.

[Levenick 1991] Levenick J. R. 1991. Inserting Introns Improves Genetic Algorithm Success Rate: Taking a Cue from Biology. In *Proceedings of the 4th International Conference on Genetic Algorithms*, R.K. Belew and L.B. Booker Eds. San Diego, CA 1991, pp 123-127.

[Lobo et al. 1998] Lobo F.G., Deb K., Goldberg D.E., Harik G., Wang L. 1998. Compressed Introns in a Linkage Learning Genetic Algorithm. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, Madison, Wisconsin, pp 551-558.

[Luke 2000] Luke S. 2000. Code Growth Is Not Caused by Introns. In *GECCO'2000*, Las Vegas, pp

[Nordin & Banzhaf 1995] Nordin P. and Banzhaf W. 1995. Complexity compression and evolution. In *Proceedings of the 6th International Conference on Genetic Algorithms* (ICGA-95), Pittsburgh, L. Eshelman (ed.), Morgan Kaufmann, San Francisco, 1995, pp. 310 - 317.

[Nordin, Francone & Banzhaf 1995] Nordin P., Francone F., and Banzhaf W. 1995. Explicitly defined introns and destructive crossover in genetic programming. In Kenneth

E. Kinnear, Jr. and Peter J. Angeline Eds., *Advances in Genetic Programming 2*. MIT Press.

[O'Neill & Ryan 2001] O'Neill M., Ryan C. Grammatical Evolution. *IEEE Trans. Evolutionary Computation*. 2001.

[O'Neill & Ryan 2000] O'Neill M., Ryan C. 2000. Crossover in Grammatical Evolution: A Smooth Operator? *Lecture Notes in Computer Science 1802, Proceedings of the European Conference on Genetic Programming*, pages 149-162. Springer-Verlag.

[O'Neill & Ryan 1999a] O'Neill M., Ryan C. 1999. Genetic Code Degeneracy: Implications for Grammatical Evolution and Beyond. In *Proceedings of the Fifth European Conference on Artificial Life*.

[O'Neill & Ryan 1999b] O'Neill M., Ryan C. 1999. Under the Hood of Grammatical Evolution. In *Proceedings of the Genetic & Evolutionary Computation Conference 1999*.

[O'Neill & Ryan 1999c] O'Neill M., Ryan C. 1999. Evolving Multi-line Compilable C Programs. *Lecture Notes in Computer Science 1598, Proceedings of the Second European Workshop on Genetic Programming*, pages 83-92. Springer-Verlag.

[Paterson & Livesley] Paterson N., Livesley M. Evolving Caching Algorithms in C by Genetic Programming. In *GP'97: Proceedings of the Second Annual Conference*, pages 262-267.

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

[Smith & Harries 1998] Smith P.W.H., and Harries K. 1998. Code Growth, Explicitly Defined Introns, and Alternative Selection Schemes. *Evolutionary Computation* **6**:4, pp 339-360.

[ICGA Workshop 1997] Workshop on Exploring Non-coding Segments and Genetics-based Encodings, International Conference on Genetic Algorithms 1997, MI, USA.

[Wineberg & Oppacher 1996] Wineberg M. and Oppacher F. 1996. The Benefits of Computing with Introns, In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo Eds., *Genetic Programming 1996: Proceedings of the First Annual Conference*, MIT Press, pages 410-415.

[Wu & Lindsay 1995] Wu A. S. and Lindsay R. K. 1995. Empirical studies of the genetic algorithm with noncoding segments. *Evolutionary Computation* **3**, pp 121-48.

[Wu & Lindsay 1996] Wu A. S and Lindsay R. K. 1996. A survey of intron research in genetics, in *Proceedings of the 4th Conference on Parallel Problem Solving from Nature*, Berlin, Germany, September 1996.