Grammatical bias and building blocks in meta-grammar Grammatical Evolution

Erik Hemberg, Michael O'Neill and Anthony Brabazon

***Abstract*—This paper describes and tests the utility of a meta Grammar approach to Grammatical Evolution (GE). Rather than employing a fixed grammar as is the case with canonical GE, under a meta Grammar approach the grammar that is used to specify the construction of a syntactically correct solution is itself allowed to evolve. The ability to evolve a grammar in the context of GE means that useful bias towards specific structures and solutions can be evolved and directly incorporated into the grammar during a run. This approach facilitates the evolution of modularity and reuse both on structural and symbol levels and consequently could enhance both the scalability of GE and its adaptive potential in dynamic environments. In this paper an analysis of the extent that building block structures created in the grammars are used in the solution is undertaken. It is demonstrated that building block structures are incorporated into the evolving grammars and solutions at a rate higher than would be expected by random search. Furthermore, the results indicate that grammar design can be an important factor in performance.**

I. Introduction

The meta Grammar Genetic Algorithm is a Genetic Programming (GP) [1] approach to a Genetic Algorithm (GA) representation [2]. The meta Grammar Genetic Algorithm (mGGA) was initially developed in [3], [4] and has shown good performance on a range of test problems. In this paper two aspects of the mGGA are analysed. The first aspect concerns the content of the evolved grammars and asks whether modular structures are being evolved and then used in solving the problems examined. The second aspect addressed focuses on the design of input grammars, by comparing the performance of the original biased grammar to unbiased variants.

The paper is structured as follows. First an overview of the meta-grammar approach to Grammatical Evolution(GE) is presented and earlier research in this area is discussed in Sec. II. Sec. III describes the two experiments undertaken. The results obtained are provided and discussed in Sec. IV, before finishing the paper in Sec. V with Conclusions and Future Work.

II. Meta Grammars in Grammatical evolution

The grammar-based Genetic Programming GP [5] approach upon which this study is based is the Grammatical Evolution by Grammatical Evolution algorithm $(GE)^2$ [6]. In turn, this is based on the GE algorithm [7].

Erik Hemberg, Michael O'Neill and Anthony Brabazon are with the Natural Computing Research & Applications Group, University College Dublin, Ireland (email: erik.hemberg@ucd.ie, m.oneill@ucd.ie, anthony.brabazon@ucd.ie)

A. Grammatical Evolution

Rather than representing the programs as parse trees, as in GP, a variable length linear genome representation is used. A genotype-phenotype mapping is employed where an an individual's binary string is interpreted as a sequence of integer values (called *codons*), which are then used to select production rules from a Backus-Naur Form (BNF) grammar. A context free grammar is a four tuple (N, Σ, R, S) [8]. Where N is a finite set of non-terminal symbols. Σ is a finite set of terminal symbols, $N \cap \Sigma = \emptyset$. R is a finite set of production rules, $A \rightarrow \alpha$, $A \in N$ and $\alpha \in (\Sigma \cup N)^*$, and S is the start symbol, $S \in N$. To determine which α is chosen $\alpha = c \bmod r$, where c is the value of the codon and r is the number of choices for the current R

B. Meta Grammar Grammatical Evolution

$(GE)^2$ 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 to the problem of interest. Fig. 1 illustrates the meta grammar GE.

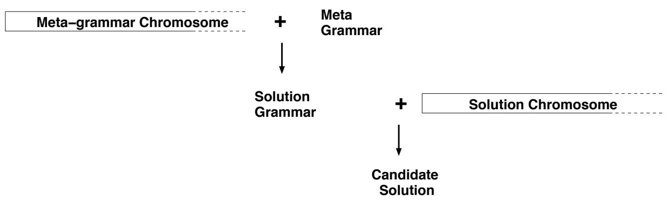


Fig. 1. An overview of the meta grammar approach to GE. The meta Grammar generates a solution grammar, which is used to generate a candidate solution.

In order to allow the evolution of a grammar we must provide a grammar to specify the form that a grammar can take. By allowing an Evolutionary Algorithm to adapt its representation, through the evolution of the grammar in this case, it is possible to automatically incorporate biases into the search process. In this case we can allow the mGGA to evolve biases towards different building block structures of varying sizes. For further examples of what can be represented with grammars see [7] and [9] for an alternative approach to grammar evolution. A more developmental approach was also explored in [10], [11].

In $(GE)^2$ the meta grammar dictates the construction of the solution grammar. In order to operationalise this, two separate variable-length, genotypic binary chromosomes were used. The first chromosome generates the solution grammar from the meta grammar and the second chromosome generates 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. For evolution to be successful it must *co-evolve* both the meta grammar and the structure of solutions based on the evolved meta grammar. Consequently, the search space is larger than in standard GE where only a solution chromosome is evolved.

1) Examples of Meta Grammar Grammatical Evolution: Fig. 2 illustrates a meta grammar mapping a binary string of size 4.

Another example of a meta grammar that could be used to evolve grammars for generating 8 bit binary strings is provided below.

```
<g> ::=  "<bitstring> ::=" <reps>
              "<bb4> ::=" <bb4>
              "<bb2> ::=" <bb2>
              "<bb1> ::=" <bb1>
              "<bit> ::=" <val>
<bb4> ::= <bb4t>
      | <bb4t> "|" <bb4>
<bb2> ::= <bb2t>
      | <bb2t> "|" <bb2>
<bb1> ::= <bb1t>
      | <bb1t> "|" <bb1>
<bb4t> ::= <bit><bit><bit><bit>
<bb2t> ::= <bit><bit>
<bb1t> ::= <bit>
<reps> ::= <rept>
       | <rept>  "|" <reps>
<rept> ::= "<bb4><bb4>"
       | "<bb2><bb2><bb2><bb2>"
       | "<bb1><bb1><bb1><bb1><bb1><bb1><bb1><bb1>"
<bit> ::= "<bit>"
      | 1
      | 0
<val> ::= <valt>
      | <valt> "|" <val>
<valt> ::= 1
      | 0
```

An example bit string grammar that could be sampled from the above meta grammar follows below. In this example, there are five possible forms that a `<bitstring>` can take on, with two possible choices for building block structures of size 4 and 1, and three choices for building block structures of size 2. The rule for generating a `<bit>` has four possible outcomes with a clear bias towards a `<bit>` becoming a `1` with a probability of 0.75 (3 of the four choices result in `1`) .

```
<bitstring> ::= <bit>11<bit>00<bit><bit>
           | <bb2><bb2><bb2><bb2>
           | 11011101
           | <bb4><bb4>
           | <bb4><bb4>
<bb4> ::= <bit>11<bit>
      | 000<bit>
<bb2> ::= 11
      | 00
      | <bit>1
<bb1> ::= 0
      | 0
<bit> ::= 1
      | 0
      | 1
      | 1
```

C. Earlier Research

There have been a number of studies of a meta grammar approach to GE [6], [3], [4], [12]. In each of these the same rate of evolutionary search was adopted on both the meta grammar and solution chromosomes through the adoption of the same rates of mutation and crossover. The original study [6] investigated the feasibility of this approach and demonstrated its effectiveness in dynamic environments, see [13] for a discussion on dynamic environments. In the mGGA [3] the meta grammar approach was shown as an effective method to perform as an alternative binary string Genetic Algorithm through the provision of a mechanism to achieve modularity. A follow-up study demonstrated that the mGGA had an improved ability to scale to harder problem instances over the Modular GA (MGA) [14]. The examination of the solutions and solution grammars evolved by meta grammar GE indicated a tendency to generate concise solution grammars which did not include a wide choice of production rules [3]. This suggested that the system was pushing most of the evolutionary search onto the solution grammar chromosome rather than balancing search between the two chromosomes. In a recent study [15] alternative rates of evolution on both the meta grammar and solutions chromosomes was conducted. The most interesting finding from this study was that most of the evolutionary search is currently focused on the meta grammar chromosome. The resulting evolved solution grammars represent only a small fraction of the potential solution space.

This study seeks to extend earlier work in two ways.

Firstly, a potential limitation of these earlier studies is that the grammars adopted by the mGGA contain bias towards the use of the building block structures of size greater than one. That is, they under-represent a more classical GA-type representation where each bit of a solution is specified individually. The study addresses this limitation by removing bias from the grammar at various levels to determine if this bias might be the source of performance loss.

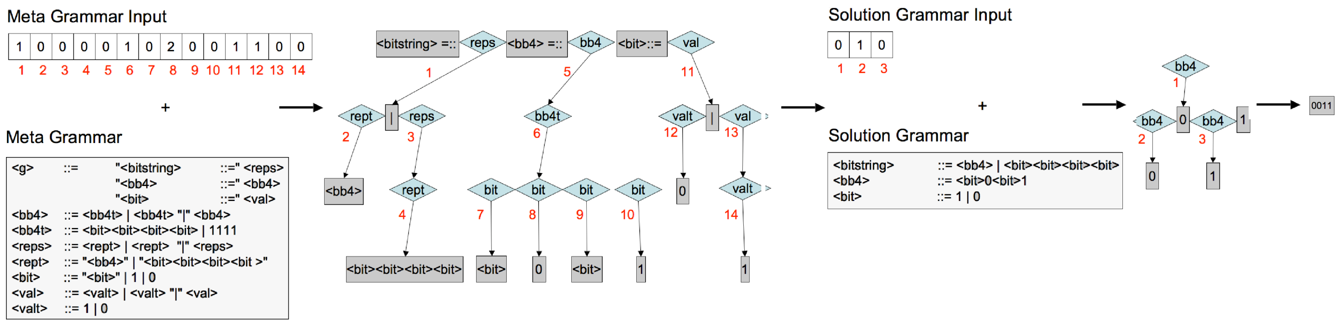
Secondly, we wish to understand if in fact the modular structures that are provided by this approach are in fact adopted in the creation of the solutions, and are therefore responsible (at least in part) for the performance gains achieved over the MGA.

III. Experiments & Results

Initially the two experiments are described followed by the experimental setup. The first experiment focuses on the bias question. Does a bias towards the building block structures result in a performance loss when compared to unbiased equivalents? The second experiment asks if the building block structures are actually used in the creation of solutions.

A. Fitness function - Checkerboard

We initially examine the Checkerboard problem which has been used in earlier studies to benchmark the performance of the mGGA against that of the MGA. In this problem a pattern of colours or states is imposed upon a two dimensional grid



```
<g>      ::=      "<bitstring>      ::=" <reps>
                  "<bb4>            ::=" <bb4>
                  "<bit>            ::=" <val>
<bb4>    ::= <bb4t> | <bb4t> "|" <bb4>
<bb4t>   ::= <bit><bit><bit><bit> | 1111
<reps>   ::= <rept> | <rept>  "|" <reps>
<rept>   ::= "<bb4>" | "<bit><bit><bit><bit >"
<bit>    ::= "<bit>" | 1 | 0
<val>    ::= <valt> | <valt> "|" <val>
<valt>   ::= 1 | 0
```

```
<bitstring>     ::= <bb4> | <bit><bit><bit><bit>
<bb4>           ::= <bit>0<bit>1
<bit>           ::= 1 | 0
```

Fig. 2. An example mapping of a meta grammar. Diamonds are non terminal symbols and rectangles are terminal symbols. The numbers by the arrows are used to denote which input chose the rule at the

called the Checkerboard, introduced in [14]. There are 2 possible states adopted for each square on the grid, i.e., black or white, which can be represented as bit values 1 and 0 respectively. Fitness is simply measured by summing the number of squares that are in an incorrect state. In this study the fitness is minimized, such that 0.0 is the best possible fitness where all of the candidate solution's squares exactly match the target checkerboard-pattern. Fig. 3 which illustrates scaled-up versions of a pattern. The instances will be referred to by the total number of bits needed to describe the board. The board size Cb_N is calculated as follows $Cb_N = ss_i^2 * 4 * 2$, where ss_i is the number of consecutive ones or zeros. In this paper the board sizes used where Cb_{32} and Cb_{128}

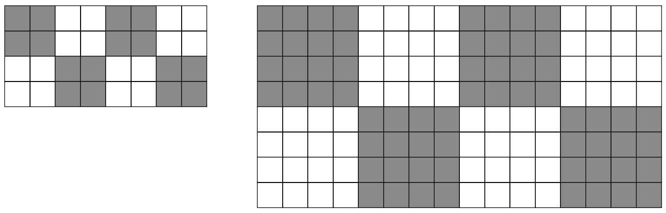


Fig. 3. Checkerboard patterns for Cb_{32} and Cb_{128}.

A dynamic noisy version of the checkerboard problem is also adopted. In [12] the checkerboard was distorted with noise to make the problem harder. The distortion was implemented by the possibility of each bit in the original board to flip with a probability of 0.05. Here it is taken a step further and the checkerboard is distorted after a set period length of 10 or 100 iterations. Therefore, we have two dynamic variants of each problem size (Cb_{32} and Cb_{128}).

B. Experiment 1 - Grammar Design

We examine a series of grammars that incrementally remove bias on different levels from the mGGA grammars adopted in earlier studies. We wish to determine if this bias is having a negative impact on performance. Each of these grammars represents a solution to the Checkerboard problem.

For each of the modified grammars only changes from the original grammar will be shown. In order to fit the grammars to the page the "..." should be read as repeat $problemsize/blocksize$ times. A large number of grammars have been analysed with a series of incremental changes between each grammar. Given space restrictions we have only presented the significant grammars.

1) Original - grammar 0: To allow the creation of multiple building block structures of different sizes the following meta grammar could be used (this approach was implemented in [3], [12]). Below a is an example of a grammar for Cb_{32}. As can be seen when expanding `<bitstring>` there will be a bias towards using building block structures of size >1.

```
<g> ::=  "<bitstring> ::=" <reps>
              "<bb16> ::=" <bb16>
              "<bb8> ::=" <bb8>
              "<bb4> ::=" <bb4>
              "<bb2> ::=" <bb2>
              "<bb1> ::=" <bb1>
               "<bit> ::=" <val>
<bb16> ::= <bb16t>
       | <bb16t> "|" <bb16>
<bb8> ::= <bb8t>
      | <bb8t> "|" <bb8>
<bb4> ::= <bb4t>
      | <bb4t> "|" <bb4>
<bb2> ::= <bb2t>
      | <bb2t> "|" <bb2>
<bb1> ::= <bb1t>
      | <bb1t> "|" <bb1>
<bb16t> ::= <bit>...<bit>
<bb8t> ::= <bit><bit><bit><bit><bit><bit><bit><bit>
<bb4t> ::= <bit><bit><bit><bit>
<bb2t> ::= <bit><bit>
<bb1t> ::= <bit>
<reps> ::= <rept>
       | <rept>  "|" <reps>
<rept> ::= "<bb16><bb16>"  | "<bb8><bb8><bb8><bb8>"
         | "<bb4><bb4><bb4><bb4><bb4><bb4><bb4><bb4>"
         | "<bb2>...<bb2>
         | "<bb1>...<bb1>
<bit> ::= "<bit>"
      | 1
      | 0
<val> ::= <valt>
      | <valt> "|" <val>
<valt> ::= 1
       | 0
```

2) Equal 1 - grammar 5: This grammar has the same probability for `<bitstring>` to use a GE GA or a building block structures of size greater than 1.

```
<g> ::= "<bitstring> ::= <GA>...<GA>|" <reps>
             "<bb16> ::= " <bb16>
              "<bb8> ::= " <bb8>
              "<bb4> ::= " <bb4>
              "<bb2> ::= " <bb2>
              "<bit> ::=" <val>
               "<GA> ::= 1 | 0"
```

```
<reps> ::= <rept>
       | "<GA>...<GA> |" <rept> "|" <reps>
<rept> ::= "<bb16><bb16>"
       | "<bb8><bb8><bb8><bb8>"
       | "<bb4><bb4><bb4><bb4><bb4><bb4><bb4><bb4>"
       | "<bb2>...<bb2>"
       | <reps>
```

3) Equal 2 - grammar 6: This grammar has the same probability for `<bitstring>` to use a GE GA or building block structures of any size.

```
<g> ::= "<bitstring> ::= <GA>...<GA>|" <reps>
             "<bb16> ::= " <bb16>
              "<bb8> ::= " <bb8>
              "<bb4> ::= " <bb4>
              "<bb2> ::= " <bb2>
              "<bb1> ::= " <bb1>
              "<bit> ::=" <val>
               "<GA> ::= 1 | 0"
<reps> ::= <rept>
       | "<GA>...<GA> |" <rept> "|" <reps>
<rept> ::= "<bb16><bb16>"
       | "<bb8><bb8><bb8><bb8>"
       | "<bb4><bb4><bb4><bb4><bb4><bb4><bb4><bb4>"
       | "<bb2>...<bb2> "
       | "<bb1>...<bb1> "
       | <reps>
```

4) GE GA - grammar 11: This is a simple GE approach to GA that does not use a meta grammar. It is implemented in order to provide a benchmark for the other results. The grammar pre-specifies the number of bit positions in the solution, and the genome is used to select what each bit becomes.

```
<bitstring> ::= <GA>...<GA>
<GA> ::= 1
     | 0
```

C. Experiment 2 - Building Block Use in Evolved Grammars

In the second part of this study we examine the building block structures that are being generated in the evolved grammars. Primarily we wish to ascertain if the evolved grammars and co-evolved solutions actually include and use building block structures of size greater than one when solving a problem. To this end we compare the frequency of occurrence of the building block rules in solutions against the frequency of occurrences of the same building block structures using a random search.

D. Setup

The settings in Table I were adopted. The population size was the one that solved the instance within 10% of where 30 runs are successful for a maximum of 800 iterations. Both chromosomes had the same initial length, roughly three times the problem size. The chromosomes were variable-length vectors of integers (2 byte integers). Rank replacement is adopted with a constant population size, where the new children are pooled with the current population, ranked, and the worst individuals are removed. One-point fixed crossover, and integer mutation where a new value was randomly chosen, are used. For the GE GA the parameters for the meta chromosome are not used.

TABLE I
PARAMETERS FOR THE GE ALGORITHM

Fitness function	Checkerboard
Checkerboard size	32, 128
Initial chromosome size	90, 300
Population size	121, 288
Initialisation	Random
Selection operation	Tournament Select
Tournament size	3
Replacement	Rank replacement
Max wraps	1
Generations	800
Crossover probability meta	0.9
Crossover probability solution	0.9
Mutation probability meta	0.01
Mutation probability solution	0.01

IV. Results

1) Experiment 1: Table II details the average generation at which a solution was found over the 30 runs and Table III details the results of a t-test on this data.

TABLE II
RESULTS. AVE IT. DENOTES AT WHICH ITERATION THE PROBLEM WAS SOLVED.

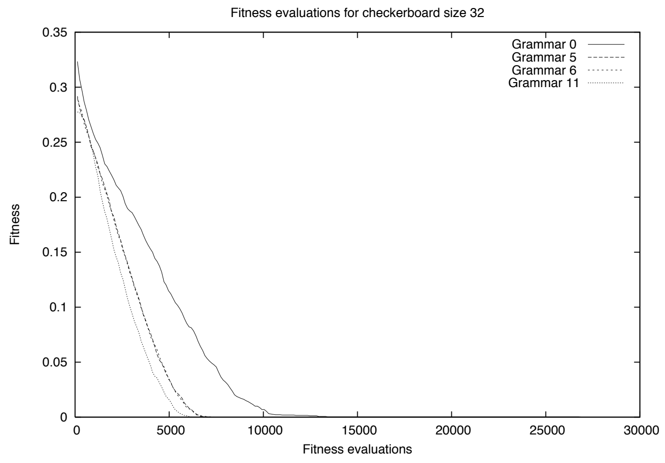
Run Cb_{32}	Ave It.	Std
Grammar 0	67.440	22.641
Grammar 5	46.900	5.355
Grammar 6	46.340	5.578
Grammar 11	41.420	4.312
Run Cb_{128}	Ave It.	Std
Grammar 0	136.960	38.839
Grammar 5	133.670	6.174
Grammar 6	134.090	6.785
Grammar 11	128.380	6.973

TABLE III
P-VALUES OF 2 SIDED T-TEST BETWEEN THE DIFFERENT GRAMMARS ON THE PROBLEM.

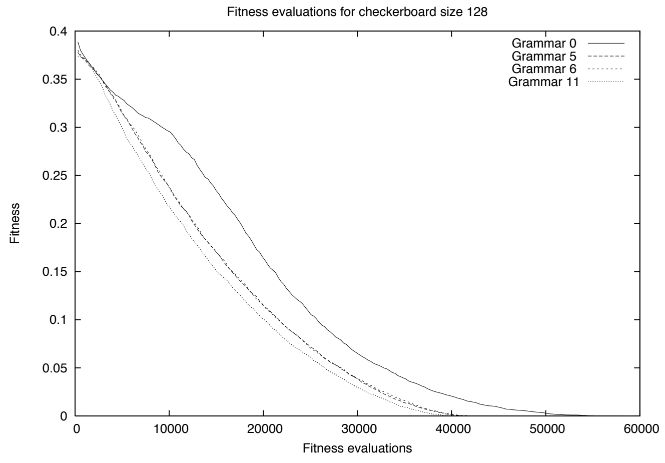
Cb_{32}				
Grammar	0	5	6	11
0		0.000	0.000	0.000
5	0.000		0.470	0.000
6	0.000	0.470		0.000
11	0.000	0.000	0.000	
Cb_{128}				
Grammar	0	5	6	11
0		0.404	0.468	0.031
5	0.404		0.648	0.000
6	0.468	0.648		0.000
11	0.031	0.000	0.000	

In the case of the simpler problem instance (Cb_{32}) both of the unbiased grammars (grammars 5 and 6) significantly outperform the biased grammar (grammar 0). This is not the case, however, for the larger Cb_{128} problem instance where statistically the results are the same. Examining the fitness plots in Fig. 4 we see that the unbiased grammars are solving the problem faster than the biased grammar on both problem instances.

An interesting result is that the simpler GE approach to GA, as represented in grammar 11, significantly outperforms all other grammars on both problem instances at the 95%



(a) Cb_{32}



(b) Cb_{128}

Fig. 4. On the x-axis is the number of fitness evaluations. On the y-axis is the normalized fitness.

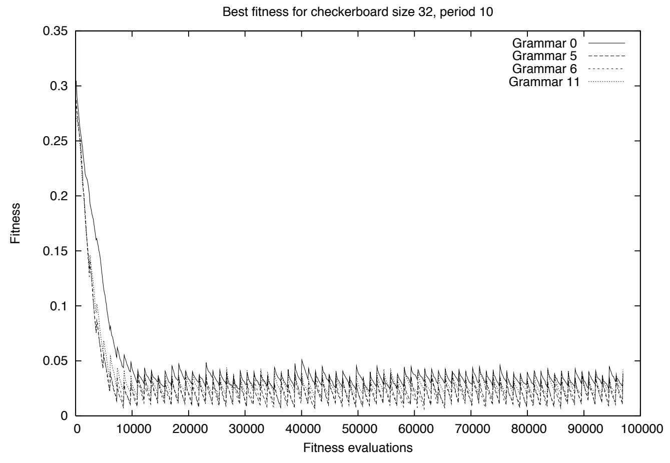
confidence level. However, on the larger instance there is no difference in performance between the biased grammar (grammar 0) and grammar 11 at the 99% level.

On both problem instances it is also worth noting that the fitness standard deviation in the biased grammar 0 is much higher than that of the unbiased grammars. This could be expected since a building block structure might solve the problem faster as well as hamper it more then a GA.

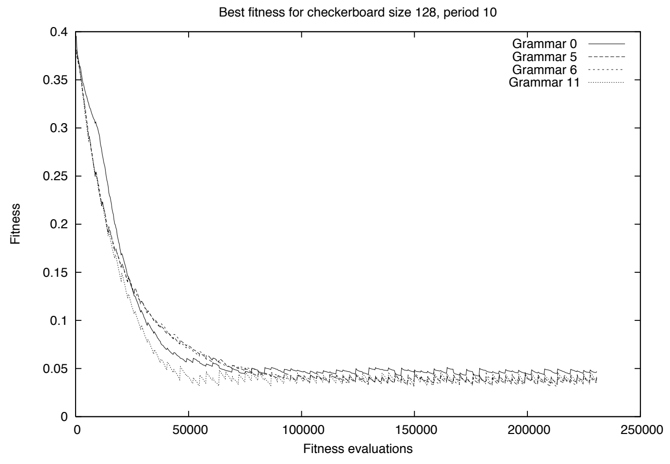
We examined the solutions evolved during the 30 runs at each generation and averaged the value at each position across the population. This result is visualised in Fig. 5 where a rapid convergence on the solution is found for all grammars .

Each column in the figure represents the *average* solution of that generation (i.e., the average value at each bit position of the solution) with the first column representing the first generation. It can be seen that the first generation is noisy (indicated by grey bit values at each locus) as expected from a random initialisation process. As we move east through the figure (moving through each generation of a run) we see very rapid convergence on the ideal target solution of the checkerboard (spread out across a line). This suggests that the problems examined in this case are too simple to expect to observe performance differences.

On the dynamic noisy checkerboard instances similar trends are observed. The number of fitness evaluations required to solve each problem are shown in Figs. 6 and 7. An alternative view of these results is to plot the area under the curve. In Figs. 8 and 9 we plot a simplification of the area under the curve by plotting a running total of the error at each period.



(a) Cb_{32}, period 10

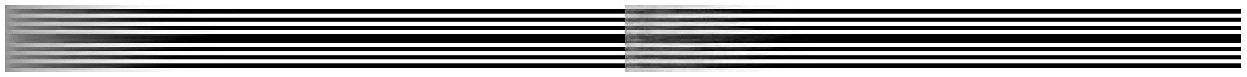


(b) Cb_{128}, period 10

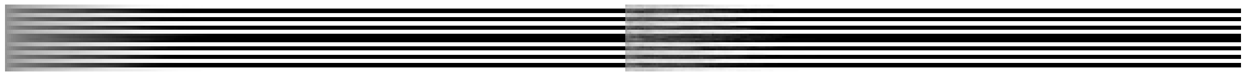
Fig. 6. On the x-axis is the number of fitness evaluations. On the y-axis is the normalized fitness.

2) Experiment 2: An idea of the use of building block structures can be formed by looking at the use of building block structures in the solution grammar of the individual that solved the problem and comparing it to random samples. Fig. 10 and 11 show these results.

It can be seen in the presence of random search that there is a 50:50 split between solutions that adopt building block structures of size N versus building block structures of size 1 (bottom half of each figure). In the presence of an

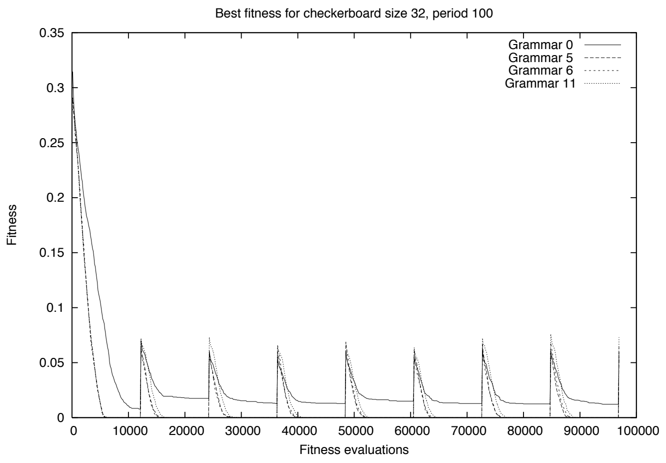


(a) Grammar 0

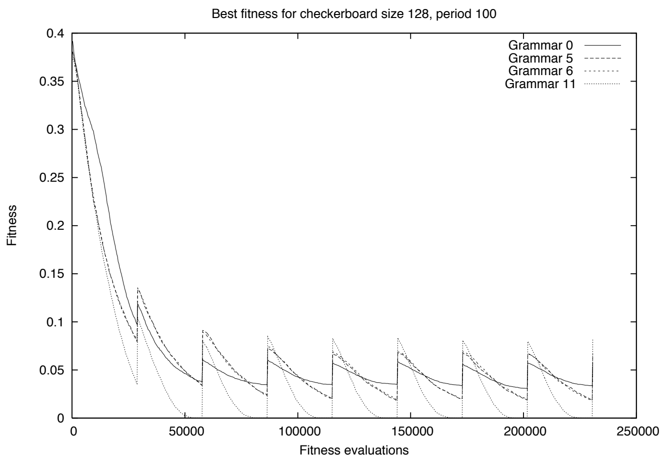


(b) Grammar 6

Fig. 5. The appearance of solutions for a sample of 100 runs of Cb_{32}. The y-axis represents the average value at each locus of the solution, the x-axis is the generation (300 in total). The left part of the figure represents the average values in the population. The right part of the figure represents the values for the best individuals. Convergence of the bit values occurs approximately within the first 50 generations.

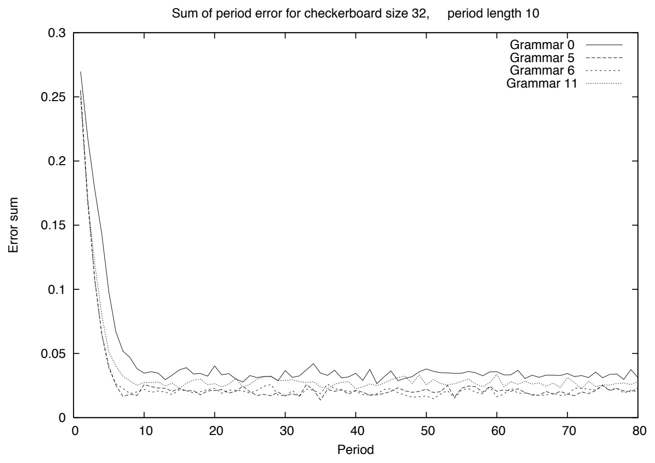


(a) Cb_{32}, period 100

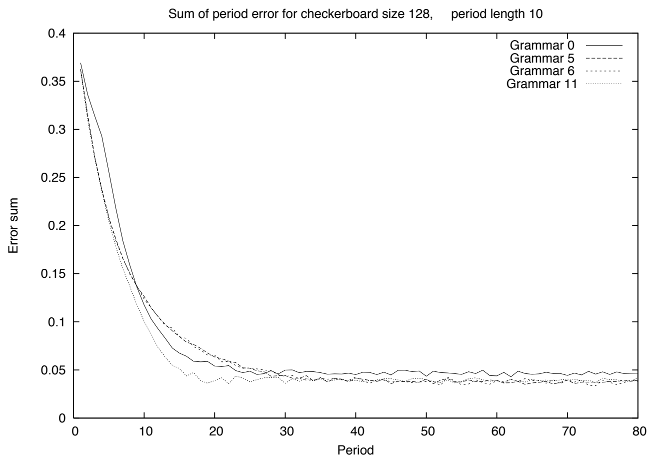


(b) Cb_{128}, period 100

Fig. 7. On the x-axis is the number of fitness evaluations. On the y-axis is the normalized fitness.



(a) Cb_{32}, period 10



(b) Cb_{128}, period 10

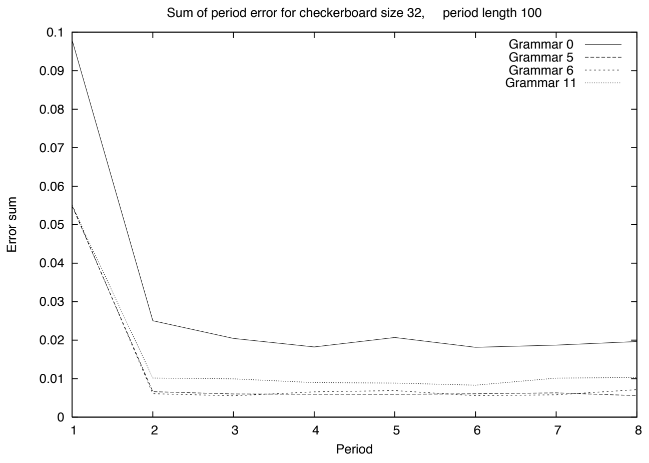
Fig. 8. On the x-axis is the period. On the y-axis is the normalized sum of errors.

evolutionary algorithm however, the relative use of building block structures of size N is significantly greater than blocks of size 1. Size 1 building block structures are only used 23% and 20% of the time on the Cb_{32} and Cb_{128} respectively.

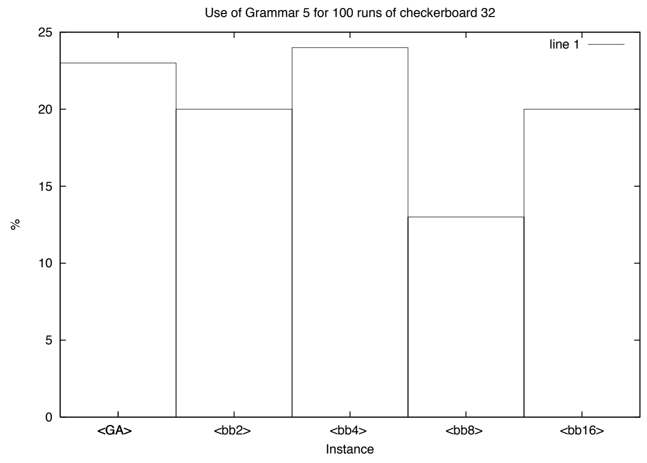
These results are encouraging and suggest that when a choice between adopting building block structures of size N versus no building block structures (i.e., size 1) is offered in the meta grammar, that building block structures are exploited in solving the problem.

V. Conclusions & Future Work

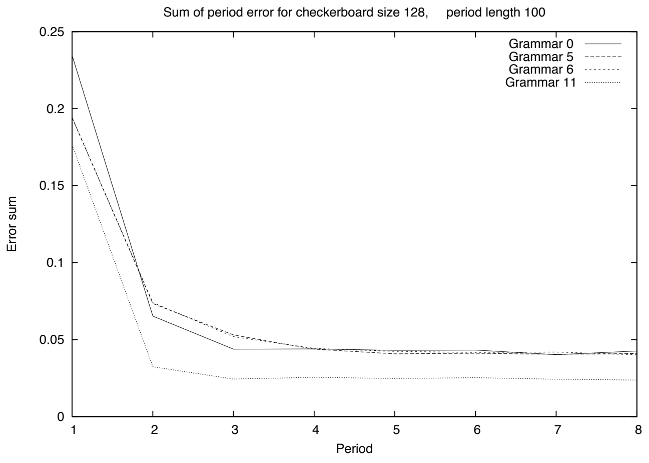
We set out to measure and understand two aspects of the meta Grammar Genetic Algorithm (mGGA). Firstly, an experiment was undertaken to determine if a bias in the grammar design used in earlier studies towards the use of building block structures impaired search efficiency. Secondly, we



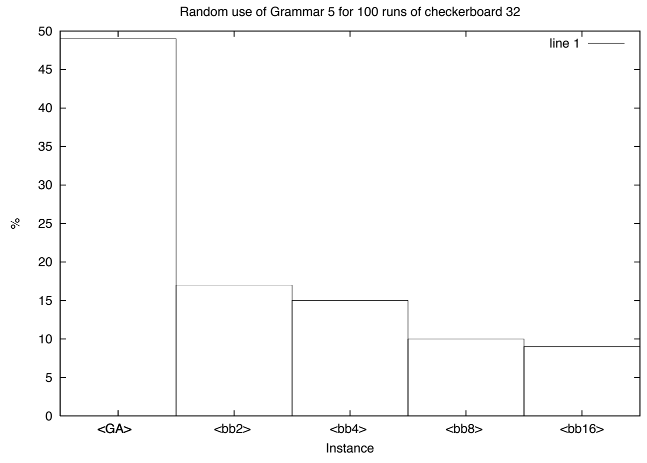
(a) Cb_{32}, period 100



(a) Cb_{32}, Grammar 5



(b) Cb_{128}, period 100



(b) Cb_{32}, Grammar 5, Random

Fig. 9. On the x-axis is the period. On the y-axis is the normalized sum of errors.

Fig. 10. Use of building block structures in solution for mGGA Cb_{32} using Grammar 5 and random search

wished to determine if the building block structures were in fact adopted by the population in solving the problem.

With respect to grammar design, it was found that it can be an important factor in the search efficiency of the meta-grammar approach on the problems analysed. An unbiased form of the grammar was found to outperform the biased equivalent.

An analysis of the adoption of building block structures by the evolutionary search found that these modular structures were utilised successfully by the population to solve the problem.

A recommendation arising from this study is towards the adoption of a meta-grammar that allows the utilisation of both a classic GA bitstring representation in conjunction with the modular building block structures.

Future work will focus on the generality of these results to different and harder problem domains. Results presented here suggest that the static problems adopted in this study were too easy to expect performance gains to be achieved by the meta-grammar approach. Similar, to the threshold of problem difficulty Koza observed to justify the use of ADFs, the additional overhead of the meta-grammar approach coupled with the building block grammars warrants its application to harder problems. Indeed, this suggestion is backed up by an earlier study where scalability of the mGGA to harder instances of the checkerboard yielded impressive performance gains relative to the modular GA [12].

Acknowledgement

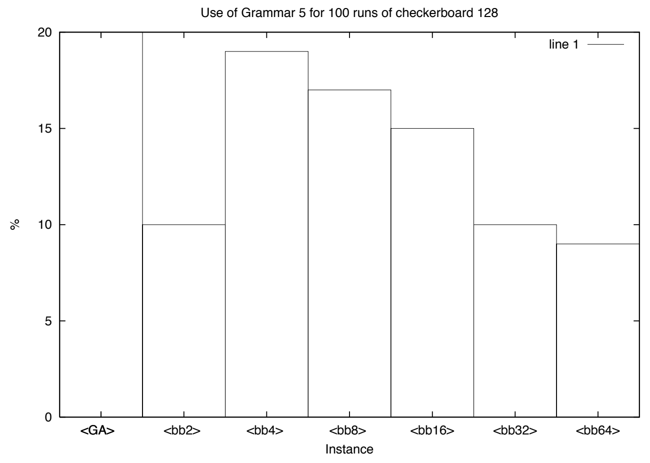
This research is based upon works supported by the Science Foundation Ireland under Grant No. 06/RFP/CMS042.

References

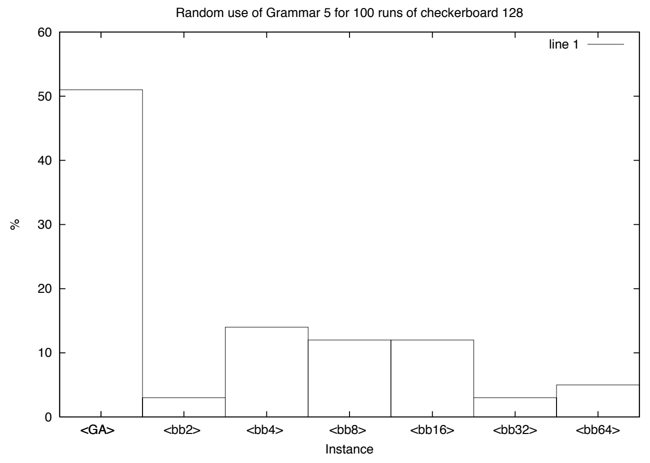
[1] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press, December 1992.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, January 1989.

[3] M. O'Neill and A. Brabazon, "mGGA: The meta-grammar genetic algorithm," in *Proceedings of the 8th European Conference on Genetic Programming*, ser. Lecture Notes in Computer Science, M. Keijzer, A. Tettamanzi, P. Collet, J. I. van Hemert, and M. Tomassini, Eds., vol. 3447. Lausanne, Switzerland: Springer, 30 Mar. - 1 Apr. 2005, pp. 311–320.



(a) Cb_{128}, Grammar 5



(b) Cb_{128}, Grammar 5, Random

Fig. 11. Use of building block structures in solution for mGGA Cb_{128} using Grammar 5 and random search

[4] I. Dempsey, M. O'Neill, and A. Brabazon, "Meta-grammar constant creation with grammatical evolution by grammatical evolution," in *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, H.-G. Beyer, U.-M. O'Reilly, D. V. Arnold, W. Banzhaf, C. Blum, E. W. Bonabeau, E. Cantu-Paz, D. Dasgupta, K. Deb, J. A. Foster, E. D. de Jong, H. Lipson, X. Llora, S. Mancoridis, M. Pelikan, G. R. Raidl, T. Soule, A. M. Tyrrell, J.-P. Watson, and E. Zitzler, Eds., vol. 2. Washington DC, USA: ACM Press, 25-29 Jun. 2005, pp. 1665–1671.

[5] P. A. Whigham, "Grammatically-based genetic programming," in *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, J. P. Rosca, Ed., Tahoe City, California, USA, 9 1995, pp. 33–41. [Online]. Available: citeseer.ist.psu.edu/whigham95grammaticallybased.html

[6] M. O'Neill and C. Ryan, "Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code," in *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, ser. LNCS, M. Keijzer, U.-M. O'Reilly, S. M. Lucas, E. Costa, and T. Soule, Eds., vol. 3003. Coimbra, Portugal: Springer-Verlag, 5-7 Apr. 2004, pp. 138–149.

[7] M. O'Neill and C. Ryan, *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Norwell, MA, USA: Kluwer Academic Publishers, 2003.

[8] T. L. Booth, *Sequential machines and automata theory.* Wiley, 1967.

[9] Y. Shan, R. I. McKay, R. Baxter, H. Abbass, D. Essam, and N. X. Hoai, "Grammar model-based program evolution," in *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*. Portland, Oregon: IEEE Press, 20-23 Jun. 2004, pp. 478–485.

[10] R. E. Keller and W. Banzhaf, "The evolution of genetic code in genetic programming," in *Proc. of the Genetic and Evolutionary Computation Conf. GECCO-99*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds. San Francisco, CA: Morgan Kaufmann, 1999, pp. 1077–1082.

[11] W. Piaseczny, H. Suzuki, and H. Sawai, "Chemical genetic programming evolutionary optimization of the genotype-to-phenotype translation set," *Artificial Life and Robotics*, vol. 9, no. 4, pp. 202–208, 2005.

[12] E. Hemberg, C. Gilligan, M. O'Neill, and A. Brabazon, "A grammatical genetic programming approach to modularity in genetic algorithms," in *Proceedings of the 10th European Conference on Genetic Programming*, ser. Lecture Notes in Computer Science, M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, and A. I. Esparcia-Alcázar, Eds., vol. 4445. Valencia, Spain: Springer, 11 - 13 Apr. 2007, pp. 1–11.

[13] J. Branke, E. Salihoğlu, and Şima Uyar, "Towards an analysis of dynamic environments," in *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2005, pp. 1433–1440.

[14] O. O. Garibay, I. I. Garibay, and A. S. Wu, "The modular genetic algorithm: Exploiting regularities in the problem space," in *Computer and Information Sciences - ISCIS 2003, 18th International Symposium, Antalya, Turkey, November 3-5, 2003, Proceedings*, ser. Lecture Notes in Computer Science, A. Yazici and C. Sener, Eds., vol. 2869. Springer, 2003, pp. 584–591.

[15] M. O'Neill, Erik Hemberg and A. Brabazon, "An investigation of meta grammars in grammatical evolution," in *EuroGP2008 Proceedings*, 2008.