# Software Project Cost Estimation Using Genetic Programming

Y. Shan, R.I. McKay, C.J. Lokan, D.L. Essam, Y. Chang
School of Computer Science, UC, University of New South Wales,
ADFA, Northcott Drive, Canberra,ACT 2600, Australia

## Abstract

Knowing the estimated cost of a software project early in the development cycle is a valuable asset for management. In this paper, an evolutionary computation method, Grammar Guided Genetic Programming (GGGP) , is used to fit model, with the aim of improving the prediction of software development. Valuable result is obtained, which is much better than simple linear regression's. In this research, GGGP, because of its flexibility and the ability of incorporating background knowledge, also shows great potentials in being applied in other software engineering modeling problems.

## 1 Introduction

Knowing the estimated cost of a particular software project early in the development cycle is a valuable asset. Management can use cost estimates to evaluate a project proposal or to manage the development process more effectively. Therefore, the accurate prediction of software development cost may have a large economic impact: in fact, some 60% of large projects significantly overrun their estimates and 15% of the software projects are never completed due to the gross misestimation of development [1].

A large range of metrics have been proposed for early estimation of software project size and effort. A number of authors have suggested that the standard sets of metrics have too many parameters, and a number of reduced sets have been suggested (large metric sets have high collection costs, and also risk generating over-fitted models). The reductions have relied on linear methods to eliminate metrics, and linear models for estimating size and effort from the metric sets, but there is a risk that some of the dependencies may be non-linear. Researchers elsewhere have begun to investigate alternative methods of developing predictive models, including fuzzy logic, neural networks, and regression trees. Evolutionary approaches have not been explored. In this paper, one of evolutionary computation approaches, Grammar Guided Genetic Programming (GGGP), is used to fit nonlinear models to a dataset of past projects, with the aim of determining appropriate metric sets and improving the prediction of software development and effort.

In the following Section 2, GGGP is introduced very briefly. The application of GGGP in evolution of software development effort estimation programs is discussed in Section 3. This includes data preparation, GP details and the results obtained. The results is analyzed in Section 4. In Section 5, the conclusion is given.

## 2 Grammar Guided Genetic Programming

One of limitation of canonical Genetic Programming (GP) [4, 5] is its requirement of closure. It means that function set should be well defined for any combination of arguments. Closure makes many program structures difficult to express. To overcome it, an early attempt is Strongly Typed Genetic Programming [6]. After this, several attempts have been tried to impose syntactical constraints, by means of grammar, in GP [2, 8, 10, 7]. In fact, in some of these research, grammar is not only a way to represent syntactical constraints, but also a way to incorporate background knowledge to guide GP process. In this paper, Grammar Guided GP is used to model the software development effort due to the salient advantages of GGGP. Grammar in GGGP bias the GP individual structure to find the optimal or near-optimal result more effectively and efficiently. Compared with canonical GP, GGGP have the following advantages:

- With the grammar constraint, the closure requirement in canonical GP is removed so that more expressive program structure can be evolved.
- Grammar in GGGP provide a natural and formalized way to represent background knowledge. With background knowledge, the search space is reduced dramatically.
- Problem related building blocks, a kind of *a priori* knowledge, can be represent in grammar which would improve more search efficiency.

- During the GP search process, the grammar itself can be evolved which lead to incremental learning. Grammar can also be modified manually during learning process to make it biased towards expected result.

# 3 Evolution of Software Development Effort Estimation Programs

## 3.1 Data preparation

The data of 423 software development projects is collected. For each of project, there are 32 attributes p1∼p32, for example, software size, team size, defects, platform, development language, team ability, etc. Among them, p1∼p4 are numeric variables, p5∼p13 are unordered categorical variables, p14∼p23 are ordered categorical variables and p24∼p32 are boolean variables. These 423 records are randomly divided to training and testing data sets, each of which contains the same number of records. In this research, we use GGGP to fit a model with the aim of determining appropriate metric sets containing the most relevant attributes and improving the prediction of software development effort.

## 3.2 Target language

One of important preparation step for GGGP is to identify a suitable target language in which to evolve programs. On one hand, the language should be expressive enough to cover potential solution space. On the other hand, too general language may ruin the efficiency of execution. Too general language deviates from one of the most important purpose of employing grammar: constraining search space. This trade-off need to be carefully considered.

Two languages are used for evolving software development effort estimation program. The context free grammar [3, 9] for these languages are in Figure 1. Each of the grammars describes a language, which try to regress a mathematic model with a numeric return value as predication for software development effort. Grammar 1 is for generating common mathematic expressions with all independent variables p1∼p32 and operators, such as +,-,*,/,exp, etc. Note in this grammar, the production for constant:
CONSTVAR = if BOOL CONSTVAR CONSTVAR
is deliberately designed so that the constants could be made depend on the other variables. Grammar 2 is more general. More complex if-then is allowed. It is a superset of Grammar 1. Most of these two grammars are self-explanatory. ORDERED_VALUE is constant for corresponding ordered categorical variables.For example, for ordered categorical variable user_groups, there are three different values: one, one_to_five, over_five. Hence, for user_group, its ORDERED_VALUE is one of these three values. UNORDERED_VALUE_SET is constant value set for corresponding unordered categorical variables. For example, for unordered categorical variable dbms, its possible values set is { access, adabas, db2, ims, ingres, oracle, rdb, others}, which contains eight elements. Therefore, for variable dbms, its corresponding UNORDERED_VALUE_SET is any subset of this set. <ephemeral const> is randomly generated floating-point constant between 0 and 10. GP parameters is summarized in Table 1.

Mean square error (MSE) is used as fitness function as following:

$$MSE = (\sum_{i=1}^{N}(estimated\_effort - actual\_effort)^2)/N$$

where N is total number of fitness case 423, estimated_effort is the prediction from the model while actual_effort is actual value.

## 3.3 Results

Using grammar 1, five GP runs are generated. On the same training and testing data sets, five linear and log regression are also conducted. With the aid of visualization techniques, it is conjectured that, linear relation maybe exits between log(software_development_effort) and log(software_size). Therefore, log regression, *i.e.* regression of linear relation between log(software_development_effort) and log(software_size), is used to verify this conjecture. The MSE is recorded for the above three kinds of regression as a criterion for comparison. The mean and standard deviation of MSE are summarized in table 2.

According to table 2, our result is quite promising. From the perspective of modeling, GP is significantly better than linear and log regression, in terms of mean square error. Among these three regressions, the performance of log regression and linear regression is comparable. The training error of log regression is larger than linear regression's while it generalizes marginally better on testing data set than linear regression. The testing error of GP is considerably smaller than the other two methods' although its testing error is larger than its training error.

domain expert interpret some of the results.

# 4 Discussion

With the successful application of GP in this software engineering program, we turned to Grammar 2, which is more general than Grammar 1. Not only common mathematic expression but also complex if-then clause can be generated in Grammar 2. In Grammar 1, if-then performs a relational test in order to evolve proper constants for the whole model while, in Grammar 2, the statement part in if-then can be highly complicated. However, no better result is found. After study the result of Grammar 1, we found that this outcome is not surprising. In most of best-of-run individuals discovered by Grammar 1, if-then clause is missing. This means that even in evolution of constants, those non-numerical attribute only play a trivial role as non-numeric variable can only appear in condition part of if-then. Therefore, it is not expected those non-numerical attribute would impact the evolved program of Grammar 2 very much.

But why non-numerical attributes has little influence on estimated software development effort? Our conjecture is that 1) the non-numerical attributes are not closely related to the software development effort or 2) the combination of these non-numerical attributes are too complex to be discovered by GP. We incline to the latter one. Intuitively, some non-numerical attributes, such as, whether fourth generation language is used, whether object oriented technique is employed, should influence the software productivity perceivably. How to discover and model these factors needs to be investigated in further research. Other further research issues, that we will explore in the future, includes:

- As we can see in our experiment, little background knowledge is incorporated into both of the grammars. No doubt, cooperated with domain expert to incorporate background knowledge to bias search space would lead to more accurate prediction.

- Grammar itself can be automatically refined during the GP process. It is possible that novel pattern may be discovery through the analysis of the automatically refined grammar.

- Another research issue is how to deal with missing values. There are a large number of missing values in our date set. This should often be the case for real world application. Several methods handling this problem are investigated in other research fields, such as decision tree. It would be meaningful to borrow them to GP for real world application.

Grammar 1:
EXP = PREOP EXP | EXP OP EXP | NUMERIC | CONSTVAR
BOOL = BOOL and BOOL | BOOL or BOOL | not BOOL | EXP CP EXP | ORDERED CP ORDERED_VALUE | UNORDERED in UNORDERED_VALUE_SET | BOOLVAR
PREOP = exp | sqrt | log
OP = + | - | * | / | power
CP = < | > | =
NUMERIC = p1 |$\cdots$|p4
ORDERED = p5 |$\cdots$| p13
UNORDERED = p14|$\cdots$ |p23
BOOLVAR = p 24 |$\cdots$| p32
CONSTVAR = if BOOL CONSTVAR CONSTVAR | CONSTVAR OP CONSTVAR | <ephemeral const>

Grammar 2: (More general)
EXP = PREOP EXP | EXP OP EXP | if BOOL EXP EXP | NUMERIC | CONST
CONST = <ephemeral const>
All the other production rules same as Grammar 1

Figure 1: Grammar of the GP languages

# 5 Conclusions

As we have mentioned, accurate estimation software development effort is important for software industry.(any figure for potential economic value of accuracy of estimation?) The research in the paper successfully used GP for evolving programs to this problem. To our knowledge, this is the first attempt at using evolutionary computation to software development effort estimation problem. In this experiment, GGGP because of its flexibility and the ability of incorporating background knowledge, also shows great potentials in being applied in other software engineering modeling problem.

# References


[1] M. Boraso, C. Montangero, and H. Sedehi. Software cost estimation: an experimental study of model performances. Technical Report TR-96-22, DEPARTIMENTO DI INFORMATATICA, UNIVERSITA DI PISA, Italy, 1996.

[2] Frederic Gruau. On using syntactic constraints with genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 19, pages 377–394. MIT Press, Cambridge, MA, USA, 1996.

| Parameter | Value |
|---|---|
| Terminals, non-terminals | (see Fig. 1) |
| Fitness function | Mean square error |
| Generation type | Steady state |
| Selection scheme | Tournament, 3 |
| Population | 1000 |
| Max. generations | 200 |
| Runs | 5 |
| Init. population tree size | Ramped half&half |
| Min/max depth initial popn | 6/9 |
| Probability crossover | 0.9 |
| Probability mutation | 0.1 |
| Probability internal corssover | 0.9 |
| Probability terminal mutaion | 0.75 |

Table 1: GP parameters

| | Linear | | Log | | GP | |
|---|---|---|---|---|---|---|
| | Mean | STD | Mean | STD | Mean | STD |
| Train | 17.9 | 2.7 | 23.0 | 2.8 | 2.90 | 0.47 |
| Test | 15.9 | 2.5 | 15.6 | 2.8 | 5.40 | 0.61 |

Table 2: Mean and standard deviation of mean square error of linear, log and GP regression. (Scaled by $10^{-6}$)


[3] D. A. Gustafson, W. A. Barrett, R. M. Bates, and J. D. Couch. *Compile construction: Theory and Practice*. Science Research Assoc, Inc., 1986.

[4] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[5] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

[6] David J. Montana. Strongly typed genetic programming. BBN Technical Report #7866, Bolt Beranek and Newman, Inc., 10 Moulton Street, Cambridge, MA 02138, USA, 7 May 1993.

[7] Michael O'Neill and Conor Ryan. Grammatical evolution. *IEEE Transaction on Evolutionary Computation*, 5(4):349–358, 2001.

[8] P. A. Whigham. Grammatically-based genetic programming. In Justinian P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, Tahoe City, California, USA, 9 July 1995.

[9] P.A. Whigham. *Grammatical Bias for Evolutionary Learning*. PhD thesis, School of Computer Science, University College, Univ. of New South Wales, Australia, 1996.

[10] Man Leung Wong and Kwong Sak Leung. Combining genetic programming and inductive logic programming using logic grammars. In *1995 IEEE Conference on Evolutionary Computation*, volume 2, pages 733–736, Perth, Australia, 29 November - 1 December 1995. IEEE Press.