mGGA: The meta-Grammar Genetic Algorithm

Michael O'Neill[1] and Anthony Brabazon[2]

[1] Department of Computer Science and Information Systems
University of Limerick, Ireland
Michael.ONeill@ul.ie
[2] Department of Accountancy
University College Dublin, Ireland
Anthony.Brabazon@ucd.ie

Abstract. A novel Grammatical Genetic Algorithm, the meta-Grammar Genetic Algorithm (mGGA) is presented. The mGGA borrows a grammatical representation and the ideas of modularity and reuse from Genetic Programming, and in particular an evolvable grammar representation from Grammatical Evolution by Grammatical Evolution. We demonstrate its application to a number of benchmark problems where significant performance gains are achieved when compared to static grammars.

1 Introduction

The objectives of this study are to investigate the adoption of principles from Genetic Programming [1] such as modularity and reuse (see Chapter 16 in [2]) for application to Genetic Algorithms, and to couple these to an adaptive representation that allows the type and usage of these principles to be evolved through the use of evolvable grammars. The goal being the development of an evolutionary algorithm with good scaling characteristics, and an adaptable representation that will facilitate it's application to dynamic problem environments. To this end a grammar-based Genetic Programming approach is adopted, in which the grammars represent the construction of syntactically correct phenotypes of the Evolutionary Algorithm.

The remainder of the paper is structured as follows. Section's 2 and 3 describes the grammatical approach to Genetic Algorithms, section 4 details the experimental approach adopted and results, and finally section 5 details conclusions and future work.

2 Grammatical Evolution by Grammatical Evolution

The grammar-based Genetic Programming approach upon which this study is based is the Grammatical Evolution by Grammatical Evolution algorithm [3], which is in turn based on the Grammatical Evolution algorithm [4–7]. This is a meta-Grammar Evolutionary Algorithm in which the input grammar is used to specify the construction of another syntactically correct grammar. The generated grammar is then used in a mapping process to construct a solution.

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 [4, 8, 9] for further examples of what can be represented with grammars, and [10] for an alternative approach to grammar evolution. By allowing an Evolutionary Algorithm to adapt its representation (in this case through the evolution of the 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 case we can allow the meta-Grammar Genetic Algorithm to evolve biases towards different building blocks of varying sizes.

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 [11]. 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) [12].

In $(GE)^2$, the universal grammar dictates the construction of the 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 meta-Grammar and the structure of solutions based on the evolved meta-Grammar.

3 meta-Grammars for Bitstrings

A simple grammar (referred to as `GE`) for a fixed-length (example contains 8 bits) binary string individual of a Genetic Algorithm is provided below. In the generative grammar each bit position (denoted as `<bit>`) can become either of the boolean values. A standard variable-length Grammatical Evolution individual can then be allowed to specify what each bit value will be by selecting the appropriate `<bit>` production rule for each position in the `<bitstring>`.

```
<bitstring> ::= <bit><bit><bit><bit><bit><bit><bit><bit>

<bit> ::= 1 | 0
```

The above grammar can be extended to incorporate the reuse of groups of bits (building blocks). In this example all building blocks that are mutliples of two are provided, although it would be possible to create a grammar that adopted more complex arrangements of building blocks. We refer to this grammar as `GE+BB`.

```
<bitstring> ::= <bbk4><bbk4>
              | <bbk2><bbk2><bbk2><bbk2>
              | <bbk1><bbk1><bbk1t><bbk1><bbk1><bbk1><bbk1><bbk1>

<bbk4> ::= <bit><bit><bit><bit>

<bbk2> ::= <bit><bit>

<bbk1> ::= <bit>

<bit> ::= 1 | 0
```

The above grammars are static, and as such can only allow one building block of size four and of size two in the second example. It would be nice to allow evolution to find a number of building blocks of any one size from which a Grammatical Evolution individual could choose from. This would facilitate the application of such a Grammatical GA to:

- problems with more than one building block type for each building block size,
- to search on one building block while maintaining a *reasonable* temporary building block solution,
- and to be able to switch between building blocks in the case of dynamic environments.

All of this can be achieved through the adoption of meta-Grammars as were adopted earlier in [3]. An example of such a grammar (referred to as `GEGE+BB`) for an 8-bit individual is given below.

```
<g> ::=
       "<bitstring> ::=" <reps>
            "<bbk4> ::=" <bbk4t>
            "<bbk2> ::=" <bbk2t>
            "<bbk1> ::=" <bbk1t>
             "<bit> ::=" <val>

<bbk4t> ::= <bit><bit><bit><bit>

<bbk2t> ::= <bit><bit>

<bbk1t> ::= <bit>

<reps> ::= <rept>
         | <rept> "|" <reps>

<rept> ::= "<bbk4><bbk4>"
         | "<bbk2><bbk2><bbk2><bbk2>"
         | "<bbk1><bbk1><bbk1><bbk1><bbk1><bbk1><bbk1><bbk1>"

<bit> ::= "<bit>"
        | 1
        | 0

<val> ::= <valt>
        | <valt> "|" <val>

<valt> ::= 1 | 0
```

In this case the grammar specifies the construction of another generative bitstring grammar. The subsequent bitstring grammar that can be produced from

the above meta-grammar is restricted such that it can contain building blocks of size 8. Some of the bits of the building blocks can be fully specified as a boolean value or may be left as unfilled for the second step in the mapping process. An example bitstring grammar produced from the above meta-grammar could be:

```
<bitstring> ::= <bit>11<bit>00<bit><bit>
              | <bbk2><bbk2><bbk2><bbk2>
              | 11011101
              | <bbk4><bbk4>
              | <bbk4><bbk4>

<bbk4> ::= <bit>11<bit>

<bbk2> ::= 11

<bbk1> ::= 1

<bit> ::= 1 | 0 | 0 | 1
```

To allow the creation of multiple building blocks of different sizes the following grammar (referred to as `GEGE+KBB`) could be adopted (again shown for 8-bit strings).

```
<g> ::=
       "<bitstring> ::=" <reps>
            "<bbk4> ::=" <bbk4>
            "<bbk2> ::=" <bbk2>
            "<bbk1> ::=" <bbk1>
             "<bit> ::=" <val>

<bbk4> ::= <bbk4t>
         | <bbk4t> "|" <bbk4>
<bbk2> ::= <bbk2t>
         | <bbk2t> "|" <bbk2>
<bbk1> ::= <bbk1t>
         | <bbk1t> "|" <bbk1>
<bbk4t> ::= <bit><bit><bit><bit>
<bbk2t> ::= <bit><bit>
<bbk1t> ::= <bit>
<reps> ::= <rept>
         | <rept>  "|" <reps>
<rept> ::= "<bbk4><bbk4>"
         | "<bbk2><bbk2><bbk2><bbk2>"
         | "<bbk1><bbk1><bbk1><bbk1><bbk1><bbk1><bbk1><bbk1>"
<bit> ::= "<bit>"
        | 1
        | 0
<val> ::= <valt>
        | <valt> "|" <val>
<valt> ::= 1
         | 0
```

An example bitstring grammar produced by the above meta-grammar is provided below.

```
<bitstring> ::= <bit>11<bit>00<bit><bit>
              | <bbk2><bbk2><bbk2><bbk2>
              | 11011101
              | <bbk4><bbk4>
              | <bbk4><bbk4>

<bbk4> ::= <bit>11<bit>
```

```
        | 000<bit>

<bbk2> ::= 11
        | 00
        | <bit>1

<bbk1> ::= 0
        | 0

<bit> ::= 1 | 0 | 0 | 1
```

Modularity exists above in the ability to specify the size and content (or partial content) of a buiding block through it's incorporation into the solution grammar. This building block can then be repeatedly reused in the generation of the phenotype. An additional mechanism for reuse is through the Wrapping operator of Grammatical Evolution. During the mapping process if we reach the end of the genotype and still have outstanding decisions to make on the construction of our phenotype we can invoke the wrapping operator to move our reading head back to the first codon in the genome. This allows the reuse of rule choices if the codons are used in the same context.

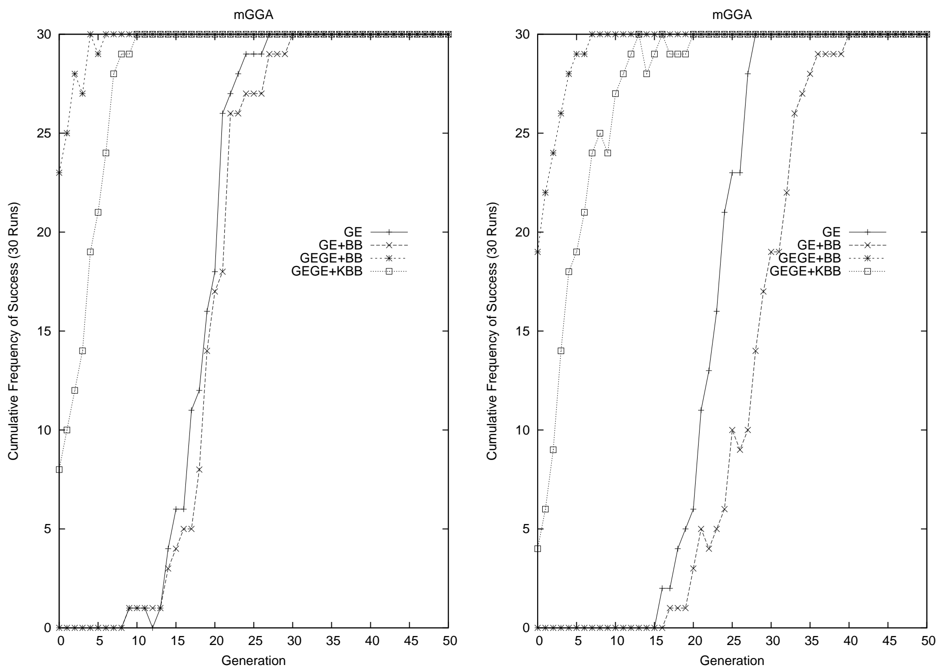

Fig. 1. Plot of the cumulative frequency of success for the OneMax 180 bit problem (left) and the OneMax 210 bit problem (right).

Given that the lengths of binary strings which may need to be represented can grow quite large it is possible to automate the creation of meta-grammars by

simply providing the length of the target solution and creating all possible building block structures that can be used to create a bitstring of the target length. In this study the target binary strings are of lengths 60, 90, 120, 180, and 210. The building block sizes incorporated in their corresponding grammars are therefore all integers that divide into the target string lengths (i.e., for a target string of length 60 the building blocks are of sizes 30, 20, 15, 12, 10, 6, 5, 4, 3, 2 and 1). Meta-grammars are of course not limited to the specification of grammars for binary strings and can be easily extended to the representation of real and integer strings as well as programs, or any structure for which it is possible to represent in a grammatical form.

4 Experimental Setup and Results

The mGGA was applied to three problem types, namely, instances of onemax, instances of a deceptive trap problem, and a dynamic problem instance. Two onemax instances were adopted with target string lengths of 180 and 210.
Similarly, two instances of a Trap5 problem were used having target string lengths of 60 and 90, with these having 12 and 18 subfunctions respectively. The dynamic problem instance has an alternating target every 20 generations between a onemax and zeromax problem with a target string length of 120 bits investigated. In each case the same parameter settings were adopted. These were, a population size of 100, tournament selection, generational replacement, a crossover probability of 0.3 between homologous chromosomes, and a mutation probability of 0.01. Initialisation of the population was performed randomly with individuals having in the range of 1 to 20 codons.

The results for the onemax instances are presented in Fig. 1. It is clear on both instances that the evolvable meta-Grammar's (GEGE+BB and GEGE+KBB) outperform the static grammars (GE and GE+BB) in terms of the speed at which the target solution is reached, although all grammars are capable of finding the perfect solution in every run beyond 50 generations. We would expect, and it is observed, that the performance of the static grammars are close due to their similarity.

Results for the Trap5 instances are presented in Fig. 2. In this case the evolvable meta-Grammar's outperform the static grammars both in terms of their speed at obtaining perfect solutions and in terms of the number of successful runs at the end of 100 generations. The static grammar runs having less than 50% success rate on the 60 bit instance, and less than 33% on the 90 bit instance. This is compared to a 100% success rate for both the GEGE+BB and GEGE+KBB grammars.

The immeadiate success of the GEGE+BB and GEGE+KBB grammars on these static problems can be attributed, in part at least, to the relatively small number of choices that need to be made to generate a perfect solution when compared to making a decision on all 60, 90, 180 or 210 bits individually. In effect if the solution grammar is generated to specify that a solution is comprised of a building block of size 1 bits, and that the building block takes on the value 0, only

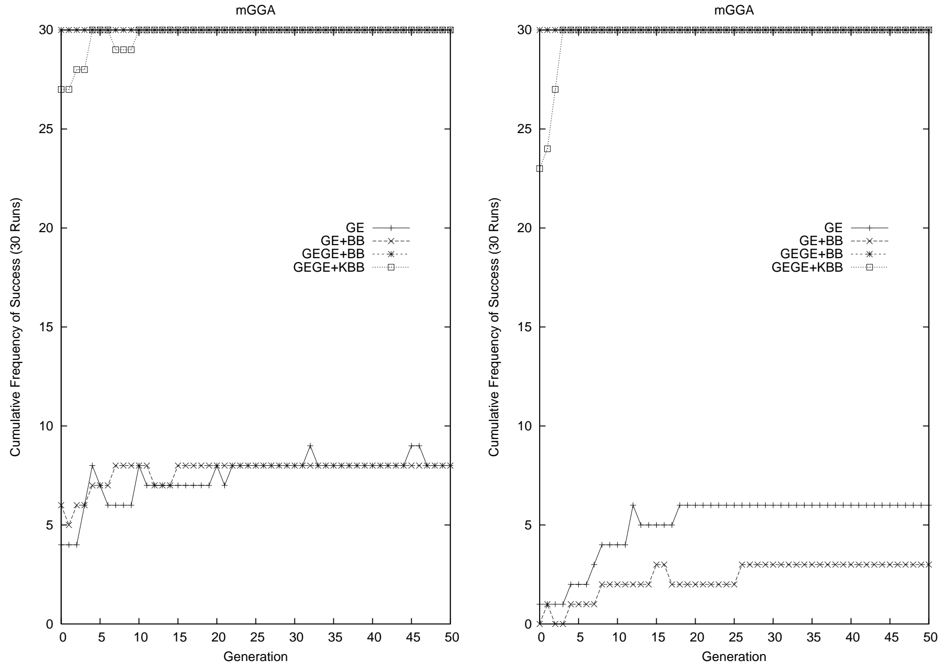

Fig. 2. Plot of the cumulative frequency of success for the 5Trap 60 bit problem (left) and 5Trap 90 bit problem (right).

two codons are required to fully specify a correct solution to the Trap5 problem instances. The evolvable representation adopted contains more redundancy of a form that provides an increased number of avenues by which a perfect solution can be reached relatively quickly within the initial generations, which goes some way to explain the superior performance of the GEGE grammars [13].

Results for the dynamic instance are provided in Fig.'s 3 and 4. We can see that the two static grammars (GE and GE+BB) and the GEGE+BB grammar perform well during the first twenty generations with the majority of runs finding a perfect solution during this time. However, from the first change in target at generation 21 up until generation 40 the performance of the GE and GE+BB grammars degrade significantly in contrast to the two evolvable grammars (GEGE+BB and GEGE+KBB), which have success rates over 50% compared to 0% for their static counterparts. On return to the original target between generations 41 to 60 the static grammars peak at generation 60 with just over a 50% success rate. During this same period the GEGE+KBB grammars success rate is improving steadily towards 66%, while the GEGE+BB grammars performance remains constant just short of a 100% success rate. With the next change in target at generation 61 performance of the static grammars falls off towards a 0% success rate while both the GEGE+BB and GEGE+KBB performance improves. The mean best fitness plot (Fig. 4) supports the trends we

Fig. 3. Plot of the cumulative frequency of success for the Dynamic 120 bit problem.

observe in the success rate plots. Over the course of the run we see a steady improvement in the mean best fitness for the GEGE+BB and GEGE+KBB grammars, with the performance of the GE and GE+BB grammars fluctuating more distinctly with each change in target solution. Performance for the GE and GE+BB grammars always being better upon return to the original target presented in the first twenty generations.

Two successful, abbreviated, sample solution grammars for the Dynamic 120 problem instance are given below for both of the GEGE+BB and GEGE+KBB meta-grammars. In each solution the same grammar represents the two target solutions that are required and can allow a switch between solutions by changing a single choice made when mapping <bitstring>.

```
GEGE+KBB Abbreviated Sample Solution (Dynamic 120 Problem)
<bitstring> ::= <bbk1>..<bbk1>
              | <bbk4>..<bbk4>
              | <bbk1>..<bbk1>
     <bbk4> ::=  00<bit><bit>
     <bbk1> ::=  1
      <bit> ::=  0

solution: 111111111111111111111111111111111111111111111111111111111111
          111111111111111111111111111111111111111111111111111111111111
```

Fig. 4. Plot of the the mean best fitness (left) and mean average fitness (right) for the Dynamic 120 bit problem.

```
GEGE+BB Abbreviated Sample Solution (Dynamic 120 Problem)
<bitstring> ::= <bbk8>..<bbk8>
              | <bbk24><bbk24><bbk24><bbk24><bbk24>
              | <bbk5>..<bbk5>
              | <bbk1>..<bbk1>
<bbk24> ::=  <bit><bit>1110<bit>10110<bit>0010<bit>0<bit><bit>1<bit><bit>
 <bbk8> ::=  0<bit>0<bit><bit>1<bit><bit>
 <bbk5> ::=  0<bit>0<bit><bit>
 <bbk1> ::=  1
  <bit> ::=  0
solution: 000000000000000000000000000000000000000000000000000000000000
          000000000000000000000000000000000000000000000000000000000000
```

5 Conclusions & Future Work

We presented the meta-Grammar Genetic Algorithm (mGGA), illustrating the application of evolvable grammars to Genetic Algorithms. On the three problem domains examined there are clear performance advantages on both the two static problems and the dynamic problem instance for the evolvable grammars GEGE+BB and GEGE+KBB over their static counterparts GE and GE+BB.

In addition to the application to more benchmark problem instances in particular to those belonging to the dynamic class, future work will investigate the effects of the wrapping operator, alternative grammars and comparisons to other Genetic Algorithms from the literature. It would be particularly interesting to analyse the scalability of these algorithms compared to the competent GA's, given that the use of wrapping coupled with the reuse of building blocks has the potential to shorten the genotypes necessary to represent harder problem instances. A number of avenues to facilitate the co-evolution of the grammar and solution such as different operator probabilities will also be investigated.

References

1. Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.
2. Koza, J. R., Keane, M. A., Streeter, M. J., Mydlowec, W., Yu, J., Lanza, G. (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.
3. O'Neill, M., Ryan, C. (2004). Grammatical Evolution by Grammatical Evolution. The Evolution of Grammar and Genetic Code. *LNCS 3003. Proc. of the European Conference on Genetic Programming 2004*, pp. 138-149, Coimbra, Portugal. Springer.
4. O'Neill, M., Ryan, C. (2003). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers.
5. O'Neill, M. (2001). Automatic Programming in an Arbitrary Language: Evolving Programs in Grammatical Evolution. PhD thesis, University of Limerick.
6. O'Neill, M., Ryan, C. (2001). Grammatical Evolution, *IEEE Trans. Evolutionary Computation*. 2001.
7. 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*, pp. 83-95, Springer-Verlag.
8. Dempsey, I., O'Neill, M., Brabazon, A. (2004). Grammatical Constant Creation. In LNCS 3103, *Proceedings of GECCO 2004*, Part 1, pp. 447-458, Seattle, WA, USA.
9. O'Neill, M., Cleary, R. (2004). Solving Knapsack Problems with Attribute Grammars. In *Proceedings of the Grammatical Evolution Workshop 2004*, GECCO 2004, Seattle, WA, USA.
10. Shan, Y., McKay, R. I., Baxter, R., Abbas, H., Essam, D., Nguyen, H.X. (2004). Grammar Model-based Program Evolution. In *Proceedings of the 2004 Congress on Evolutionary Computation, CEC 2004*, Vol. 1, pp. 478-485, Portland, Oregan, USA.
11. Chomsky, N. (1975). *Reflections on Language*. Pantheon Books. New York.
12. Pinker, S. (1995). *The language instinct: the new science of language and the mind*. Penguin, 1995.
13. Rothlauf, F. (2002). *Representations for Genetic and Evolutionary Algorithms*. Physica-Verlag, 2002.