GDF: A tool for function estimation through grammatical evolution

Ioannis G. Tsoulos [(1)]*, Dimitris Gavrilis[(2)], Evangelos Dermatas[(2)]

[(1)]Department of Computer Science, University of Ioannina, P.O. Box 1186, Ioannina 45110 - GREECE,
[(2)]Department of Electrical & Computer Egineering, University of Patras, Patras 26500 - GREECE

Abstract

This article introduces a tool for data fitting that is based on genetic programming and especially on the grammatical evolution technique. The user needs to input a series of points and the accompanied dimensionality n and the tool will produce via the genetic programming paradigm a function $f : R^n \to R$ that is optimal in the least squares sense. The tool is entirely written in ANSI C++ and it can be installed in every UNIX system.

PACS:02.30.Mv;02.60.-x;02.60.Ed ;07.05.Mh;

PROGRAM SUMMARY

Title of program: GDF

Program available from: CPC Program Library, Queen's University of Belfast, N. Ireland.

Computer for which the program is designed and others on which it has been tested: The tool is designed to be portable in all systems running the GNU C++ compiler.

Installation: University of Ioannina and University of Patras, Greece.

Programming language used: GNU-C++.

Memory required to execute with typical data: 200KB.

No. of bits in a word: 32

*Corresponding author. Email: sheridan@cs.uoi.gr

No. of processors used: 1

Has the code been vectorised or parallelized?: No.

No. of bytes in distributed program,including test data etc.: 30 Kbytes.

Distribution format: gzipped tar file.

Keywords: Function approximation, stochastic methods, genetic programming, grammatical evolution.

Method of solution: Genetic programming.

LONG WRITE UP

1 Introduction

The problem of function estimation consists of finding a function that will best approximate a set of n-dimensional points given their output. Function estimation finds many applications in physics, chemistry, signal processing etc. and it can be formulated as follows: *Given M points and associated values (x_i, y_i), $i = 1, \ldots, M$ with $x_i \in R^n$ estimate a function $f : R^n \rightarrow R$ that minimizes the least squares "Error"*

$$E_T = \sum_{i=1}^{M} \left(f\left(x_i\right) - y_i\right)^2 \tag{1}$$

Through the years many methods have been proposed for this problem, such as spline based [1, 2] or neural network based [3, 4]. Although these techniques have been applied successfully to many data fitting problems, they produce functional forms which consist of applications of specific functions such as polynomials or sigmoidal functions. The proposed programming tool takes as input the points (x_i, y_i) and creates a functional form that minimizes the quantity in equation 1 through the procedure of Grammatical Evolution. Grammatical Evolution is an evolutionary processes that can create programs in an arbitrary language. The production is performed using a mapping process governed by a grammar expressed in Backus Naur Form. Grammatical evolution has been applied successfully to problems such as symbolic regression [6], discovery of trigonometric identities [7], robot control [8], caching algorithms [9], financial prediction etc. The rest of this article is organized as follows: in section 2 the contents of the distribution are presented, in section 3 the installation steps for any UNIX programming environment are expressed in detail, in section 4 the main parts of the distribution such as the underlying algorithm, the grammar specifications and the `gdf` program are thoroughly analyzed and finally in section 5 some conclusions from the application of the tool are listed.

2 Distribution

The package is distributed in a tar.gz file named `GDF.tar.gz` and under UNIX systems the user must issue the following commands to extract the associated files:

1. gunzip `GDF.tar.gz`
2. tar xfv `GDF.tar`

These steps create a directory named `GDF` with the following contents:

1. **bin**: It is a directory which is initially empty. After the compilation of the package it will contain the executable `gdf` and the text file named grammar.txt. The first file is the created programming tool and the second one is an auxiliary file that contains the grammar of the tool, expressed in BNF format.
2. **doc**: This directory contains the documentation of the package (this file) in different formats: A LyX file, A LaTeX file and a PostScript file.
3. **include**: The directory which contains the header files for all the classes of the package.
4. **src**: The directory with the source files of the package.
5. **Makefile**: It is the file that will be read by the make utility in order to built the tool. There is no need for the user to modify this file.

3 Installation

The following steps are required in order to build the tool:

1. Uncompress the tool as described in the previous section.
2. Issue the command `cd GDF`
3. Type `make`.

After the compilation the binary file `gdf` will be placed in the bin subdirectory accompanied with the text file `grammar.txt` The tool is entirely written in GNU C++ version 3.2.3, but it can be installed in systems with different ANSI C++ compiler. The only modification required is to replace the line

```
CC = c++
```

in the file `Makefile` under the `src` subdirectory with the following one

```
CC = mycpp
```

where `mycpp` is the name of the corresponding ANSI C++ compiler in the hosting operating system.

4 Program details

4.1 The underlying algorithm

The programming tool is based on the following stochastic algorithm:

Initialization Step:

1. The program reads the data to be fitted from a text file.

2. The program reads the used grammar from a text file.

3. Every chromosome in the genetic population is initialized. The initialization is performed by a randomly selection of a number in the range [0,255] for every element of each chromosome.

4. The values for the parameters **selection rate** and **mutation rate** are selected. The values for these parameters must be in the range (0,1).

5. Set $k = 0$, where k is the amount of the generations.

6. Set the value for the parameter maxK, where maxK is the maximum allowed number of generations.

Evolution Step:

1. For every chromosome in the population, a function is created through the process of Grammatical Evolution.

2. The fitness of each member of the population is evaluated.

3. The chromosomes are sorted in descending order according to their fitness values.

4. Some chromosomes are selected through tournament selection. The selected chromosomes create the mating pool.

5. A bunch of
$$g = (1 - \text{selection rate}) \times \text{population size}$$
chromosomes is created and it replaces the corresponding worst chromosomes in the population.

6. The mutation procedure is applied to each member of the population with probability equal to **mutation rate**.

7. **Set** $k = k + 1$

8. If $k > \text{maxK}$ or the best fitness value has fallen below a predefined threshold, then the **Evolution** Step is terminated.

Figure 1: Grammar specification

```
<S> ::= <expr>
<expr> ::= ( <expr> )
           | <expr> <op> <expr>
           | <func> ( <expr> )
           | <terminal>
<op> ::= +
<op> ::= -
<op> ::= *
<func> ::= sin
<func> ::= cos
<func> ::= log10
```

4.2 Grammar specification

The file that contains the grammar specification must be determined by the user with the -g option from the command line. The grammar must be specified in any simple text (ASCII) file with the format shown in figure 1.

The start symbol (<S>) is required and must be specified in the above form. The start symbol can give only one non-terminal symbol (e.g. <expr>). The available non-terminal symbols are <expr>, <func>, <op>. The available terminal symbols are +,-,*,/,(,). The numbers are represented as lists of digits (including ".") and can be specified by <terminal>. The available functions are: sin, cos, log, exp, log10, tan, abs, sqrt, int, atan, acos, asin. If a non-teminal specification has more that one rules, those rules can be specified with a "|" instead of typing the entire left hand (e.g. in the second rule of <expr> , "<expr> ::=" is replaced by "|"). In this way, the user can easily alter the program parameters by specifying a different grammar. If, for example, it is known that log or log10 cannot exist in the desired output, the user can remove them from the grammar specification.

4.3 The main program gdf

The created executable gdf takes the following series of parameters in the command line

1. **-h**: The program prints a help screen to the user with a description for each command line parameter and it terminates.

2. **-g** grammar_file: The parameter grammar files determines a file with a valid grammar for the tool. The user must have read access to the specified file.

3. **-p** problem_file: The parameter problem_file determines a file containing the points where the data fitting procedure will be applied. The user

Figure 2: Data Fitting Format

$$
\begin{array}{l}
\mathrm{D} \\
\mathrm{M} \\
\begin{array}{ccccc}
x_{11} & x_{12} & \cdots & x_{1D} & y_1 \\
x_{21} & x_{22} & \cdots & x_{2D} & y_2 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
x_{M1} & x_{M2} & \cdots & x_{MD} & y_M
\end{array}
\end{array}
$$

must have read access to the specified file and the contents of the file must conform to the format of the figure 2. The integer number D in the file determines the dimensionality of the specific problem, the number M determines the amount of points in the file and each consecutive line defines a point where the data fitting procedure will be applied. This parameter is the only one required from the program

4. **-t** test_file: The parameter test_file determines a file in the same format as the problem_file, where the produced function will be tested after the termination of the genetic algorithm. The user must have read access to the specified file and the dimension in the file test_file must be the same with that of the problem_file.

5. **-c** count: The parameter count specifies the number of chromosomes in the genetic population. The default value for this parameter is 500.

6. **-l** length: The parameter length specifies the length of each chromosome in the genetic population. The default value for this parameter is 100. The choice of chromosomes with static length was made in order to prevent the tool from creating extremely large chromosomes.

7. **-s** srate: The parameter srate specifies the value for the parameter **selection rate** of the genetic algorithm. The default value for this parameter is 0.1

8. **-m** mrate: The parameter mrate determines the value for the parameter mutation rate of the genetic algorithm. The default value for this parameter is 0.05

9. **-r** seed: The parameter seed specifies the seed for the random number generator. The default value for this parameter is 1.

The maximum number of generations in the gdf program is set to 2000. In each generation the program prints in the screen the following quantities:

1. The number of generations passed.

2. The best discovered function.

3. The fitness value of the best discovered function.

4.4 Test runs

The performance of the proposed tool was measured by using 5 different datasets: one for the continuous function $f(x) = x\sin(x^2)$ and 4 real life problems.

4.4.1 The function $f(x) = x\sin\left(x^2\right)$

The tool was tested on this function using a dataset with 100 random points from the function in the range [-2,2]. The tool was issued with the following command:

```
./gdf -p xsinxx.data
```

where xsinxx.data is the file containing the points for the data fitting. The last 10 lines from the output of the above program are the following:

```
generation=156 f(x)=sqrt(sqrt((log(2.72))))*sin(abs(exp(log(((x1))*x1))))*x1 fitness=-1.260575367e-06
generation=157 f(x)=sqrt(sqrt((log(2.72))))*sin(abs(exp(log(((x1))*x1))))*x1 fitness=-1.260575367e-06
generation=158 f(x)=sqrt(sqrt((log(2.72))))*sin(abs(exp(log(((x1))*x1))))*x1 fitness=-1.260575367e-06
generation=159 f(x)=sqrt(sqrt((log(2.72))))*sin(abs(exp(log(((x1))*x1))))*x1 fitness=-1.260575367e-06
generation=160 f(x)=sqrt(sqrt((log(2.72))))*sin(abs(exp(log(((x1))*x1))))*x1 fitness=-1.260575367e-06
generation=161 f(x)=sqrt(sqrt((log(2.72))))*sin(abs(exp(log(((x1))*x1))))*x1 fitness=-1.260575367e-06
generation=162 f(x)=sqrt(sin((log(4.72))))*sin(abs(exp(log(((x1))*x1))))*x1 fitness=-4.10576003e-07
generation=163 f(x)=sqrt(sin((log(4.72))))*sin(abs(exp(log(((x1))*x1))))*x1 fitness=-4.10576003e-07
generation=164 f(x)=sqrt(sin((log(4.72))))*sin(abs(exp(log(((x1))*x1))))*x1 fitness=-4.10576003e-07
generation=165 f(x)=sqrt(sin((log(4.82))))*sin(abs(exp(log(((x1))*x1))))*x1 fitness=-4.832536538e-11
```

4.4.2 The Ailerons problem

This problem has 40 attributes and consists of 7150 points. This data set addresses a control problem, namely flying a F16 aircraft. The attributes describe the status of the aeroplane, while the goal is to predict the control action on the ailerons of the aircraft. The original owner of the database is Rui Camacho (rcamacho@garfield.fe.up.pt). The program `gdf` were trained with 200 points from the dataset and the resulting expression was tested over the rest 6950 points. Ten independent experiments were conducted with different random seeds each time and in all cases the absolute value of the fitness was below 3×10^{-6}. The best discovered function was

$$f(x) = \frac{x_9}{227.3} - \frac{\sin\left(\frac{x_{16}}{\sqrt{x_{39}}}\right)}{\sqrt{1187 - x_8 - 756.92x_3}}$$

with fitness -1.65×10^{-6} and test error per point 1.07×10^{-7}. The resulting function depends only on 5 from the 40 attributes .

4.4.3 The Elevators problem

The original source of this problem is the experiments of Rui Camacho (rcamacho@garfield.fe.up.pt). The problem has 18 attributes and this data set is also obtained from the task of controlling a F16 aircraft, although the target variable and attributes are different from the ailerons domain. In this case the goal variable is related to an action taken on the elevators of the aircraft. From this dataset 200 points were used for training and 8452 for testing. The best discovered function was

$$f(x) = \frac{5.7}{866.7} - 9.8x_{13} - x_{18} + \frac{x_8}{3} - 9.91x_{10}\exp(x_4)$$

with fitness value -1.64×10^{-3} and mean test error 3.41×10^{-5}.

4.4.4 The Pyrimidines problem

The source of this dataset is the URL:

`http://www.ncc.up.pt/~ltorgo/Regression/DataSets.html`

and it is a problem of 27 attributes and 74 number of patterns. The task consists of Learning Quantitative Structure Activity Relationships (QSARs). The Inhibition of Dihydrofolate Reductase by Pyrimidines.The data are described in: King, Ross .D., Muggleton, Steven., Lewis, Richard. and Sternberg, Michael.J.E. Drug Design by machine learning: the use of inductive logic programming to model the structure-activity relationships of trimethoprim analogies binding to dihydrofolate reductase. From the above dataset 50 patterns were used for training and 24 for testing. The best discovered function was:

$$f(x) = \cos(\cos(\sqrt{x_{20}}))\cos(x_{22} - \log(\sin(\exp(x_6)))) + \frac{\sin(x_8)}{1.3}\sqrt{x_{15}\sin(\sin(x_9))}$$

with fitness value $-1.33 \times 10^{-1}$and mean test error 7.25×10^{-3}.

4.4.5 The Basketball problem

The source of this dataset is from Smoothing Methods in Statistics available from the:

`ftp://stat.cmu.edu/datasets`

which is a problem of four attributes and it tries to identify the points scored per minute from the attributes "assists per minute", "player height","time played" and "player age". From the 96 available patterns 60 were used for training and 36 for testing. The best discovered function was:

$$f(x) = \sin(8.59)\frac{\cos\left(\sqrt{\exp(\sin(x_1))}\right)}{\sqrt{\cos\left(\frac{x_3}{33.1 - 0.245\cos(x_3 x_2)}\right)}}$$

with fitness value -3.78×10^{-1} and mean test error 6.99×10^{-3}.

5 Conclusions

The introduced tool is a program aimed to fit a function in a series of points of an arbitrary dimension. The applied function is created through the evolutionary process of Grammatical Evolution and as a consequence there is no guarantee that the goal will be achieved. However, the user can apply the tool even in cases where the existence of an analytical solution is difficult to be found. Also, the tool is provided with the ability of changing the underlying grammar according to the problem.

References

[1] De Boor C., "A practical guide to splines", Springer - Verlang, New York, 1978.

[2] Kincaid D., and Cheney W., "Numerical Analysis", Brooks/Cole Publishing Company, 1991.

[3] Hornik K., Stinchcombe M., and White H., Neural Networks 2 (1989) 359.

[4] Cybenko G., "Approximation by superpositions of a sigmoidal function", Mathematics of Control Signals and Systems 2 (1989) 303-314.

[5] J. R. Koza, Genetic Programming: On the programming of Computer by Means of Natural Selection. MIT Press: Cambridge, MA, 1992.

[6] M. O'Neill and C. Ryan, Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language, volume 4 of Genetic programming. Kluwer Academic Publishers, 2003.

[7] C. Ryan, M. O'Neill, and J.J. Collins, "Grammatical Evolution: Solving Trigonometric Identities," In proceedings of Mendel 1998: 4th International Mendel Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks, Rough Sets., Brno, Czech Republic, June 24-26 1998. Technical University of Brno, Faculty of Mechanical Engineering, pp. 111-119.

[8] Collins J. and Ryan C., "Automatic Generation of Robot Behaviors using Grammatical Evolution," In Proc. of AROB 2000, the Fifth International Symposium on Artificial Life and Robotics.

[9] M. O'Neill and C. Ryan, "Automatic generation of caching algorithms," In Kaisa Miettinen, Marko M. Mkel, Pekka Neittaanmki, and Jacques Periaux (eds.), Evolutionary Algorithms in Engineering and Computer Science, Jyvskyl, Finland, 30 May - 3 June 1999, John Wiley & Sons, pp. 127-134, 1999.

[10] A. Brabazon and M. O'Neill, "A grammar model for foreign-exchange trading," In H. R. Arabnia et al., editor, Proceedings of the International conference on Artificial Intelligence, volume II, CSREA Press, 23-26 June 2003, pp. 492-498, 2003.