A Grammatical Evolution Approach for Software Effort Estimation

Rodrigo C. Barros
Institute of Mathematical Sciences and Computing
University of São Paulo
São Carlos, SP, Brazil
rcbarros@icmc.usp.br

Márcio P. Basgalupp
Institute of Science and Technology
Federal University of SP
S. J. dos Campos, SP, Brazil
basgalupp@unifesp.br

Ricardo Cerri
Institute of Mathematical Sciences and Computing
University of São Paulo
São Carlos, SP, Brazil
cerri@icmc.usp.br

Tiago S. da Silva
Institute of Mathematical Sciences and Computing
University of São Paulo
São Carlos, SP, Brazil
tiago.silva@icmc.usp.br

André C. P. L. F. de Carvalho
Institute of Mathematical Sciences and Computing
University of São Paulo
São Carlos, SP, Brazil
andre@icmc.usp.br

ABSTRACT

Software effort estimation is an important task within software engineering. It is widely used for planning and monitoring software project development as a means to deliver the product on time and within budget. Several approaches for generating predictive models from collected metrics have been proposed throughout the years. Machine learning algorithms, in particular, have been widely-employed to this task, bearing in mind their capability of providing accurate predictive models for the analysis of project stakeholders. In this paper, we propose a grammatical evolution approach for software metrics estimation. Our novel algorithm, namely SEEGE, is empirically evaluated on public project data sets, and we compare its performance with state-of-the-art machine learning algorithms such as support vector machines for regression and artificial neural networks, and also to popular linear regression. Results show that SEEGE outperforms the other algorithms considering three different evaluation measures, clearly indicating its effectiveness for the effort estimation task.

Categories and Subject Descriptors

I.2.m.c [**Artificial Intelligence**]: Miscellaneous—*Evolutionary computing and genetic algorithms*; D.2.8 [**Software Engineering**]: Metrics/Measurement

Keywords

Grammatical Evolution, Software Effort Estimation, Software Metrics, Genetic Programming, Evolutionary Algorithms

1. INTRODUCTION

Properly estimating the effort required to develop (or maintain) software is of great importance. Effort estimation supports project management in scheduling resources and evaluating risk factors. Either over or underestimation of software effort can result in troublesome scenarios. For instance, underestimation leads to schedule and budget overruns, which are recurrent causes of project cancellation. Overestimation, on the other hand, leads to loss of competitiveness and waste of resources, and it may ultimately prevent the company of securing a contract due to the excessive calculated costs.

The need for accurate effort estimates has motivated the investigation of many different strategies for generating predictive models in an efficient and effective fashion. An increasing number of *machine learning* approaches have been proposed and/or applied to predict software development (or maintenance) effort. Examples include case-based reasoning [10,17,20], artificial neural networks [5,6,19], decision trees [2, 12, 15], Bayesian networks [26, 29], support vector machines for regression [11, 27], genetic programming [23, 31], and evolutionary algorithms in general [3, 4].

The idea of exploiting evolutionary algorithms (EAs) for software effort estimation is based on the assumption that effort estimation can be formulated as an optimization problem [31], in which the EA strives for the predictive model that globally optimizes a given evaluation criterion — usually a measure of predictive performance. Given that the most popular technique for estimating software effort[1] is linear regression [22], it is fair to say that genetic programming (GP) is an obvious choice of EA for estimating effort, bear-

[1]The work of Jorgensen and Shepperd [22] actually refers to software cost estimation, though it is well-known that cost and effort are directly related, and that effort is usually preferred in academic studies due to confidentiality reasons.

ing in mind its straightforward application to evolve mathematical functions. Both traditional GP [8,14] and grammar-based (or grammar-guided) GP [33] have been successfully applied to software effort estimation, with promising results being reported.

In this work, we further exploit the use of EAs for software effort estimation. We propose a new *grammatical evolution* approach for software effort estimation called SEEGE (Software Effort Estimation with Grammatical Evolution). To the best of our knowledge, this is the first work to propose a grammatical evolution algorithm for software effort estimation. We believe grammatical evolution combines the advantages of grammar-based GP — flexibility and ability of incorporating prior knowledge — with the advantages of genetic algorithms — simplicity of breeding operations over vectors. In addition, grammatical evolution allows for neutral mutations, the so-called degenerative genetic code [28].

This paper is organized as follows. Section 2 presents work related to our approach. Section 3 details SEEGE, our novel grammatical evolution algorithm for software effort estimation. Section 4 describes the methodology we followed to set up the experiments, whereas Section 5 depicts the performance analysis of SEEGE in public effort data sets. Finally, we end this paper with our conclusions and future work directions in Section 6.

2. RELATED WORK

Several studies in the literature have exploited traditional GP for software effort estimation [1,8,9,16,23,31]. Burgess and Lefley [8] critically evaluate the potential of GP for effort estimation. They compare GP with previously published approaches, in terms of accuracy and ease of use. They show evidence that GP can offer significant improvements in accuracy, though they comment on the numerous parameters that need to be set.

Lefley and Shepperd [23] evaluate various statistical and machine learning techniques, including a GP tool, in the context of effort estimation. The main hypothesis tested was whether GP could significantly produce a better estimate of effort taking into account a public data set. They conclude that sophisticated estimation techniques such as artificial neural networks and GP provide better fitting models, though they require extra effort in design, whereas a simple linear regression approach can still provide useful estimates.

Tsakonas and Dounias [1] also employ GP in order to produce mathematical expressions that are highly accurate and can be used for estimating the development effort by revealing relationships between the project's features and the required work. According to the authors, their system was proved capable to produce results that not only carry higher regression accuracy as compared to those found in literature, but also are interpretable mathematical formulas, easy to be used by project managers.

Ferrucci et al. [16] carry out an empirical analysis to provide an insight on the use of GP for effort estimation, and in particular to analyze how the estimation accuracy of GP is affected by the use of different fitness functions. They experiment with different fitness functions based on widely recognized indicators used to evaluate the accuracy of the estimates and combinations of them.

Chavoya et al. [9] make use of GP with the goal of estimating the effort required in the development of short-scale projects. They compare it with the results obtained from a neural network and linear regression. Accuracy results from the model obtained with GP suggest that it could be used to estimate software development effort of short-scale projects when these projects are developed in a disciplined manner within a development-controlled environment.

Sarro et al. [31] also aim at getting an insight on the effects of using different measures as fitness function in a GP for effort estimation. In their study, they analyzed the performance of five different evaluation criteria — MMRE, Pred(.25), MdMRE, MEMRE and MdEMRE.

Only the work of Shan et al. [33] attempts on employing a grammar-based genetic programming approach (GGGP) for effort estimation. They conclude that GGGP performs better than linear regression in all aspects.

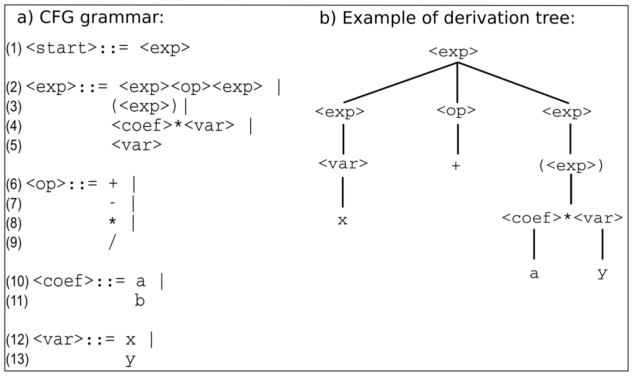
To the best of our knowledge, this is the first work to propose a grammatical evolution (GE) approach for software effort estimation. GE is a reasonably recent evolutionary algorithm that presents some interesting advantages over GP and other EAs: it combines the flexibility of grammar-based approaches with the simplicity of evolving linear strings. Hence, it inherits the advantages of GGGP and genetic algorithms, while taking advantage of neutral mutations [28]. We present our approach in detail in the next section.

3. SOFTWARE EFFORT ESTIMATION WITH GRAMMATICAL EVOLUTION

Software Effort Estimation with Grammatical Evolution (SEEGE) is a grammar-based Evolutionary Algorithm (EA) aimed at estimating software effort. As it is grammar-based, it differs from traditional EAs in the use of a grammar G in the evolutionary process. The grammar G defines a language L whose terms are the functions and terminals. A grammar can be represented by a four-tuple $G = \{N, T, P, S\}$, where N is the set of non-terminals, T is the terminal set, P is the set of production rules and S is the start symbol. The elements in P are responsible for deriving the language L by combining elements from sets N and T. The well-known Chomsky hierarchy classifies the types of grammars in four levels, from 0 to 3. Level 2, context-free grammars (CFG), are the most commonly used in grammar-based EAs [7,25,34].

Figure 1 presents a CFG grammar, written in the well-known Backus-Naur form (BNF), and an example of a derivation tree. For deriving the tree in 1(b), the following production rules were executed: `<start>` $\Rightarrow^1$ `<exp>` $\Rightarrow^2$ `<exp><op><exp>` $\Rightarrow^5$ `<var><op><exp>` $\Rightarrow^{12}$ `x<op><exp>` $\Rightarrow^6$ `x+<exp>` $\Rightarrow^3$ `x+(<expr>)` $\Rightarrow^4$ `x+(<coef>*<var>)` $\Rightarrow^{10}$ `x+(a*<var>)` $\Rightarrow^{13}$ `x+(a*y)`.

In grammar-based GP, the individual that undergoes evolution is the derivation tree presented in Figure 1(b). Therefore, genotype and phenotype are both represented by the derivation tree. Grammatical Evolution (GE) differs from grammar-based GP in the sense that genotype and phenotype have different encoding structures. The genotype is represented as a linear string of bits and/or integers of variable size. The phenotype, which is the structure that defines fitness, is represented by the derivation tree. Hence, a mapping from genotype to phenotype is required in a GE algorithm. The mapping process to decode the string chromosomes into derivation trees is called *genotype-phenotype mapping* (GPM). Some allegedly benefits of this approach are the unconstrained search of the genotype while ensur-


a) CFG grammar:

```
(1) <start>::= <exp>

(2) <exp>::= <exp><op><exp> |
(3)          (<exp>)|
(4)          <coef>*<var> |
(5)          <var>

(6) <op>::= + |
(7)         - |
(8)         * |
(9)         /

(10) <coef>::= a |
(11)           b

(12) <var>::= x |
(13)          y
```

Figure 1: (a) CFG grammar for deriving expressions. (b) An example of tree derived from the grammar in (a). The derived expression was $x + (a \times y)$.

ing phenotype validity, and enhancing genetic diversity by allowing mutations which are neutral with respect to the phenotype (various genotypes can represent the same phenotype). Figure 2 depicts the *grammatical evolution* scheme.

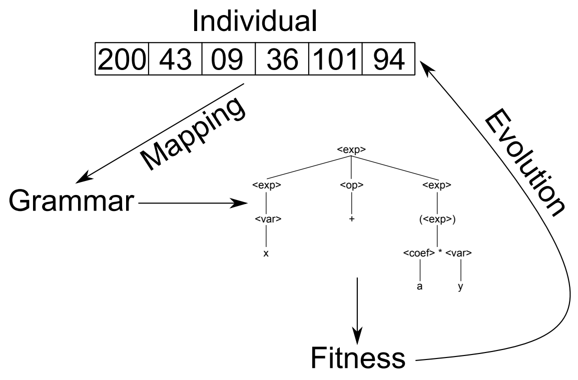


Figure 2: Grammatical evolution scheme.

3.1 Grammar and Genotype-Phenotype Mapping

SEEGE employs the GPM process proposed by Ryan et al. [30], in which codons are used to decide which choice to take when a non-terminal has more than one outcome. In molecular biology, a codon is a triplet of nucleic acids which encodes one amino acid. Its analogue in our case is a bit sequence of b bits, each encoding a symbol of the program. The genotype, thus, is a binary string consisting of n codons. Just like in a nucleotide sequence, where different codons can encode one amino acid, it is desirable that several codons get mapped into the same symbol (many-to-one mapping) in order to allow different genotypes to represent the same phenotype. Each codon is coded as a 8-bit sequence, and a chromosome consists of a variable number of codons. A 8-bit codon represents an integer value, which is used in a GPM function to choose the appropriate production rule from the grammar.

The GPM function is given by: (codon integer value) `MOD` (number of rules for the current non-terminal), where `MOD` is the modulus operation (it returns the remainder of the division of one number by another).

As an example of the GE approach, suppose we want to map the individual depicted in Figure 2 $[200, 43, 09, 36, 101, 94]$ (already decoded from binary to integer) into a derivation tree, and consider that we are using the grammar shown in Figure 1. The GPM function in GE starts reading the string from left to right. The first integer is 200, and the grammar starts with the non-terminal `<exp>`. This non-terminal has 4 production rules associated to it (each element separate by "|"). The computation of the `MOD` operation — `MOD`(200, 4) — results in index 0. Considering that each non-terminal has n production rules associated to it, and that they are indexed from 0 to $n-1$, the selected production rule derives the non-terminal `<exp><op><exp>`. Next, we derive the leftmost non-terminal <exp> by mapping the next integer from the sequence (43) through the `MOD` function, resulting in index 3. The selected production rule derives the non-terminal `<var>`. Following this procedure, integer 9 results in index 1, which points to terminal x. Integer 36 is used to select the production rule in the `<op>` non-terminal. It results in index 0, selecting terminal $+$. Next, integer 101 applied to the non-terminal `<exp>` results in index 1, selecting non-terminal (`<exp>`). Then, integer 94 results in index 2, selecting non-terminal `<coef>`$*$`<var>`. At this moment, an important function from GE called wrapping is invoked, because the string has run out of codons before the derivation tree is complete. Wrapping works by returning the pointer to the beginning of the codons string, so codons can be re-utilized. In this example, 200 is once again evaluated, this time regarding non-terminal `<coef>`. It results in index 0, which means that terminal a is selected. The final non-terminal to be derived is `<var>`. The next integer to be read is 43, which results in index 1 and the posterior selection of terminal y. The derivation tree is completed, even though the second "scanning" over the codons has ended in the middle of the string.

To estimate software effort, we implemented two different grammars in SEEGE (see Figure 3). Grammar (a) is a simplified version of grammar (b). In both grammars, `<coef>` is a number (ephemeral random constant) that has its value randomly drawn from a uniform distribution — $U(0, 1)$ generates real numbers between 0 and 1, whereas $U(0, 100)$ generates integer numbers between 0 and 100 — and `<var>` is a predictive attribute of the training data set (chosen randomly from the set of predictive attributes). Note that grammar (b) allows if-then-else rules to be derived, enhancing the search-space of the resulting mathematical function.

3.2 Evolution

In the beginning of the evolutionary process, the individuals of the population are randomly initialized. They have variable length, with a minimum of five codons each, and a chance of 85% that new codons will be added to the individual. Hence, the process of adding codons finishes when the 85% probability is not fulfilled. Recall that each codon is comprised of 1 byte (8 bits), which is randomly generated.

To evolve the current generation of individuals, the following mutually-exclusive genetic operations can be performed: crossover, mutation, and duplication. Crossover has a chance of 90% of being performed, and both duplication and mutation have a chance of 5% of being applied.

```
<start>::= <exp>

<exp>::= <exp><op><exp> | <coef><op><var> |
         <op2><exp> | <coef> | <var>

<op>::=  + | - | * | /

<op2>::= sin | cos | log

<coef>::= U(0,1)

<var>::=  Predictive Attribute
```

(a) A simplified context-free grammar for effort estimation.

```
<start>::= <exp>

<exp>::= <exp><op><exp> | <coef><op><var> |
         <op2><exp> | <coef> | <var> |
         if<bool>then<exp>else<exp>

<op>::=  + | - | * | /

<op2>::= sin | cos | log

<bool>::= <exp><op3><exp>

<op3>::= > | < | >= | <= | = | !=

<coef>::= U(0,1) | U(0,100)

<var>::=  Predictive Attribute
```

(b) A more detailed context-free grammar for effort estimation.

Figure 3: Grammars used in SEEGE.

These operations are executed until all individuals of the new population are generated.

For crossover to be performed, two individuals are chosen via tournament selection (default tournament size of 7). After the individuals are selected, they are truncated, which means that the codons that are not used to generate the derivation tree are removed. The truncated individuals then take part in a standard one-point crossover operation, generating two children. In the duplication process, one individual is also selected using tournament selection, and it is also truncated to eliminate codons that are not used to generate a derivation tree. Then, two codons of the individual are randomly selected, and all codons located between these two selected codons are copied to the end of the individual. Regarding mutation, it requires one individual to be selected via tournament selection. This individual is then traversed codon by codon, where each codon has a 10% probability of having its value replaced by a randomly-generated 8-bit value. The three operators are illustrated in Figure 4.

Most papers that propose EAs for software effort estimation employ either the mean square error (MSE) or the mean magnitude of the relative error (MMRE) as fitness functions [31]. In the current version of SEEGE, we follow the literature and also employ the MSE error measure:

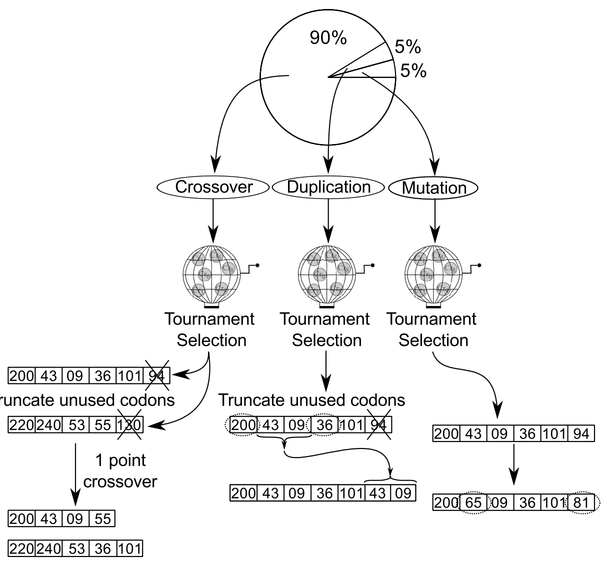


Figure 4: SEEGE evolutionary operators.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad (1)$$

where y_i is the actual effort, $\hat{y}_i$ is the predicted effort value, and N is the number of objects in the data set.

4. EXPERIMENTAL PLAN

In this section, we present the methodology we employ for evaluating the performance of SEEGE in software effort data sets. In Section 4.1, we describe the data sets that we employ during the experiments, whereas in Section 4.2 we detail the algorithms that are used as baselines for SEEGE. Finally, Section 4.3 depicts the statistical tests we make use for providing some reassurance about the validity of the comparison results.

4.1 Data Sets

The empirical analysis presented in this paper is based on 10 different data sets from the PRedictOr Models In Software Engineering (PROMISE) Repository [32]. They are: albrecht, china, coc81, coc81lnh, cocomo-sdr, cocomo-nasa, desharnais, kemerer, maxwell, and nasa93. Table 1 provides some details about these data sets.

4.2 Algorithms, Parameters, and Evaluation

In order to assess the relative performance of SEEGE, we compare it to state-of-the-art machine learning algorithms, namely support vector machines for regression (SVM-Regression) and artificial neural networks (ANN). In addition, we compare it to traditional least-square linear regression, which is frequently employed for software effort estimation [22].

The implementations of the baseline methods are those found within the Weka Machine Learning Toolkit [18], namely: SMOreg (SVM-Regression), MultilayerPerceptron (ANN), and LinearRegression (Linear Regression).

SEEGE was implemented within the ECJ framework of evolutionary computation [24]. Bearing in mind that we

Table 1: **Summary of the 10 effort data sets.**

Data Set	#projects	#features	min Effort	max Effort	μ Effort	σ Effort	Description
albrecht	24	7	0.5	105.2	21.88	28.42	Data from 24 applications developed by IBM.
china	499	17	26	54620	3921.05	6480.86	No description available.
coc81	63	20	5.9	11400	683.53	1821.51	Data from COCOMO data analysis of 63 projects.
coc81lnh	63	71	5.9	11400	683.32	1821.58	Data from COCOMO data analysis of 63 projects with effort multipliers expressed as “low,medium,hight”.
cocomo-sdr	12	134	1	22	5.73	6.84	Data from a Turkish Software Industry.
cocomonasa-v1	60	56	8.4	3240	406.41	656.97	Data from 60 NASA projects from 1980s and 1990s.
desharnais	81	11	546	23940	5046.31	4418.77	Data from a Canadian software house in the late 1980s.
kemerer	15	6	23.2	1107.31	219.247	263.06	Data from large business applications.
maxwell	62	27	583	63694	8223.21	10499.90	Data from one of the biggest commercial banks in Finland.
nasa93	93	124	8.4	8211	1135.93	1135.93	Data from 93 NASA projects from 1971 to 1987.

have proposed two grammars for SEEGE, we name its two versions as *SEEGE-a* and *SEEGE-b*, where SEEGE-a is the implementation of SEEGE with the grammar presented in Figure 3(a), and SEEGE-b is the implementation of SEEGE with the grammar presented in Figure 3(b).

The baseline methods were executed with their default parameter values, which are usually a combination of values that work well across a wide variety of data sets. In order to keep the comparison as fair as possible, we did not attempt to optimize the parameters of SEEGE. The list of configurable parameters of SEEGE is presented in Table 2.

Since SEEGE is a non-deterministic method, we execute it 10 different times varying the random seed of the stochastic operations. For evaluating the performance of both SEEGE and baseline methods, we employ a 10-fold cross-validation procedure, in which we collect the following evaluation measures: root mean square error (RMSE), which is simply the root of MSE; mean absolute error (MAE), and mean magnitude of the relative error (MMRE):

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |\hat{y}_i - y_i| \tag{2}$$

$$MMRE = \frac{1}{N}\sum_{i=1}^{N} \frac{|\hat{y}_i - y_i|}{y_i} \tag{3}$$

Table 2: **Configurable SEEGE parameters.**

Parameter	Value
Initialization Probability	85%
Number of Individuals	1000
Minimum Individual Size	5
Number of Generations	100
Crossover Probability	90%
Duplication Probability	5%
Mutation Probability	5%
Mutation per Codon Probability	10%
Tournament Size	7
Elite	10

4.3 Statistical Analysis

To evaluate the statistical significance of the experimental results, we present the results of statistical tests by following the approach proposed by Demšar [13]. In brief, this approach seeks to compare multiple algorithms on multiple data sets, and it is based on the use of the Friedman test with a corresponding post-hoc test. The Friedman test is a non-parametric counterpart of ANOVA, as follows. Let R_i^j be the rank of the j^{th} of k algorithms on the i^{th} of N data sets. The Friedman test compares the average ranks of algorithms, $R_j = \frac{1}{N}\sum_i R_i^j$. The Friedman statistic, given by:

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right] \tag{4}$$

is distributed according to χ_F^2 with $k-1$ degrees of freedom, when N and k are large enough.

Iman and Davenport [21] showed that Friedman's χ_F^2 is undesirably conservative and derived an adjusted statistic:

$$F_f = \frac{(N-1)\times\chi_F^2}{N\times(k-1)-\chi_F^2} \tag{5}$$

which is distributed according to the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom.

If the null hypothesis of similar performances is rejected, we proceed with the Nemenyi post-hoc test for pairwise comparisons. The performance of two classifiers is significantly different if their corresponding average ranks differ by at least the critical difference

$$CD = q_\alpha\sqrt{\frac{k(k+1)}{6N}} \tag{6}$$

where critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$.

5. RESULTS AND DISCUSSION

Table 3 shows the RMSE values for Linear Regression, ANN, SVM-Regression, and both SEEGE versions (SEEGE-a and SEEGE-b). It illustrates the average RMSE over the 10-fold cross-validation runs and the standard deviation of the RMSE obtained in those runs (best absolute values in bold). It is possible to see that SEEGE generates the model with the lowest error values for all data sets. SEEGE-a provides the best model for seven data sets, whereas SEEGE-b achieves it in the remaining three.

To evaluate the statistical significance of the RMSE results, we calculated the average Friedman rank for Linear

Table 3: RMSE values of Linear Regression, ANN, SVM-Regression, and SEEGE.

Data Set	Linear Regression	ANN	SVM-Regression	SEEGE-a	SEEGE-b
albrecht	12.23 ± 11.20	11.50 ± 8.92	11.73 ± 10.11	**2.41 ± 4.81**	2.87 ± 4.92
china	775.06 ± 502.78	991.86 ± 599.01	720.92 ± 581.80	**544.34 ± 133.71**	597.55 ± 148.91
coc81	1359.14 ± 1184.96	815.88 ± 1143.77	758.13 ± 1325.62	**375.58 ± 173.83**	386.39 ± 227.29
coc81lnh	1081.15 ± 1530.09	1143.03 ± 1494.28	2422.65 ± 1169.92	453.40 ± 341.74	**428.35 ± 197.11**
cocomo-sdr	7.66 ± 6.05	9.93 ± 4.94	8.75 ± 6.97	0.66 ± 2.01	**0.57 ± 1.68**
cocomonasa-v1	545.96 ± 421.03	303.35 ± 334.83	481.73 ± 468.43	**81.45 ± 62.95**	112.90 ± 80.27
desharnais	2921.27 ± 1199.51	6045.28 ± 3728.47	2828.75 ± 1235.14	**1551.03 ± 412.06**	1755.36 ± 408.50
kemerer	183.05 ± 162.60	178.52 ± 203.75	91.65 ± 156.25	25.58 ± 55.70	**24.30 ± 48.96**
maxwell	6160.94 ± 3709.70	5952.22 ± 2604.27	5758.57 ± 3117.25	**2391.53 ± 1119.28**	2746.10 ± 1065.49
nasa93	819.01 ± 864.54	1336.17 ± 990.01	1384.26 ± 921.77	**347.55 ± 138.35**	444.10 ± 125.45
Average Rank	4.2	4.1	3.7	1.3	1.7

Table 4: MAE values of Linear Regression, ANN, SVM-Regression, and SEEGE.

Data Set	Linear Regression	ANN	SVM-Regression	SEEGE-a	SEEGE-b
albrecht	10.69 ± 9.20	10.30 ± 7.86	10.50 ± 8.26	**2.09 ± 4.15**	2.51 ± 4.42
china	351.31 ± 109.85	474.27 ± 166.24	270.54 ± 119.03	**293.31 ± 46.25**	307.68 ± 53.50
coc81	1078.00 ± 804.84	553.69 ± 753.86	523.14 ± 863.54	**243.87 ± 96.78**	270.83 ± 149.07
coc81lnh	892.00 ± 1025.29	832.00 ± 1019.81	1987.11 ± 770.11	**292.54 ± 195.10**	310.54 ± 143.92
cocomo-sdr	7.48 ± 6.11	9.74 ± 5.00	8.61 ± 6.96	0.66 ± 2.00	**0.57 ± 1.67**
cocomonasa-v1	437.08 ± 297.18	234.79 ± 259.85	385.62 ± 411.86	**61.26 ± 42.91**	85.98 ± 58.11
desharnais	2277.29 ± 908.01	4387.11 ± 2717.91	2235.54 ± 1035.25	**1209.47 ± 279.83**	1420.81 ± 329.92
kemerer	167.69 ± 133.60	168.36 ± 193.84	78.61 ± 128.21	23.16 ± 50.92	**21.29 ± 41.78**
maxwell	4642.71 ± 2545.82	4250.80 ± 1727.51	4190.51 ± 1875.96	**1858.02 ± 768.23**	2177.90 ± 805.25
nasa93	560.13 ± 511.89	1020.66 ± 694.63	1054.89 ± 722.09	**252.81 ± 90.40**	319.10 ± 91.19
Average Rank	4.2	4.1	3.5	1.3	1.9

Regression, ANN, SVM-Regression, SEEGE-a, and SEEGE-b: 4.2, 4.1, 3.7, 1.3 and 1.7, respectively. The average rank clearly suggests that both versions of SEEGE outperform the baseline methods regarding RMSE. The calculation of Iman's F statistic resulted in $F_f = 30.47$. Critical value of $F(k-1,(k-1)(n-1)) = F(4,36)$ for $\alpha = 0.05$ is 2.63. Since $F_f > F_{0.05}(4,36)$ $(30.47 > 2.63)$, the null-hypothesis is rejected. We proceed with a post-hoc Nemenyi test to find which method provides better results in a pairwise fashion. The critical difference $CD = 1.93$. The differences between the average rank of SEEGE-a and the rank of the baseline methods – Linear Regression, ANN, and SVM-Regression – are 2.90, 2.8, and 2.4, respectively. The difference between SEEGE-b and the baseline methods are: 2.5, 2.4, and 2.0, respectively. Given that every difference is greater than CD, we can argue that the performance of SEEGE is significantly better than Linear Regression, ANN, and SVM-Regression with statistical significance, regarding RMSE.

Table 4 shows the MAE values for Linear Regression, ANN, SVM-Regression, SEEGE-a, and SEEGE-b. Results show once again that SEEGE generates the best models for all data sets. In eight data sets SEEGE-a generated the best model, whereas SEEGE-b did the same for two data sets. Regarding the statistical analysis, the computed value of $F_f = 21$. Since $F_f > F_{0.05}(4,36)$ $(21 > 2.63)$, the null-hypothesis is rejected. The differences between the average rank of SEEGE-a and the baseline methods – Linear Regression, ANN, and SVM-Regression – are 2.9, 2.8 and 2.2, respectively. The differences between the average rank of SEEGE-b and the baseline methods are 2.3, 2.2, and 1.6. Note that only the difference between SEEGE-b and SVM-Regression (1.6) is not greater than CD (1.93).

Finally, Table 5 shows the MMRE values for all methods. Results show that SEEGE generates the best model in nine out of ten data sets: SEEGE-a is the best method in seven data sets and SEEGE-b in two. SVM-Regression generates the best model for the china data set, whereas the remaining methods do not provide the best model for any data set.

The average rank for Linear Regression, ANN, SVM-Regression, SEEGE-a, and SEEGE-b are: 4.0, 4.2, 3.6, 1.4, and 1.8, respectively. It clearly suggests that SEEGE is the best performing method regarding MMRE. The calculation of Iman's F statistic results in $F_f = 19.13$. Since $F_f > F_{0.05}(4,36)$ $(19.13 > 2.63)$, the null-hypothesis is rejected.

The differences between the average rank of SEEGE-a and the baseline methods (Linear Regression, ANN, and SVM-Regression) are 2.6, 2.8, and 2.2, respectively. Also, the differences between SEEGE-b and the baselines are 2.2, 2.4, and 1.8, respectively. Once again, with the exception of the difference between SEEGE-b and SVM-Regression, all the others are greater than CD, which indicates that the performance of SEEEGE is significantly better than Linear Regression, ANN, and SVM-Regression regarding MMRE with a significance level of $\alpha = 0.05$.

The statistical analysis previously presented clearly shows that the models generated by SEEGE outperform well-known methods such as Linear Regression, ANN, and SVM-Regression regarding their predictive performance in terms of three distinct evaluation measures, namely: RMSE, MAE, and MMRE.

It is interesting to notice that SEEGE-a often outperforms SEEGE-b, though the grammar employed in SEEGE-b offers many more choices for building mathematical models than SEEGE-a (*i.e.*, it allows `if-then-else` rules and integer constants). We believe SEEGE-b may have been slightly affected by *overfitting* — when the model over-learns the training data and fails to properly generalize to unseen data. Nevertheless, there were data sets in which SEEGE-b was indeed the best option, which means that some data sets require more complex functions than others. This was clearly the case of the cocomo-sdr and kemerer data sets, in which

Table 5: MMRE values of Linear Regression, ANN, SVM-Regression, and SEEGE.

Data Set	Linear Regression	ANN	SVM-Regression	SEEGE-a	SEEGE-b
albrecht	1.15 ± 1.73	1.69 ± 2.98	1.62 ± 2.48	**0.24 ± 0.54**	0.29 ± 0.49
china	0.24 ± 0.10	0.32 ± 0.08	**0.10 ± 0.04**	0.11 ± 0.02	0.11 ± 0.02
coc81	20.56 ± 18.47	4.51 ± 2.96	6.59 ± 6.22	**1.06 ± 0.49**	2.34 ± 1.45
coc81lnh	18.94 ± 14.67	9.08 ± 7.21	44.04 ± 34.46	**1.23 ± 1.03**	3.53 ± 3.69
cocomo-sdr	2.34 ± 2.39	3.03 ± 2.36	2.91 ± 3.12	0.19 ± 0.57	**0.15 ± 0.45**
cocomonasa-v1	5.34 ± 4.53	0.93 ± 0.69	1.74 ± 1.75	**0.25 ± 0.10**	0.54 ± 0.29
desharnais	0.66 ± 0.26	1.33 ± 0.87	0.58 ± 0.24	**0.37 ± 0.09**	0.47 ± 0.10
kemerer	1.16 ± 0.92	1.23 ± 1.38	0.36 ± 0.41	0.18 ± 0.36	**0.13 ± 0.20**
maxwell	1.46 ± 1.73	0.95 ± 0.63	1.07 ± 0.94	**0.35 ± 0.10**	0.44 ± 0.10
nasa93	4.19 ± 3.42	6.48 ± 5.76	5.83 ± 4.61	**1.03 ± 0.52**	1.64 ± 1.24
Average Rank	4.0	4.2	3.6	1.4	1.8

SEEGE-b consistently provided the best models regardless of the evaluation measure. For exemplifying this behavior, consider the following model generated by SEEGE-b for a particular training fold of the cocomo-sdr data:

```
IF $(0.63 + x_{132}) > (0.63/x_{17})$
THEN effort $= x_0$
ELSE effort $= cos(x_{121}/(sin(x_{13}) - (-5 - x_{13})))$
```

Albeit its high complexity, the above model was required so SEEGE-b could achieve its good results in cocomo-sdr. The thin line between accurate modeling and data overfitting has to be carefully explored. When in doubt, we recommend the user to choose the less-complex model, since the comprehensibility of simple models increases the confidence of the project manager in the prediction of effort for new projects to be developed.

We also highlight that SEEGE is capable of producing comprehensible models (much the same as linear regression), whereas ANN and SVM-Regression are "black box" approaches that do not offer any insight on the predictions that are provided. A comprehensible model allows the stakeholder to create new hypotheses regarding the available data, and also visually inspect whether the predictive model produces a logical output — *e.g.*, more requirement documents and adjusted function points necessarily mean more effort. By careful inspecting the generated model, the project manager can detect possible errors in the model that are a result of a poor data collection process, and then act towards improving the data collection and pre-processing.

6. CONCLUSIONS

In this paper, we proposed a new grammatical evolution (GE) algorithm called SEEGE (Software Effort Estimation with Grammatical Evolution). To the best of our knowledge, this was the first work to develop a GE approach for software effort estimation. GE is the state-of-the-art in grammar-based evolutionary computation, since it combines the flexibility of grammar-based approaches with the simplicity and efficiency of linear-string genetic operators.

We implemented two different grammars for SEEGE, and we evaluated its performance through three different evaluation measures, namely RMSE, MAE, and MMRE. The experiments were conducted taking into account 10 public software effort data sets from the PROMISE repository [32]. For comparison purposes, we analyzed the performance of SEEGE against state-of-the-art machine learning algorithms such as SVMs for regression and artificial neural networks. Also, we compared SEEGE with least-square linear regression, which is a widely-employed technique for software effort estimation [22].

Results indicated that SEEGE clearly outperforms the baseline methods with statistical significance, according to the protocol recommended by Demšar [13]. In addition, they indicated that in most cases, a simpler grammar is preferred over a more detailed one. Overly-complex models were shown to be useful only in two out of the ten data sets employed in this study.

As future work, we plan to perform a deeper analysis on the causes of overfitting in order to develop a strategy that can detect it during evolution, and act accordingly. A multi-objective fitness function that takes into account model complexity could be developed to mitigate this problem.

Acknowledgments

The authors would like to thank Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), Brazil, for funding this research.

7. REFERENCES

[1] T. A. and D. G. Deriving models for software project effort estimation by means of genetic programming. In *KDIR 2009 - 1st International Conference on Knowledge Discovery and Information Retrieval*, 2009.

[2] R. C. Barros, D. D. Ruiz, N. N. Tenorio Jr., M. P. Basgalupp, and K. Becker. Issues on estimating software metrics in a large software operation. In *Proceedings of the 32nd Annual IEEE Software Engineering Workshop*, SEW '08, pages 152–160, Washington, DC, USA, 2008. IEEE Computer Society.

[3] M. P. Basgalupp, R. C. Barros, T. S. da Silva, and A. C. P. L. F. de Carvalho. Software effort prediction: A hyper-heuristic decision-tree based approach. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC 2013)*, 2013.

[4] M. P. Basgalupp, R. C. Barros, and D. D. Ruiz. Predicting software maintenance effort through evolutionary-based decision trees. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 1209–1214, New York, NY, USA, 2012. ACM.

[5] S. Berlin, T. Raz, C. Glezer, and M. Zviran. Comparison of estimation methods of cost and duration in it projects. *Inf. Softw. Technol.*, 51(4):738–748, Apr. 2009.

[6] P. L. Braga, A. L. I. Oliveira, and S. R. L. Meira. Software effort estimation using machine learning

techniques with robust confidence intervals. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence - Volume 01*, ICTAI '07, pages 181–185, Washington, DC, USA, 2007. IEEE Computer Society.

[7] P. Bruhn and A. Geyer-Schulz. Genetic programming over context-free languages with linear constraints for the knapsack problem: first results. *Evolutionary Computation*, 10(1):51–74, Mar. 2002.

[8] C. J. Burgess and M. Lefley. Can genetic programming improve software effort estimation? a comparative evaluation. *Information and Software Technology*, 43(14):863 – 873, 2001.

[9] A. Chavoya, C. Lopez-Martin, and M. E. M.-C. a. Applying genetic programming for estimating software development effort of short-scale projects. In *Eighth International Conference on Information Technology: New Generations*. IEEE Computer Society, 2011.

[10] N.-H. Chiu and S.-J. Huang. The adjusted analogy-based software effort estimation based on similarity distances. *J. Syst. Softw.*, 80(4):628–640, Apr. 2007.

[11] A. Corazza, S. Di Martino, F. Ferrucci, C. Gravino, F. Sarro, and E. Mendes. How effective is Tabu search to configure support vector regression for effort estimation? In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, PROMISE '10, pages 4:1–4:10, New York, NY, USA, 2010. ACM.

[12] G. Costagliola, S. Di Martino, F. Ferrucci, C. Gravino, G. Tortora, and G. Vitiello. Effort estimation modeling techniques: a case study for web applications. In *Proceedings of the 6th International Conference on Web engineering*, ICWE '06, pages 9–16, New York, NY, USA, 2006. ACM.

[13] J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[14] J. J. Dolado. On the Problem of the Software Cost Function. *Information and Software Technology*, 43(1):61–72, 1 Jan. 2001.

[15] M. O. Elish. Improved estimation of software project effort using multiple additive regression trees. *Expert Syst. Appl.*, 36(7):10774–10778, Sept. 2009.

[16] F. Ferrucci, C. Gravino, R. Oliveto, and F. Sarro. Genetic programming for effort estimation: An analysis of the impact of different fitness functions. In *Search Based Software Engineering (SSBSE), 2010 Second International Symposium on*. IEEE Computer Society, 2010.

[17] G. R. Finnie, G. E. Wittig, and J.-M. Desharnais. A comparison of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models. *J. Syst. Softw.*, 39(3):281–289, Dec. 1997.

[18] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1), 2009.

[19] A. Heiat. Comparison of artificial neural network and regression models for estimating software development effort. *Information and Software Technology*, 44(15):911 – 922, 2002.

[20] S.-J. Huang and N.-H. Chiu. Optimization of analogy weights by genetic algorithm for software effort estimation. *Information and Software Technology*, 48(11):1034 – 1045, 2006.

[21] R. Iman and J. Davenport. Approximations of the critical region of the friedman statistic. *Communications in Statistics*, pages 571–595, 1980.

[22] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE Trans. Softw. Eng.*, 33(1):33–53, Jan. 2007.

[23] M. Lefley and M. J. Shepperd. Using genetic programming to improve software effort estimation based on general data sets. *Genetic and Evolutionary Computation GECCO2003*, 2724(1):2477–2487, 2003.

[24] S. Luke. ECJ 20: A Java evolutionary computation library. http://cs.gmu.edu/∼eclab/projects/ecj/, 2004.

[25] R. M. MacCallum. Introducing a perl genetic programming system - and can meta-evolution solve the bloat problem? In *Proceedings of the 6th European conference on Genetic programming*, EuroGP'03, pages 364–373, Berlin, Heidelberg, 2003. Springer-Verlag.

[26] E. Mendes and N. Mosley. Bayesian network models for web effort prediction: A comparative study. *IEEE Trans. Softw. Eng.*, 34(6):723–737, Nov. 2008.

[27] A. L. Oliveira. Estimation of software project effort with support vector regression. *Neurocomputing*, 69(13–15):1749 – 1753, 2006.

[28] M. O'Neill and C. Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349 –358, aug 2001.

[29] P. Pendharkar, G. Subramanian, and J. Rodger. A probabilistic model for predicting software development effort. *IEEE Trans. Softw. Eng.*, 31(7):615 – 624, july 2005.

[30] C. Ryan, J. Collins, J. Collins, and M. O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Proceedings of the First European Workshop on Genetic Programming*, pages 83–95. Springer-Verlag, 1998.

[31] F. Sarro, F. Ferrucci, and C. Gravino. Single and multi objective genetic programming for software development effort estimation. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12. ACM, 2012.

[32] J. Sayyad Shirabad and T. Menzies. The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada, 2005.

[33] Y. Shan, R. I. Mckay, C. J. Lokan, and D. Essam. Software project effort estimation using genetic programming. In *Proceedings of International Conference on Communications Circuits and Systems*, pages 1108–1112. Press, 2002.

[34] I. Tanev and K. Shimohara. On role of implicit interaction and explicit communications in emergence of social behavior in continuous predators-prey pursuit problem. In *Proceedings of GECCO'03*, pages 74–85, 2003.