Symbolic Regression by Grammar-based Multi-Gene Genetic Programming

Jan Žegklitz
Czech Technical University in Prague
Technická 2, Prague 6, Czech Republic
zegkljan@fel.cvut.cz

Petr Pošík
Czech Technical University in Prague
Technická 2, Prague 6, Czech Republic
petr.posik@fel.cvut.cz

ABSTRACT

Grammatical Evolution is an algorithm of Genetic Programming but it is capable of evolving programs in an arbitrary language given by a user-provided context-free grammar. We present a way how to apply Multi-Gene idea, known from Multi-Gene Genetic Programming, to Grammatical Evolution, just by modifying the given grammar. We also describe modifications which improve the behavior of such algorithm, called Multi-Gene Grammatical Evolution. We compare the resulting system to GPTIPS, an existing implementation of MGGP.

CCS Concepts

•**Computing methodologies** → **Supervised learning by regression; Genetic programming;**

Keywords

machine learning; symbolic regression; genetic programming; grammatical evolution

1. INTRODUCTION

Symbolic regression (SR) is a method based on genetic programming (GP) [6] that solves a regression task by evolving symbolic mathematical expressions as the models of the given data. A successful solver of SR tasks is the multi-gene variant of GP, MGGP, evolving a model in the form of linear combination of several nonlinear components. Except GP, there are many other paradigms evolving models in the form of analytical expressions, with more or less different principles, which can also profit from the use of the multi-gene approach. In this article we explore the possibility of applying the multi-gene (MG) idea to one of those GP alternatives, grammatical evolution (GE). After reviewing GE and MGGP in Sec. 2, in Sec. 3 we propose a simple way of turning a GE system into MGGE based solely on the grammar manipulation. We also discuss the differences between MGGP and MGGE resulting from a different representation and operators. In Sec. 4 we experimentally compare two variants of the resulting algorithm with GPTIPS, an existing implementation of an MGGP system. Section 5 concludes the paper.

GECCO '15, July 11 - 15, 2015, Madrid, Spain

ISBN 978-1-4503-3488-4/15/07…$15.00

DOI: http://dx.doi.org/10.1145/2739482.2768484

2. RELATED WORK

In the two following subsections we review the two key components in our research, the Grammatical Evolution and Multi-Gene Genetic Programming.

2.1 Grammatical Evolution

Grammatical Evolution (GE) [9] is a GP algorithm that uses context-free grammars in Backus-Naur Form (BNF) and simple linear genotypes for the representation of individuals, rather than trees as in ordinary GP. The form of the solutions produced by GE is determined mainly by the user-provided grammar. Thanks to this fact, the user has full control over how the solutions look like. BNF grammar can also be used to implement typing, allowing the algorithm to behave like Strongly Typed GP [8].

2.2 Multi-Gene Genetic Programming

MGGP [5, 11, 10] is an extension to the classical GP utilizing the power of least-squares linear regression.

In ordinary GP (in the context of symbolic regression), a candidate solution is a single mathematical function that takes rows of an $N \times M$ feature matrix (N being the number of datapoints, M the number of feature variables) and produces an $N \times 1$ vector of estimated target variable.

In MGGP, each solution is formed by a linear combination of one or more such functions, called genes. Such a solution has a form

$$\hat{\mathbf{y}} = b_0 + b_1 g_1(\mathbf{x}) + b_2 g_2(\mathbf{x}) + ... + b_n g_n(\mathbf{x}) \qquad (1)$$

where n is the number of genes in the particular solution. However, the vector of coefficients $\mathbf{b}$ is not known and must be found during the evaluation. Each gene is applied to the feature matrix the same way as in the ordinary GP, producing an $N \times 1$ vector $\mathbf{g_i}$ (i is the number of the particular gene). The output $\hat{\mathbf{y}}$ of the whole solution is then given by the formula $\hat{\mathbf{y}} = \mathbf{G}\mathbf{b}$ where $\mathbf{G} = [\mathbf{1}\ \mathbf{g_1}\ \mathbf{g_2}\ \cdots\ \mathbf{g_n}]$ with $\mathbf{1}$ being an $N \times 1$ vector of ones. The optimal coefficient vector $\mathbf{b}$ can then be found using the least-squares estimation with respect to the true target vector $\mathbf{y}$:

$$\mathbf{b} = (\mathbf{G}^\top \mathbf{G})^{-1} \mathbf{G}^\top \mathbf{y} \qquad (2)$$

In practice the Moore-Penrose pseudo-inverse is used instead of standard matrix inverse in Equation 2 because the columns of **G** may be collinear (e.g. due to multiple genes being effectively identical).

The advantage of MGGP is that the genetic search does not have to find the linear parts as they are computed automatically, and it just has to focus on capturing the nonlinear characteristics of the solved problem.

3. MG GRAMMATICAL EVOLUTION

There is no reason the MG approach couldn't be used with other paradigms than pure GP, especially with GE. We propose a new approach called Multi-Gene Grammatical Evolution, or MGGE, which is a result of application of the multi-gene principle to GE.

The most straightforward way of making the GE multi-gene is by following exactly the same type of modification as in MGGP: the individuals will not carry only one genotype describing one expression, but more genotypes describing the particular expressions. However, this modification requires the inner workings of GE to be modified. Instead of such modification, we took another approach which uses already available tools to express the multi-gene nature.

3.1 MG individuals by grammar manipulation

In order to express a multi-gene individual by already existing tools available in GE, we decided to incorporate the „multigeneness“ into the grammar used by the GE algorithm. We developed a simple procedue which takes the original grammar, called *base grammar*, and transforms it into a new grammar, called *multi-gene grammar*, which already provides rules for storing multiple genes. The procedure is described in Algoritm 1.

Algorithm 1 Grammar manipulation procedure that transforms a *base grammar* to a *multi-gene grammar*.

Input: a *base grammar* g_b with start symbol $S(g_b)$, maximum number of genes G_{max}
Output: a *multi-gene grammar* g_{mg} with start symbol $S(g_{mg})$
$g_{mg} \leftarrow g_b$ (copy the base grammar)
add rule `<gene> ::=` $S(g_b)$ to g_{mg}
add rule
`<mg-start> ::= <gene> | <gene>#<gene>`
`| ... | <gene>#...#<gene>` (G_{max}-times)
$S(g_{mg}) \leftarrow$`<mg-start>`

The base grammar is transformed by being put „under“ a new rule `<gene>`, and another new rule, `<mg-start>`, which expands to all possible numbers of genes, is added to the grammar and set as its start symbol. The `#` character is used just as a textual separator of the genes. The parser, of course, needs to be modified to cope with such modified grammar.

By using the grammar modification, the GE algorithm itself doesn't need any modification to be able to work, only the parser needs to be changed and the least-squares fitting mechanism must be introduced.

3.2 MG crossover operators

In MGGP the crossover operator is probabilistically chosen to be one of these:

- high (or gene) level crossover – one or more genes in both parents are exchanged, some genes are deleted if G_{max} is exceeded
- low level crossover – picks one gene in both parents and a subtree in each of those two genes is exchanged

But GE has its own set of possible crossover operators:

- ripple crossover – both parent genotypes are split into two parts and the tails are exchanged
- subtree crossover – GP-like crossover but working on the derivation tree rather than the parse tree; can be performed directly with the linear representation [4]

The MGGE can then either be run as an ordinary GE algorithm, i.e. using ripple or subtree crossover (or some combination of them), or the operators from MGGP can be adapted and used. The low level crossover is identical to the subtree crossover except for a constraint that is needed to force a subtree to be chosen at a non-terminal label that is „below“ the `<gene>` rule to prevent accidentally swapping whole genes. The high level crossover can be implemented by modifying the subtree crossover:

- the subtrees are restricted to be rooted only in the `<gene>` non-terminals
- more than one subtree can be selected in either parent; all subtrees are then swapped

4. EXPERIMENTAL EVALUATION

To gauge the performance of MGGE, we performed a series of experiments. The algorithm was tested on four datasets, two artifically generated and two real-world ones. The description of the datasets follows.

Sextic polynomial (6R): classical SR benchmark [7]. The dataset consists of 100 (x, y) pairs, x is sampled form the interval $[-1, 1]$, y is given by the formula $y = x^6 - 2x^4 + x^2$.

2D unwrapped ball function (2UB): The dataset consists of 1000 $(\mathbf{x}, y)$ pairs, $\mathbf{x}$ is sampled using the 2D Halton sequence [3], y is given by the formula $y = \frac{10}{5+(x_1-3)^2+(x_2-3)^2}$.

Forest Fires (FF): This dataset [2] retrieved from the UCI repository [1] is a real-world regression dataset where the task is to predict the burned area of the forest. All features are numeric except the 3rd and 4th features which are month („jan“ to „dec“) and day („mon“ to „sun“) respectively. These were transformed to numbers by mapping the month to the numbers 1 to 12 („jan“ being mapped to 1, „dec“ being mapped to 12) and day to the numbers 1 to 7 („mon“ being mapped to 1, „sun“ being mapped to 7).

Airfoil Self-Noise (ASN): Retrieved from the UCI repository, this is a real-world regression dataset where the task is to predict a sound pressure level of an airfoil in a wind tunnel experiment.

Note that our experiments focus solely on the ability of the algorithms to fit the expression as closely as possible, so no validation or testing datasets were used. Thus the resultant models cannot be used as predictive models.

4.1 Algorithms

For each dataset we tested the following algorithms for comparison:

- pure GE using ripple crossover (GE)

- MGGE with ripple crossover (MGGE-r)
- MGGE with the multigene crossover operators (MGGE)
- GPTIPS2, a MATLAB implementation [11, 10] of MGGP (GPTIPS)

parameter	value
pop. size	100
crossover prob.	0.8
prob. of high level crossover (MGGE)	0.2
mutation prob. (per codon)	0.08
duplication prob.	0.1
pruning prob.	0.2
no. of elites	2
tournament size	4
max. no. of genes (MGGE-ripple, MGGE)	10
max. tree depth (GPTIPS)	18
max. mutation tree depth (GPTIPS)	10

Table 1: Settings of the tested algorithms. If a parameter is applicable to only some of the algorithms, they are stated in the parentheses.

The settings of all algorithms is in the Table 1 as the most of the parameters were set identically for all the algorithms.

For all algorithms we used Ramped Half'n'Half initialization [6] with maximum depth of 18 for GE (in terms of the derivation tree) and GPTIPS (in terms of the expression tree) and with minimum depth of 3 and maximum depth of 20 for MGGE-ripple and MGGE (because of the extra rules).

In all algorithms the same (base) grammar (see Figure 1) was used. GPTIPS used an equivalent set of functions, i.e. $+$, $-$, $\cdot$, $\div$, $\sin(x)$, $\cos(x)$, $\arctan(x)$, $\sqrt{x}$, e^x, $\ln x$, $|x|$.

```
  <E>   ::=  (<E><OP><E>) | <F>(<E>) | <C> | <V>
 <OP>   ::=  + | - | * | /
  <F>   ::=  sin | cos | arctan | sqrt
           |  exp | log | abs
  <V>   ::=  x1 | x2 | ...
  <C>   ::=  0.1 | 0 | 1
```

Figure 1: Grammar used in experiments. The `exp` and `log` are natural, i.e. e^x and $\log_e(x)$ respectively.

In all algorithms if an expression resulted in a mathematical error (e.g. division by zero) it received infinite fitness, effectively killing such individual.

4.2 Results

The evolution plots for the problems 6R, 2UB, FF and ASN can be seen in the Figures 2, 3, 4, 5 respectively.

It is more than clear that all the multigene algorithms completely outperform the ordinary GE algorithm. This is due to the least squares optimization which makes even bad solutions good compared to those of GE.

GPTIPS outperformed both MGGE algorithms in all domains. This may have several reasons. First of them is the fact that GPTIPS works with Koza-style trees, while GE-based algorithms work with linear chromosome (if ripple crossover is considered) or with derivation trees, i.e. trees of

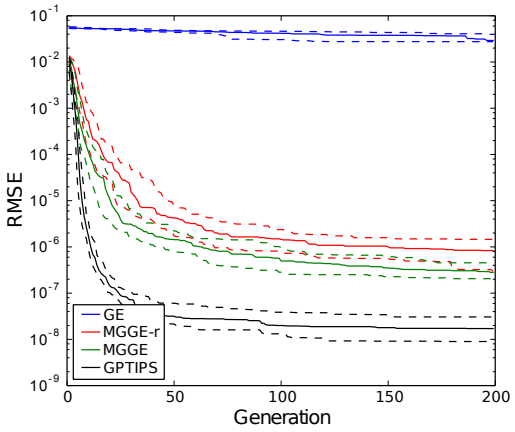

Figure 2: The evolution plot of the 6R problem. The solid lines are the medians of 50 runs, the dashed lines are the 1st and 3rd quartiles.

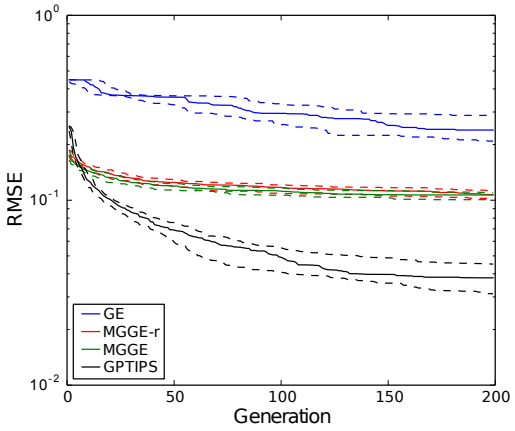

Figure 3: The evolution plot of the 2UB problem. The solid lines are the medians of 50 runs, the dashed lines are the 1st and 3rd quartiles.

the transcription process from the start non-terminal down to the terminals (if subtree crossover is considered). The second reason is the mutation operator. In GPTIPS, classical GP mutation is employed, i.e. replacing a subtree with a randomly generated one. In GE-based algorithms the mutation simply changed one codon to a different random number which is likely to cause the rest of the chromosome to be interpreted in a different way.

We also note that if a wall-clock time was measured (which we didn't) instead of the number of generations, the results could look differently. Also only single parameter setting was tested; if the parameters were tuned for each algorithm separately, the results could, again, look differently.

5. CONCLUSION AND FURUTRE WORK

In this article we proposed a method of transferring the multi-gene principles to GE – MGGE – only by means of transforming the base grammar in a way that it enables encoding of multiple genes. We presented two variants of MGGE, the first one being effectively a pure GE, the second one using specialized multi-gene operators.

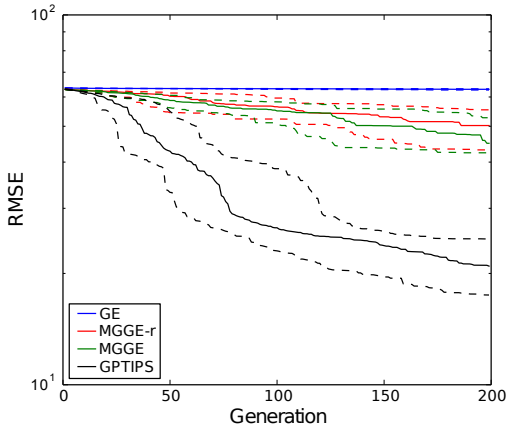

Figure 4: The evolution plot of the FF problem. The solid lines are the medians of 50 runs, the dashed lines are the 1st and 3rd quartiles.

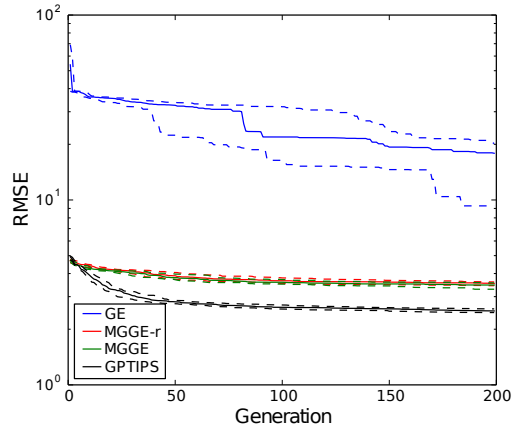

Figure 5: The evolution plot of the ASN problem. The solid lines are the medians of 50 runs, the dashed lines are the 1st and 3rd quartiles.

We compared GE, both MGGEs and GPTIPS, an existing implementation of MGGP, on four regression datasets. We showed that the MGGEs outpeform the GE but are outperformed by GPTIPS. The reasons for this fact are not completely understood, but among the most likely causes are the mutation operator and the fact that the GE algorithms work with grammar, might be very different (depends the particular grammar) than the Koza-style GP trees.

5.1 Future work

The structure of the grammar may have significant impact on the behaviour of the algorithm, e.g. if the grammar contains only a single non-terminal (except extra non-terminals needed by MGGE) the subtree crossover is then equivalent to the GP subtree crossover. However, the grammar we used allowed e.g. to change an operator to a different one with one subtree crossover. Investigation in the area of the grammar structure and its influence on the performance is needed.

In this article we tested the algorithms on low-dimensional problems only. Dealing with high number of dimensions is one of the current challenges in symbolic regression and the algorithms need to be tested on such data too.

Another area of research are the operators. In our research we used either the GE's ripple crossover or the MGGP's high and low level crossover and in all GE-based algorithms we used codon-level mutation. However, it might be interesting e.g. to adapt ripple crossover to work restricted to a gene, or to use a different mutation with less disturbing effects.

6. ACKNOWLEDGMENTS

Jan Žegklitz was supported by the Czech Science Foundation project Nr. 15-22731S. Petr Pošík was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS14/194/OHK3/3T/13.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme „Projects of Large Infrastructure for Research, Development, and Innovations“ (LM2010005), is greatly appreciated.

7. REFERENCES

[1] K. Bache and M. Lichman. UCI machine learning repository, 2013. http://archive.ics.uci.edu/ml.

[2] P. Cortez and A. d. J. R. Morais. A data mining approach to predict forest fires using meteorological data. 2007.

[3] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1):84–90, 1960.

[4] R. Harper and A. Blair. A structure preserving crossover in grammatical evolution. In *2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2537–2544, 2005.

[5] M. Hinchliffe, H. Hiden, B. McKay, M. Willis, M. Tham, and G. Barton. Modelling chemical process systems using a multi-gene genetic programming algorithm. In *Late Breaking Paper, GP'96*, pages 56–65, Stanford, USA, 1996.

[6] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[7] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

[8] D. J. Montana. Strongly typed genetic programming. *Evolutionary computation*, 3(2):199–230, 1995.

[9] C. Ryan, J. Collins, and M. O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Genetic Programming*, volume 1391 of *Lecture Notes in Computer Science*, pages 83–96. Springer Berlin Heidelberg, 1998.

[10] D. P. Searson. GPTIPS2: an open-source software platform for symbolic datamining. In A. H. Gandomi, A. H. Alavi, and C. Ryan, editors, *Springer Handbook of Genetic Programming Applications*. 2015. In press.

[11] D. P. Searson, D. E. Leahy, and M. J. Willis. GPTIPS: an open source genetic programming toolbox for multigene symbolic regression. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, pages 77–80, March 2010.