Anniversary Scientific Conference
„40 YEARS DEPARTMENT OF INDUSTRIAL AUTOMATION", University of Chemical Technology and Metallurgy, Sofia, 18 March 2011

MULTI-OBJECTIVE OPTIMIZATION USING GRAMMATICAL EVOLUTION

A. Nasuf, A. Bhaskar, A. J. Keane

Computational Engineering and Design Group, School of Engineering Sciences, University of Southampton.
1 University Road, *Southampton, SO17 1BJ, e-mail: alkin.nasuf@soton.ac.uk*

Abstract: Grammatical Evolution is a multidisciplinary tool for evolving grammar sentences. A Grammar formalism to generate optimization variables using Grammatical Evolution is presented. The use of this formalism in single and multi objective parametric optimization is demonstrated. Multi-objective grammatical evolution is achieved by replacing the Genetic Algorithm with the Non-dominated Genetic Algorithm II method. Optimization results with some non-trivial single and multi-objective test functions are presented.

Key words: optimization, genetic programming, grammar, evolutionary computation, artificial intelligence, BNF

INTRODUCTION

Grammars are effective means of synthesizing all sorts of complex structures. Every grammar contains a set of rules. The rules are applied to predefined groups of primitives in order to select some and arrange those selected in a logical order. In general, each stand-alone primitive is meaningless, but when combined with other primitives following certain grammar rules it plays an essential role in the final complex structure.

The use of grammar to describe complex structures dates back to the work of Panini who creates 3,959 linguistic rules to formalise Sanskrit grammar around the 4th century BC. Seminal contributions towards the development of automated grammar formalisms lie in the work of Chomsky [1] who developed three natural language grammar formalisms. One of the formalisms is named "*a phrase-structure grammar*". This formalism can construct grammatically correct sentences as combinations of "terminal" and "non-terminal" natural language words following certain rules. In phrase-structure (context-free) grammar all non-terminal primitives are enclosed within angular brackets (< >). The role of terminal primitives is to terminate the addition of new primitives to the sentence, while non-terminal primitives are used to assist the rule selection process.

In the late 1960s, Backus and Naur [2] transformed the phrase-structure grammar formalism into syntax to express context-free grammar in programming languages, known as Backus-Naur Form (BNF). When used in a program code, the BNF syntax provides choice, e.g.,

<select> ::= *choice* 1 | *choice* 2 | ... | *choice n*

The intelligence behind rule selection in grammar is common for everyone familiar with grammar rules. Recently, in pattern learning and machine intelligence, it has been of interest to study how such intelligence can be transferred into computers. The good thing about BNF syntax, context-free grammar in particular, is that it provides a simple mechanism to represent grammar rules in a computer code. If a machine can decide which rules to select from the BNF syntax it will be able to construct logical sentences just as humans do.

One of the proposals to implant intelligence into a machine is to develop an evolutionary intelligence algorithm, where through evolution, the best sequence of the selected primitives survives. Genetic Programming (GP) is an algorithmic methodology for automated programming, where the program primitives are selected and arranged in a logical order by an evolutionary intelligence [3]. The intelligent selection of rules in GP is inspired by the natural selection of living organisms based on fitness calculations.

Grammatical Evolution (GE) is a branch of GP and as such is classified, as an evolutionary automatic programming method [4]. In GE, an automated programming is achieved by applying grammar rules to building blocks of the potential program i.e. program primitives. These primitives can be operands, variables, constants, functions, classes, objects, modules, etc. GE provides an intelligent selection of rules; this is achieved by the BNF syntax and the Genetic Algorithm (GA).

As in every evolutionary algorithmic methodology, the GA is inspired by the natural selection of living organisms [5]. In the GA, the genotype of two parents is paired in order to produce an "offspring" with unique characteristics. In natural selection, species strive to select their partner based on an estimate of their survival probability. In this way the chances of survival of the genes carried by their offspring in the following generations is increased significantly. This is the driving mechanism of evolution, where the search for the best and the fittest is continuous.

GE is a multidisciplinary tool for evolving grammar sentences. Initially it was developed as an automatic programming method [4, 6, 7, and 8], but since then has been successfully applied to solve a variety of complex problems [9]. Some interesting applications of GE include automatic composition of music [10], evolving financial predicting models and market index trading rules [11, 12], automatic programming of robots [13], evolutionary design [14, 15], and numerical parametric optimization [16].

This paper focuses on numerical optimization aspects of GE. In the next section a technical implementation of GE for single and multi objective optimization is presented. This is followed by experimental results using several non-trivial optimization test functions. Finally conclusions are presented.

TECHNICAL IMPLEMENTATION

Grammatical Evolution. An innovative feature of GE is its ability to separate search and solution spaces by the genotype – phenotype mapping. Where, the genotype (binary code) generated by the GA is mapped on to phenotype (grammar rules) by the following sequence:

1. The genotype is generated by the GA as a binary string with certain length $\mu=\lambda\times 8$ e.g 00111011,11110100, ... 00001011, where λ is the number of variables in the search space.
2. This string is then split into smaller 8 bit strings know as "codons". The number of codons in the string is $\lambda=\mu/8$
3. Each 8 bit codon is then decoded into an integer e.g. for an 8 bit binary string, the integer is between 0 and 255.
4. The rule to be selected from each BNF expression is determined as the modulus of the sequential decoded integer by the number of possible rules in that BNF expression.
5. If, at the end of the integer sequence, the grammar sentence is still incomplete (due to the presence of non-terminals), the selection continues from the beginning of the integer sequence until the sentence is complete or a certain number of loops is reached. This is known as the "wrapping" process.

The grammar rule selection process may be illustrated by an example to generate a real number. Suppose a genotype with length $\mu=5\times 8$ is produced by the GA as a binary string (11011110,00101110,01101001,11010001,00011011). The decoded integers for each sequential set of 8 bits from this string are: 11011110, 00101110, 01101001, 11010001, 00011011 = 22, 46, 105, 209, and 27. In order to generate a real number using the grammar this sequence of integers is applied to the following BNF:

<int>::= <int><int> | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<var> = <int>.<int>

The number of possible rules in the expression "<int>::= <int><int> | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0" is 11 – each rule is separated by "|". The initial sentence in this BNF is "<var> = <int>.<int>". The modulus of the first value in the decoded integer sequence given the number of available rules is (22 mod 11=0). Thus the first selected rule is "<int> => <int><int>" (the rule count starts from 0). The second selected rule is "<int> => 2" (46 mod 11=2), the third rule is "<int> => <int><int>" (105 mod 11=6), and so on. The rule selection continues until all primitives are of terminal type or a threshold for the number of wraps is reached.

The applied genotype selects 8 out of 11 unique rules to generate a value of 26.526 from the initial sentence "<var> = <int>.<int>", shown in Table (1). In this example the "wrapping" occurred once after the application of the fifth rule.

Table 1

#	modulus	applied rule
1.	22 mod 11 = 0	**<int>** => **<int><int>**.<int>
2.	46 mod 11 = 2	**<int>**<int>.<int> => **2**<int>.<int>
3.	105 mod 11 = 6	2**<int>**.<int> => 2**6**.<int>
4.	209 mod 11 = 0	26.**<int>** => 26.**<int><int>**
5.	27 mod 11 = 5	26.**<int>**<int> => 26.**5**<int>
	wrapping	wrapping
6.	22 mod 11 = 0	26.5**<int>** => 26.5**<int><int>**
7.	46 mod 11 = 2	26.5**<int>**<int> => 26.52<int>
8.	105 mod 11 = 6	26.52**<int>** => 26.526

Single-objective optimization. Single-objective optimization of functions with GE is straightforward. The GA is used to generate binary strings with the length μ representing the members in the population. Each binary string is used to select rules form a given BNF producing one or more variables. These variables are then used to evaluate the objective function in order to obtain the fitness values. The population members are then paired in order to produce the offspring population. The definition of BNF syntax–the initial grammar sentence in particular–plays an important role in improving the convergence speed of the optimization. Another important parameter is the length of the binary string μ. Basically, it defines the size of the search space (the number of 8 bit strings decoded into integers known as "codons"). The value of μ is determined by taking into account the number of optimization variables plus the desired accuracy of generated real numbers. A shorter length of binary string limits the search space, while longer length may increase it unnecessarily. In general, the length of the binary string strongly depends on the BNF syntax used.

To illustrate a single-objective optimization with GE consider the non-trivial "bump" function

$$\frac{\left|\sum_{i=1}^{n}\cos^4(x_i)-2\prod_{i=1}^{n}\cos^2(x_i)\right|}{\sqrt{\sum_{i=1}^{n}ix_i^2}}, \quad (1)$$

subject to:
$\prod_{i=1}^{n}x_i > 0.75$ and $\sum_{i=1}^{n}x_i < 15n/2$,
where $0 \le x_i \le 10, i = 1,2,\cdots,n$ and n is the number of variables, see [17].

The number of variables in this example is set to n=2. Suppose that the required accuracy of the generated variable is four digits after the decimal point. The following BNF is to be used:

<int> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<vars>= x_1, x_2

where x_i = <int>.<int><int><int><int> and $0 \le x_i < 10$.

The search completion criterion is set to 25,000 evaluations. The crossover probability is set to p_c=0.2 with one point crossover. The binary string length is set to $\mu=10\times 8=80$. The mutation type is selected as multiple with probability p_m=0.05.

Unlike single mutation, where a randomly selected bit from the binary string mutates with the probability p_m in multiple mutations, each bit in the string with length μ mutates with the same probability. The use of multiple mutations with high probability appears to be beneficial to GE optimization, when the binary string length is relatively short. It introduces rather random search behaviour. Some results supporting this statement are presented here, but further investigation is needed.

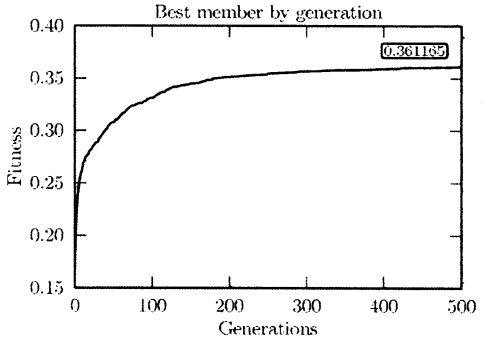

Figure 1: Averaged results from 50 runs of the "bump" function with 2 variables.

Also in this example the best member in the "parent" population is directly copied to the offspring population without applying any crossover or mutation operations (elitism).

The optimal solution in the maximization problem is 0.36485 located at x_1=1.5903, x_2=0.4716, see [17]. The GE being a stochastic search method the average of 50 runs is taken. The results are shown in Figure (1). The average location of the discovered optima is x_1=1.5813, x_2=0.4745.

Multi-objective optimization. In this implementation of a multi-objective GE, the multi-objectiveness is achieved using the Non-dominated Sorting Genetic Algorithm II (NSGAII) [18]. Here, the GA engine, which is basically the core of GE, is replaced by NSGAII. The main difference between the GA and the NSGAII is that the later uses a ranking selection strategy. In NSGAII each member from the mating population pool formed by parents and offsprings is ranked taking into account the fitness of all objectives.

The ranking values are obtained by facing each member in terms of its fitness values with other members from the mating population pool. In this way the non-dominated members are positioned on the same Pareto optimal front i.e. assigned the same rank. The best performing members from the first Pareto optimal front receive rank 1. The members on second Pareto optimal front receive rank 2 and so on. This is illustrated in Figure (2). The new offspring population is formed by members with the lowest possible rank. NSGAII is known to be an "elitist" reproduction strategy, since is keeps "alive" only the best performing members. Detailed description on the technical implementation of NSGAII is given in [18].

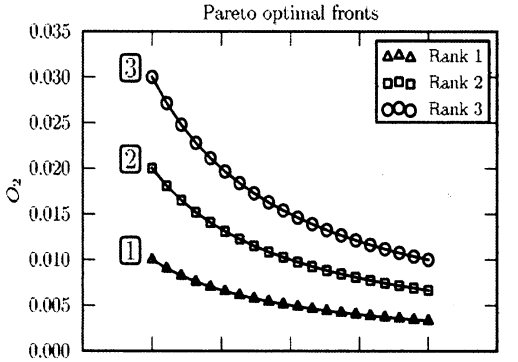

Figure 2: An example of ranked Pareto optimal fronts.

RESULTS AND DISCUSSION

Results. In the following single-objective optimization conducted using the "bump" test function with n = 20 variables, the binary string length is set to μ=100×8= 800. The crossover rate is set to p_c=0.5 with five point crossover. The multiple mutation probability is p_m=0.003. Also the best member in the "parent" population is directly copied to the offspring population without applying any crossover or mutation operations. The search completion criterion is set to 60,000 evaluations.

The averaged results of 50 runs are shown in Figure (3). The discovered optimum for the "bump" function with n=20 variables is 0.76077 located at (3.2373, 3.1054, 3.1172, 3.2016, 2.9362, 2.8992, 2.9273, 3.0574, 3.0692, 0.2927, 0.3766, 0.6960, 0.4900, 0.4631, 0.4247, 0.2768, 0.4468, 0.3423, 0.5804, 0.2556). During experimental runs it was discovered that the true optima seems to be somewhere around 0.78.

The BNF used to generate these variables is

<int> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<vars>= $x_1, x_2, \cdots, x_{20}$,

where x_i = <int>.<int><int><int><int> and $0 \leq x_i < 10$.

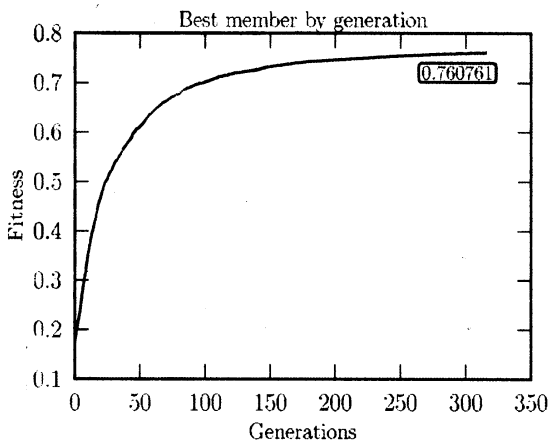

Figure 3: Averaged results from 50 runs of "bump" function with 20 variables.

The first multi-objective optimization is conducted using the test functions,

$$f_1(x) = 1 - \exp(-4x_1)\sin(6\pi x_1)^6$$
$$f_2(x) = g\left(1 - (f_1/g)^2\right)$$
$$g(x) = 1 + 9\left(\sum_{i=2}^{10}(x_i/9)\right)^{0.25}, \quad (2)$$

where $0 \leq x_i \leq 1$ and $i = 1, 2, \cdots, 10$.

The BNF used here is:

<int> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<vars>= $x_1, x_2, \cdots, x_{10}$,

where x_i = 0.<int><int><int> and $0 \leq x_i < 1$.

The binary string length is μ=30×8=240. The crossover rate is p_c=0.5 with five point crossover. The multiple mutation probability is p_m=0.003. The search completion criterion is 25,000 evaluations.

The averaged Pareto front from 50 runs is shown in the Figure (4).

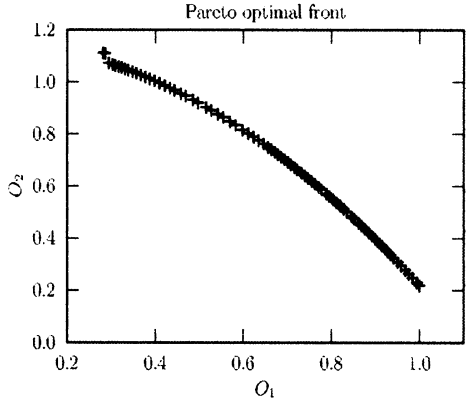

Figure 4: Averaged Pareto optimal front from 50 runs of the bi-objective optimization test function presented in Equation (2).

The second multi-objective optimization experiment is conducted using the test functions,

$$f_1(x) = \sum_{i=1}^{n-1}\left(-10\exp\left(-0.2\sqrt{x_i^2 + x_{i+1}^2}\right)\right)$$
$$f_2(x) = \sum_{i=1}^{n}\left(|x_i|^{0.8} + 5\sin(x_i)^3\right), \quad (3)$$

where $-5 \le x_i \le 5$ and $i = 1,2,3$.

The used BNF here is:

<int> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<r> ::= 1 | 2 | 3 | 4 | 0 | -0 | -1 | -2 | -3 | -4
<vars>= x_1, x_2, x_3

where x_i = <r>.<int><int><int> and -5 < x_i < 5.

The binary string length this time is μ=15×8=120. The crossover rate is p_c=0.2 with one point crossover. The multiple mutation probability is p_m=0.003. The search completion criterion is again 25,000 evaluations.

The averaged Pareto front from 50 runs is shown in Figure (5).

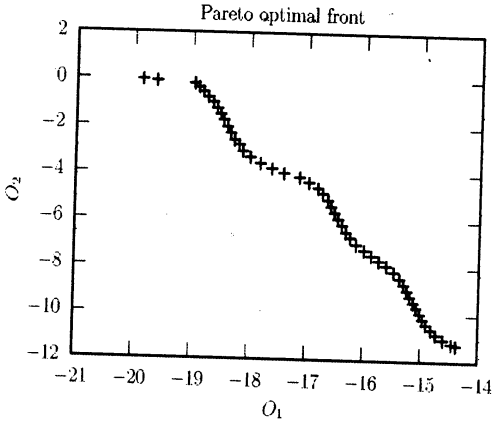

Figure 5: Averaged Pareto optimal front from 50 runs of the bi-objective optimization test function presented in Equation (3).

Discussion. There has been much discussion about the efficacy of single point crossover operation in GP. One of the studies introduces a new improved crossover operation [19], while another study remarks on the importance of the traditional single point crossover to GP [20]. In some of the optimization experiments presented above a 5 point crossover operation with probability p_c=0.5 was used successfully. The use of multiple point crossovers appears to be beneficial when the binary string length is particularly long.

CONCLUSIONS

GE can be used as a numerical optimization tool. During our experiments, it was noticed that the convergence speed in optimization with GE strongly depends on the selected BNF syntax, the crossover type, and the rate. Other important parameters are the length of the binary string μ and the multiple mutation probability p_m.

Our experimental finding was that the crossover is a rather essential part of GE. In GE, the selection of crossover type and rate strongly depend on binary string length and the BNF syntax used , i.e. it is problem specific.

ACKNOWLEGMENTS

This work was supported by EPSRC grant EP/E004547/1: *The role of topology and shape in structural design.*

REFERENCES

1. Chomsky N. Three models for the description of language. IRE Transactions on Information Theory, 1956, 2(3):113-124
2. Backus J. W. The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference, IFIP Congress, 1959, p.125-131
3. Koza J., Poli R. Genetic programming. Search Methodologies, 2005, p.127-164
4. O'Neill, M. and Ryan, C. Grammatical Evolution: Evolutionary automatic programming in an arbitrary language. Springer Netherlands, 2003
5. Goldberg D.E. Genetic algorithms in search, optimization, and machine learning. Addison-wesley, 1989
6. Ryan C., Collins J.J., O'Neill M. Grammatical Evolution: Evolving Programs for an Arbitrary Language. Lecture Notes in Computer Science 1391. First European Workshop on Genetic Programming 1998
7. O'Neill M., Ryan C. Grammatical Evolution: A Steady State approach. In Proceedings of the Second International Workshop on Frontiers in Evolutionary Algorithms 1998, p.419-423
8. O'Neill, M. and Ryan, C. Grammatical evolution. IEEE Transactions on Evolutionary Computation, 2002, 5(4):349-358
9. http://www.grammatical-evolution.org/pubs [05 Feb. 2011]
10. de la Puente, A.O. Alfonso, R.S. Moreno, M.A. Automatic composition of music by means of grammatical evolution. Proceedings of the 2002 Conference on APL: array processing languages: lore, problems, and applications, 2002, p.148-155
11. Brabazon, A. and O'Neill, M. Biologically inspired algorithms for financial modelling. Springer-Verlag New York Inc., 2006
12. O'Neill, M. et al. Evolving market index trading rules using grammatical evolution. Applications of evolutionary computing, Springer, 2001, p.343-352
13. O'Neill M., Collins J.J., Ryan C. Automatic Generation of Robot Behaviours using Grammatical Evolution in Proceedings of AROB 2000. The Fifth International Symposium on Artificial Life and Robotics, p.351-354
14. O'Neill, M. et al. Shape grammars and grammatical evolution for evolutionary design. Proceedings of the 11th Annual conference on Genetic and evolutionary computation, 2009, p.1035-1042
15. O'Neill, M. et al. Evolutionary design using grammatical evolution and shape grammars: Designing a shelter. International Journal of Design Engineering, 2010, 3(1):4–24
16. Murphy, J. and O'Neill, M. and Carr, H. Exploring grammatical evolution for horse gait optimization. Genetic Programming, 2009, p.183-194
17. Keane A.J. and Nair P.B. Computational approaches for aerospace design, the pursuit of Excellence. John Wiley & Sons, 2005
18. Deb K. et al. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. Parallel Problem Solving from Nature PPSN VI, 2000, p.849-858
19. Francone F.D. et al. Homologous crossover in genetic programming Proceedings of the Genetic and Evolutionary Computation Conference, 1999, p.1021-1026
20. O'Neill M. et al. Crossover in grammatical evolution: The search continues. Genetic Programming, 2001, p.337-347