Grammatical Evolution by Grammatical Evolution: The Evolution of Grammar and Genetic Code

Michael O'Neill and Conor Ryan

Biocomputing and Developmental Systems
Dept. Of Computer Science & Information Systems
University of Limerick, Ireland.
Michael.ONeill@ul.ie, Conor.Ryan@ul.ie

Abstract. This study examines the possibility of evolving the grammar that Grammatical Evolution uses to specify the construction of a syntactically correct solution. As the grammar dictates the space of symbols that can be used in a solution, its evolution represents the evolution of the genetic code itself. Results provide evidence to show that the co-evolution of grammar and genetic code with a solution using grammatical evolution is a viable approach.

1 Introduction

This paper details an investigation examining the possibility of evolving the grammar that Grammatical Evolution (GE) [1–4] uses to specify the construction of a syntactically correct solution. By evolving the grammar that GE uses to specify a solution, one can effectively permit the evolution of the genetic code. The ability to evolve genetic code is important when one has little or no information about the problem being solved, or the environment in which a population exists is dynamic in nature and adaptiveness is essential for survival.

Evolutionary automatic programming methods have, to date, largely focused on problem domains that are static in nature, and while this is perfectly adequate for many, there are a significant number of real world problems that have a dynamic component (e.g. scheduling, robot control, prediction, trading. etc.) and require a more adaptive or open-ended representation that can facilitate progression to different environments.

This paper is concerned with the co-evolution of the *genetic code* along with the very individuals that use the code to guide their mapping. The genetic code typically specifies the symbols that are available for incorporation into a solution and can be used to dynamically incorporate bias towards important symbols at different points in time, and this is the mechanism under investigation in this study. In addition, when the genetic code is represented as a grammar, it can also be used to determine how structures may be legally constructed, and any modification of the code modifies the space in which a particular individual searches. Thus, in theory, co-evolution of the grammar and genetic code could

be used to dynamically reduce the search space, or even bias individuals towards different regions of the search space.

There have been two previous studies on the evolution of genetic code in the genetic programming literature using the developmental GP representation [6, 7]. These studies provide strong evidence demonstrating the effective co-evolution of genetic code and solution. The experiments reported here serve a similar objective, that is, to determine if co-evolution of genetic code (or grammar) and solution is possible using grammatical evolution. The distinguishing feature of this study is that the genetic code is represented by a grammar, and a mechanism to evolve the grammar is presented. Discussions on genetic codes, genetic code evolution and its implications are presented for biology [8–10] and for evolutionary computation [11, 12].

The remainder of the paper is structured as follows. Section 2 describes the grammatical approach to grammar evolution, section 3 details the experimental approach adopted and results. Section 4 provides some discussion on the results and comparisons to the evolution of genetic code with the developmental GP approach, and finally section 5 details conclusions and examples of future work that is now possible given the success of this study.

2 Grammatical Evolution by Grammatical Evolution

When we have a set of production rules for a non-terminal, such as, for example, `<op>::= + | -`, a codon is used to select the rule to be applied in the development of sentences in the language specified by the grammar. In a similar manner to a biological genetic code, the productions above represent a degenerate genetic code by which a codon is mapped to a symbol in the output language [1].

In biology, a codon (on mRNA), which is comprised of a group of three nucleotides from the set $\{A, U, G, C\}$, is mapped to an amino acid from the set of 20 naturally occurring amino acids. In nature, the code is encoded in transfer RNA (tRNA) molecules, which have a domino like structure, in that one end matches (with a certain affinity dubbed the *wobble hypothesis*) to a codon, while the amino acid corresponding to this codon is bound to the other end of the tRNA molecule [8]. In this sense, the above productions are equivalent to two such tRNA molecules, one matching a set of codons to + while the other matches a different set of codons to −. By modifying the grammar, we are changing the types of tRNA molecules in our system, or to it put another way, we are directly modifying the genetic code by changing the mapping of codon values to different rules (amino acids).

In order to allow evolution of a grammar, grammatical evolution by grammatical evolution $(GE)^2$, we must provide a grammar to specify the form a grammar can take. This is an example of the richness of the expressiveness of grammars that makes the GE approach so powerful. See [13, 1] for further examples of what can be achieved with grammars. By allowing an evolutionary algorithm to adapt its representation (in this case through the evolution of the genetic code

or grammar) it provides the population with a mechanism to survive in dynamic environments, in particular, and also to automatically incorporate biases into the search process.

In this approach we therefore have two distinct grammars, the *universal grammar* (or grammars' grammar) and the *solution grammar*. The notion of a universal grammar is adopted from linguistics and refers to a universal set of syntactic rules that hold for spoken languages [14]. It has been proposed that during a child's development the universal grammar undergoes modifications through learning that allows the development of communication in their parents native language(s) [15].

In $(GE)^2$, the universal grammar dictates the construction of the solution grammar. Given below are examples of these grammars for solutions that generate expressions, which could be used for symbolic regression type problems.

Universal Grammar (Grammars' Grammar)

```
<g> ::=
      ‘‘<expr> ::= <op> <expr> <expr> | <var>’’
      ‘‘<op> ::=’’ <ops>
      ‘‘<var> ::=’’ <vars>

<ops> ::= <opt> ‘‘|’’ <ops>
      | <opt>

<opt> ::= + | - | * | /

<vars> ::= <vart> ‘‘|’’ <vars>
       | <vart>

<vart> ::= m | v | q | a
```

Solution Grammar

```
<expr> ::= <op> <expr> <expr>
         | <var>

<op> ::= ?

<var> ::= ?
```

In the example universal grammar, a grammar, `<g>`, is specified such that it is possible for the non-terminals `<var>` and `<op>` to have one or more rules, with the potential for rule duplication. These are the rules that will be made available to an individual during mapping, and this effectively allows bias for symbols to be subjected to the processes of evolution. The productions `<vars>` and `<ops>` in the universal grammar are strictly non-terminals, and do not appear in the solution grammar. Instead they are interim values used when producing the solution grammar for an individual.

The hard-coded aspect of the solution grammar can be seen in the example above with the rules for `<op>` and `<var>` as yet unspecified. In this case we have restricted evolution to occur only on the number of productions for `<var>` and `<op>`, although it would be possible to evolve the rules for `<expr>` and even for the entire grammar itself. It is this ability that sets this form of genetic code/grammar evolution apart from previous studies in genetic programming. Notice that each individual has its own solution grammar.

In this study two separate, variable-length, genotypic binary chromosomes were used, the first chromosome to generate the solution grammar from the universal grammar and the second chromosome the solution itself. Crossover operates between homologous chromosomes, that is, the solution grammar chromosome from the first parent recombines with the solution grammar chromosome

from the second parent, with the same occurring for the solution chromosomes. In order for evolution to be successful it must co-evolve both the genetic code and the structure of solutions based on the evolved genetic code.

3 Experiments & Results

In this section a number of proof of concept problems are tackled to illustrate the use of grammar and genetic code evolution in $(GE)^2$. For the experiments that follow 100 independent runs are conducted in each case.

The objective of this study is to determine if grammar and code evolution is possible, rather than to test the efficacy of the approach as a method for symbolic regression, so we have not performed comparisons with other approaches. Moreover, the parameters have been chosen to deliberately slow down the evolutionary process to facilitate the observation of the co-evolution of the grammar/code and solution. The evolutionary parameters are as follows: pairwise tournament selection, generational replacement, bit mutation probability 0.01, one-point crossover probability 0.3, codon duplication probability 0.01. Wrapping is turned off, and codon lengths are initialised in the range [1,10], with a codon size of 8-bits. Fitness is minimisation of the sum of errors over the 100 test cases, and a protected division operator is adopted that returns one in the event of a division by zero.

3.1 Quartic Symbolic Regression

An instance of a symbolic regression problem is tackled in order to verify that it is possible for the co-evolution of a genetic code (or grammar) to occur along with a solution. The target function is $f(m, v, q, a) = a + a^2 + a^3 + a^4$ with the three input variables m,v, and q introducing an element of noise. 100 randomly generated input vectors are created for each call to the target function, with values for each of the four input variables drawn from the range [0,1]. Runs are conducted with a population size of 100, for 100 generations. The progress of evolution toward the target solution can be seen in Fig. 1 with ever decreasing error at successive generations.

Fig. 1 shows the increasing frequency of occurrence of the target solution symbols a, $+$ and $*$. Curiously, after 50 generations the frequency of $*$ is dramatically less than a and $+$, and even less than $/$, even though there are double the number of multiplication symbols in the target solution as there are addition operators. It is not until after this point that we begin to see an increase in the frequency of $*$, which, although it finishes considerably lower than the other two symbols, finishes higher than all others. This could have implications as to how a solution to this problem is constructed, suggesting that firstly terms are added together with the use of multiplication not occurring until much later, perhaps replacing some of the addition operators, or through expansion of terms with the multiplication of a by itself.

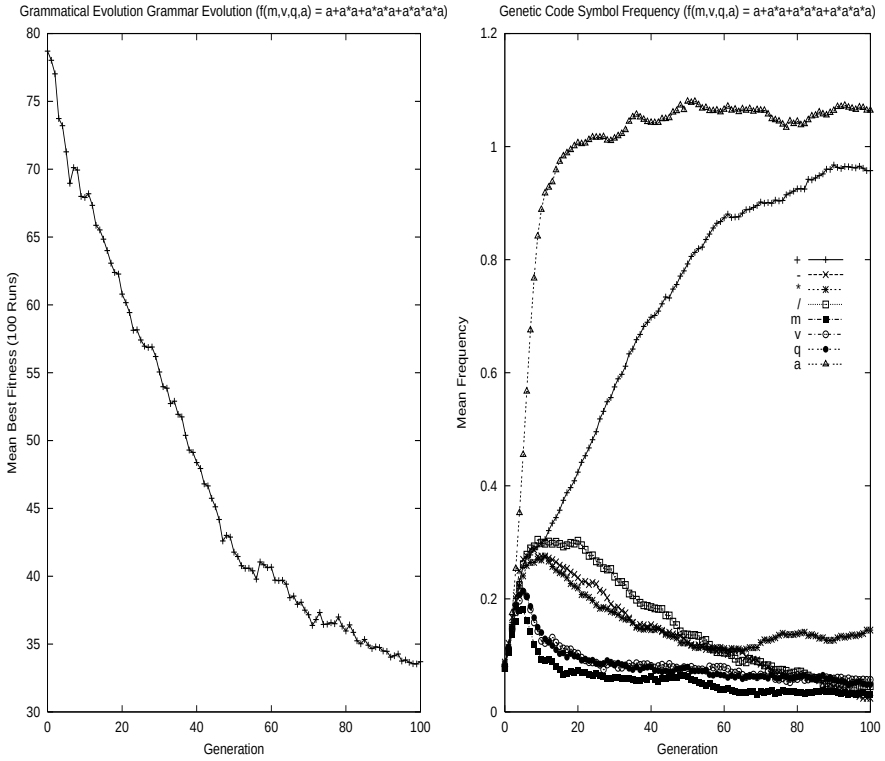

Fig. 1. A plot of the mean best fitness (left) and mean symbol frequency (right) from 100 runs of the quartic symbolic regression problem.

3.2 Dynamic Symbolic Regression I

As indicated in the introduction, dynamic problems are another area in which one could expect to derive some benefit from using evolvable grammars. In this case, one could reasonably expect a system with an evolvable grammar to be able to react more quickly to a change in the environment than a static one could, as a single change in a grammar can reintroduce lost genetic material.

The target functions for the first instance are:

1. $f(m, v, q, a) = a + a^2 + a^3 + a^4$
2. $f(m, v, q, a) = m + m^2 + m^3 + m^4$
3. $f(m, v, q, a) = v + v^2 + v^3 + v^4$
4. $f(m, v, q, a) = q + q^2 + q^3 + q^4$
5. $f(m, v, q, a) = a + a^2 + a^3 + a^4$

The target changes between the functions above every 20 generations. The only difference between each successive function is the variable used. 100 randomly generated input vectors are created for each call to the target function,

with values for each of the four input variables drawn from the range [0,1]. The symbols −, and / are not used in any of the target expressions. Runs were conducted with a population size of 500, for 100 generations, with all other parameters as reported earlier. A plot of the average best fitness and average symbol frequencies can be seen in Fig. 2. A sample of evolved grammars from one of the runs is given below, where in each case the grammar selected is the best solution from the generation just prior to a change in target.

Target 1

```
<op>::= +
<var>::= a
<expression>::= + a a
fitness: 34.6511
```

Target 2

```
<op>::= +
<var>::= m
<expression>::= + m m
fitness: 34.2854
```

Target 3

```
<op>::= + | -
<var>::= v
<expression>::= + v v
fitness: 36.6667
```

Target 4

```
<op>::= + | *
<var>::= q
<expression>::= + + q q * * q q * q q
fitness: 22.8506
```

Target 5

```
<op>::= + | *
<var>::= a
<expression>::= + * a + a a * a a
fitness: 7.85477
```

Table 1. Statistics for both the static and evolvable grammars on the first dynamic problem instance. Lower scores indicate better performance.

Fitness	mean	median	std. dev	signif.
Case	fixed(dynamic)	fixed(dynamic)	fixed(dynamic)	
1	37.33 (40.55)	37.75 (38.22)	7.81 (10.082)	Yes
2	35.48 (36.08)	37.1 (36.57)	6.35 (8.73)	No
3	34.26 (31.53)	36.6 (36.48)	7.54 (10.79)	Yes
4	35.39 (28.74)	37.2 (35.08)	7.96 (12.46)	Yes
5	20.05 (15.1)	22.00 (20.54)	5.99 (10.17)	Yes

The results presented suggest that, when using dynamic grammars, it is possible to successfully preserve and improve solution structure, while still being able to learn appropriate terminal values. This is reflected in the fitness plot where, when the fitness function changes, in most cases there is a decrease in solution fitness for a short period when solutions adjust to the new variable adopted. Later on in the simulations we reach the point where the structure becomes closer to the target and changes in variables alone no longer confer as much damage to fitness, which is again illustrated in the fitness plot (Fig. 2).

A performance comparison of the dynamic and static equivalent of the grammar (given below) for this problem is presented in Fig. 2 and corresponding statistics can be found in Table 1.

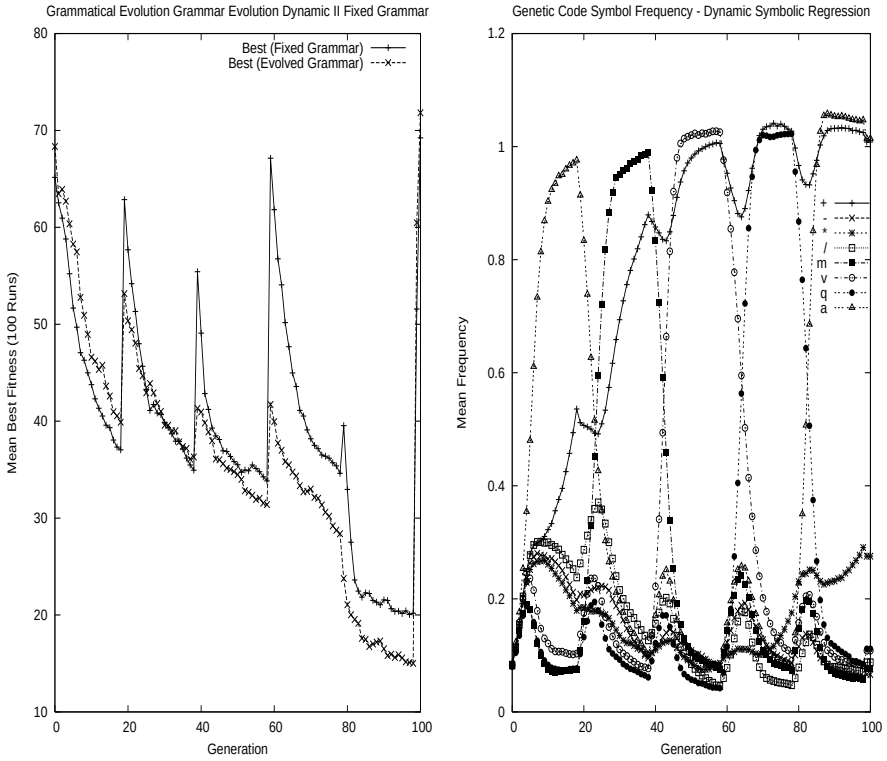

Fig. 2. Plot of the mean best fitness over 100 generations on the first dynamic symbolic regression instance with both static and dynamic grammars (left). Symbol frequency plot (right).

```
<expr> ::= <op> <expr> <expr> | <var>

<op> ::= + | - | * | /

<var> ::= m | v | q | a
```

3.3 Dynamic Symbolic Regression II

The target functions for the second dynamic symbolic regression problem instance are:

1. $f(m, v, q, a) = a + a^2 + a^3 + a^4$
2. $f(m, v, q, a) = m + a^2 + a^3 + a^4$
3. $f(m, v, q, a) = m + m^2 + a^3 + a^4$
4. $f(m, v, q, a) = m + m^2 + m^3 + a^4$
5. $f(m, v, q, a) = m + m^2 + m^3 + m^4$

The target changes between the functions above every 20 generations. The transition used in this problem differs from the previous in that only one term

changes each time. However, the change is larger each time (because the power that the new term is raised to increases). 100 randomly generated input vectors are created for each call to the target function, with values for each of the four input variables drawn from the range [0,1]. The symbols q, v, $-$, and / are not used in any of the target expressions. As in the previous dynamic symbolic regression problem instance runs are conducted with a population size of 500, for 100 generations, with all other parameters as per the standard values reported earlier. A plot of the average best fitness and average symbol frequencies can be seen in Fig. 3.

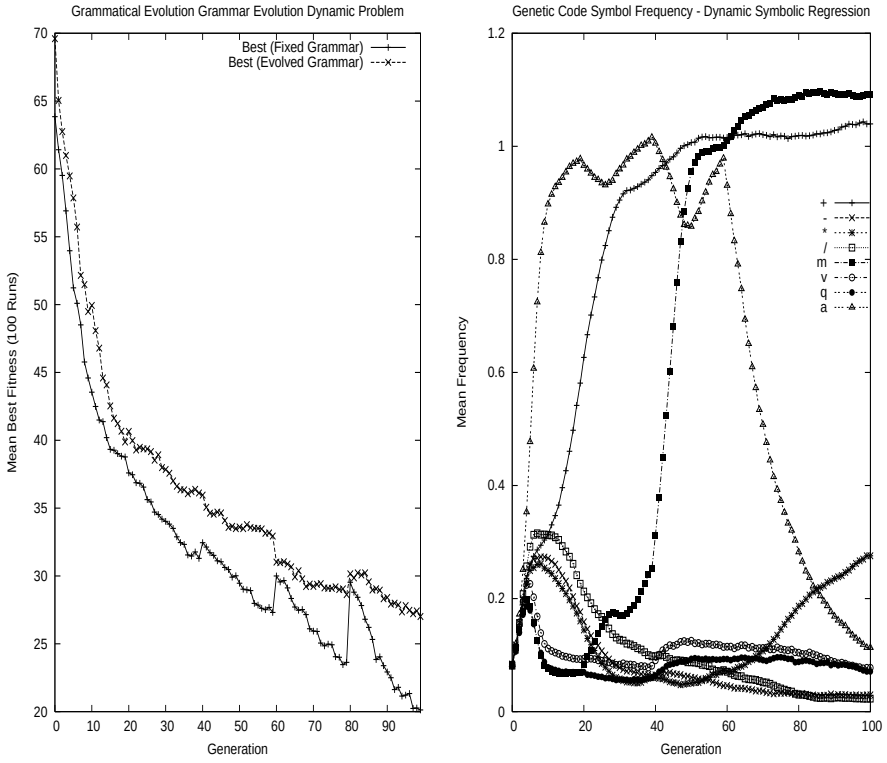

Fig. 3. Plot of the mean best fitness over 100 generations on the second dynamic symbolic regression instance with both dynamic and static grammars (left), and the mean symbol frequency (right).

It is interesting to note that fitness keeps improving over time for the evolvable grammar, with an occasional deterioration corresponding with a change in the fitness function. Notice how the disimprovement is more pronounced later in the runs, particularly for the static grammar, which is due to higher powers being exchanged. These results suggest that the evolvable grammar is more

adaptable in scenarios with larger changes facilitating smoother transitions to successive targets. Also evident from Fig. 3 is the manner in which the quantity of a in the population decreases over time while that of m increases. The two plots intersect at around generation 42, shortly after the target has changed to $f(m, v, q, a) = m + m^2 + a^3 + a^4$. However, the plots remain very close until around generation 60, at which time m^3 becomes part of the solution.

A sample of evolved grammars from one of the runs is given below, where the grammars presented represent the best solution at the generation just prior to each fitness change.

Target 1

```
<op>::= + | +
<var>::= a
<expression>::= (+ a a)
fitness: 37.4525
```

Target 2

```
<op>::= +
<var>::= m | a
<expression> = (+ a m)
fitness: 33.8423
```

Target 3

```
<op>::= + | *
<var>::= m | a
<expression> = (+ a (+ m (* a m)))
fitness: 22.9743
```

Target 4

```
<op>::= + | *
<var>::= m
<expression> = (+ m (* (+ (* m m) m ) m) )
fitness: 15.6311
```

Target 5

```
<op>::= + | *
<var>::= m
<expression>::= (+ (* (+ m (* m (+ m (* m m) m) m) ) ) )
fitness: 4.57967e-15
```

Table 2. Statistics for the second dynamic problem instance. Lower numbers indicate a better fitness.

Fitness	mean	median	std. dev	signif.
Case	fixed(dynamic)	fixed(dynamic)	fixed(dynamic)	
1	39.27 (41.63)	37.98 (38.65)	9.18 (12.59)	No
2	31.55 (36.06)	31.93 (36.60)	6.77 (3.84)	Yes
3	27.62 (33.46)	25.82 (34.52)	6.3 (4.1)	Yes
4	24.05 (29.2)	22.62 (32.17)	5.83 (6.39)	Yes
5	21.34 (27.47)	18.74 (35.2)	11.42 (14.94)	Yes

A performance comparison of the dynamic and static equivalent of the grammar (static grammar as per earlier dynamic problem instance) for this problem is presented in Fig. 3 and corresponding statistics can be found in Table 2. In this case the static grammar outperforms the evolving grammar in terms of best fitness values achieved for all targets but the first. With the evolving grammar there is, as usual, a warm up period where a suitable grammar must be adopted before good solutions can be found. When successive targets are very similar to previous ones this almost negates the potential benefits that a more adaptive representation can bring, as in the case of the evolvable grammars. Clearly, some

dynamic problems are more dynamic than others [16], especially in terms of the degree of change. Previous work [17] on genetic algorithms applied to dynamic problems has shown that, when the change is relatively small, a standard GA with high mutation can handle those types of problems. We believe it is likely to be the same for genetic programming.

These results would also lend support to the idea of introducing different operator rates on the grammar chromosome to the solution chromosome, allowing the population to converge towards a similar grammar, facilitating the exploration of solutions based on a similar grammar. If these rates were adaptable, then it may be possible to allow grammars to change more often if the target changes are large, and vice versa.

4 Discussion

In addition to the problems reported in this paper, we tackled two symbolic regression problems taken from the literature on genetic code evolution, where one is a very simple function [6], and the other extremely difficult [7]. For space reasons we have not reported the details of these experiments here, but the results were positive, clearly demonstrating successful grammar/genetic code and solution co-evolution, showing similar trends to those observed for the static quartic symbolic regression instance.

There are a number of differences between this study on genetic code evolution to the Keller & Banzhaf studies [6, 7] that are largely representation dependent. These include:

- Variable-length genotypes are adopted with GE as opposed to fixed length in the earlier studies.
- Genetic codes are not seeded at the first generation to be equivalent as was the case for developmental GP; an individuals' binary string is initialised randomly in this case, and thus the genetic code is randomly generated. For developmental GP the code was set such that − was the only symbol represented initially, and thus fitness of an individual was at the lowest possible value.
- The same mutation rate is adopted for the genetic code as for the solutions, whereas independent mutation rates were used in the previous studies. Separate rates of mutation were adopted previously as it was hypothesised that in order for successful evolution to occur changes to the genetic code should occur at a slower rate than a solution, with several different individuals having the same or similar genetic codes at any one point in time. With the current setup of two independent chromosomes it would be possible to implement separate mutation rates for each chromosome, this being an avenue for further investigation.
- Crossover is adopted in this study, which was not present previously.

Despite these differences, the results presented here support the findings of the earlier studies, providing further evidence to support the claim that the co-evolution of genetic code/grammar and solution is possible.

5 Conclusions & Future Work

This study demonstrates the feasibility of the evolution of grammatical evolution's grammar on a number of symbolic regression problem instances. In particular, the results demonstrate the ability of grammatical evolution to learn the importance of the various terminal symbols, and thus the ability to dynamically evolve bias toward individual symbols over the course of a run.

This study opens the door to a number of exciting areas of future investigation using $(GE)^2$, and a sample of possible directions follows.

In this study we have focused on symbolic regression problems, and to test generality beyond symbolic regression it is our intention to extend grammar evolution to other problem domains.

Evolving the genetic code through grammar evolution brings the distinguishing ability to evolve both symbol coding rules (e.g. `<op>` and `<var>` as used in this study) and structural rules (e.g. `<expr>`). In this way it would also be possible to evolve biases towards specific structural configurations of the evolving programs, and also to evolve the complete grammar including the number and type of nonterminals.

The ability to evolve the grammar initially input to grammatical evolution opens up the exploration of a more open-ended form of evolution. For example, it is now possible to dynamically define parameterised functions incorporating their specification into the grammar. A static approach to function definition has been previously tackled [18], however, with the ability to evolve the number of functions along with their respective parameters, outputs and data types, this would represent a powerful extension to grammatical evolution, allowing the dynamic modularisation of code and as a consequence improving its scalability.

In a similar manner to the use of dynamically defined functions using grammar evolution, it would also be possible to extend our earlier investigations on constant generation techniques [19] through the provision of various grammatically based constant generation strategies in the universal grammar. The appropriate strategy could then be incorporated into the grammar and evolved. Investigations are currently underway in each of these directions.

Acknowledgements

The authors would like to thank the various members of the Biocomputing and Developmental Systems Group at the University of Limerick for a number of interesting discussions on this work, and Wolfgang Banzhaf for an insightful discussion on an early draft of this work.

References

1. O'Neill, M., Ryan, C. (2003). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers.

2. O'Neill, M. (2001). Automatic Programming in an Arbitrary Language: Evolving Programs in Grammatical Evolution. PhD thesis, University of Limerick.
3. O'Neill, M., Ryan, C. (2001). Grammatical Evolution, *IEEE Trans. Evolutionary Computation*. 2001.
4. Ryan, C., Collins, J.J., O'Neill, M. (1998). Grammatical Evolution: Evolving Programs for an Arbitrary Language.*Proc. of the First European Workshop on GP*, 83-95, Springer-Verlag.
5. Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.
6. Keller, R., Banzhaf W. (1999). The Evolution of Genetic Code in Genetic Programming. In *Proc. of GECCO '99*, vol. 2, 1077-1082, Morgan Kaufmann.
7. Keller, R., Banzhaf W. (2001). Evolution of Genetic Code on a Hard Problem. In *Proc. of GECCO 2001*, 50-56, Morgan Kaufmann.
8. Lewin, B. (2000). *Genes VII*. Oxford University Press, 2000.
9. Sella, G., Ardell, D. H. (2001). The Coevolution of Genes and the Genetic Code. *Santa Fe Institute Working Paper* 01-03-015, February 2001.
10. Ardell, D. H.,Sella, G. (2001). On the Evolution of Redundancy in Genetic Codes. *Journal of Molecular Evolution*, 53:269-281.
11. Kargupta, H., Ghosh, S. Toward Machine Learning Through Genetic Code-Like Transformations. *Genetic Programming and Evolvable Machines*, Vol. 3, No. 3., pp.231-258, September 2002.
12. Freeland, S. J. (2002). The Darwinian Genetic Code: An Adaptation for Adapting? *Genetic Programming and Evolvable Machines*, Vol. 3, No. 2., pp.113-128.
13. Ryan, C., O'Neill, M. (2002). How to do anything with Grammars. *Proc. of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference 2002*, pp. 116-119.
14. Chomsky, N. (1975). *Reflections on Language*. Pantheon Books. New York.
15. Pinker, S. (1995). *The language instinct: the new science of language and the mind*. Penguin, 1995.
16. Ryan, C., Collins, J.J., Wallin, D. (2003). Non-stationary Function Optimization using Polygenic Inheritance. In Foster et al. (Eds). *GECCO 2003: Proceedings of Genetic and Evolutionary Computation Conference*. Springer-Verlag.
17. K. Ng and K. Wong. (1995). A new diploid scheme and dominance change mechanism for non-stationary function optimisation. In *Proceedings of ICGA-5*, 1995.
18. O'Neill, M., Ryan, C. (2000). Grammar based function definition in Grammatical Evolution. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pp. 485-490, Las Vegas, USA. Morgan Kaufmann.
19. O'Neill, M., Dempsey, I., Brabazon, A., Ryan, C. (2003). Analysis of a Digit Concatenation Approach to Constant Generation. In LNCS 2610, *Proceedings of EuroGP 2003, the 6th European Conference on Genetic Programming*, Essex, UK, April 2003. pp. 173-183.