Search-Based Prediction of Fault Count Data

Wasif Afzal*, Richard Torkar and Robert Feldt
Blekinge Institute of Technology,
S-372 25 Ronneby, Sweden
{*waf,rto,rfd*}*@bth.se*

Abstract

Symbolic regression, an application domain of genetic programming (GP), aims to find a function whose output has some desired property, like matching target values of a particular data set. While typical regression involves finding the coefficients of a pre-defined function, symbolic regression finds a general function, with coefficients, fitting the given set of data points. The concepts of symbolic regression using genetic programming can be used to evolve a model for fault count predictions. Such a model has the advantages that the evolution is not dependent on a particular structure of the model and is also independent of any assumptions, which are common in traditional time-domain parametric software reliability growth models. This research aims at applying experiments targeting fault predictions using genetic programming and comparing the results with traditional approaches to compare efficiency gains.

1. Introduction to the research problem

A software fault is a defect in an executable product that causes system failures during operations [11]. The number of faults in a software module or particular release of a software system represents the quantitative measure of software quality. A fault prediction model then uses previous software quality data in the form of metrics (including software fault data) to predict the number of software faults in a module or release of a software system [12]. The practical aspect of such models has strong implications on the quality of the software project. The information gained from such models can be an important decision making tool for the project managers to make better decisions in uncertain situations. A fault prediction model helps a software development team in prioritizing the effort to be spent on a software project. If the predictions forecasts a high number of faults in the coming release of a project, then the management has the option of investing required levels of effort to circumvent possible failures in operation. Proper allocation of resources for quality improvement might cause considerable savings for a software project. The development of large software systems is costly therefore even small gains in prediction accuracy should be appreciable [10]. Apart from the efficiency gains, architectural improvements can be made by better designing high-risk segments of the system [14].

There have been a number of software fault prediction and reliability growth modeling techniques proposed in software engineering literature [9, 5]. Despite the presence of large number of models, there is no agreement within the research community about the best model. One of the reasons for such a situation is that models exhibit different predictive accuracies across different data sets. Therefore, the quest for a consistently accurate predictor model is continuing. The result is that the prediction problem is seen as being largely unsolvable and NP-hard: *the ability to build prediction systems for software engineers remains an important but largely unsolved problem...due to the fact that the problem is NP-hard* [23] *...this problem is largely unsolvable* [5].

2. Genetic programming for predictions

The use of statistical regression analysis (e.g., linear, logarithmic and logistic) for software fault predictions may not be the best approach. This argument is supported by the fact that software engineering data come with certain characteristics that creates difficulties in making accurate software prediction models. These

*Wasif Afzal is a PhD student, advised by Richard Torkar and Robert Feldt, at the Department of Systems and Software Engineering, Blekinge Institute of Technology, Sweden. This paper is written specifically for the PhD forum.

978-0-7695-3675-0/09 $25.00 © 2009 IEEE
DOI 10.1109/SSBSE.2009.17

characteristics include missing data, large number of variables, strong collinearity between the variables, heteroscedasticity[1], complex non-linear relationships, outliers and small size [10]. Therefore, it is not surprising that we possess an incomplete understanding of the phenomenon under study, so *it is very difficult to make valid assumptions about the form of the functional relationship between the variables* [4]. This reason is also highlighted by [21]. This argument strengthens earlier established results that show program metrics to be insufficient for accurate prediction of faults. Moreover, the acceptability of models has seen little success due to lack of meaningful explanation of the relationship among different variables and lack of generalisabilty of model results [10]. Additionally, these *parametric* models are often characterized by a number of assumptions [9] that are necessary for developing a mathematical model. These assumptions are often violated in real-world situations (see e.g. [24]), therefore, causing problems in the long-term applicability and validity of these models.

Under this scenario, what becomes significantly interesting is to have modeling mechanisms that can exclude the pre-suppositions about the model and is based entirely on the fault data. This is where the application of symbolic regression using genetic programming (GP) becomes feasible. The advantages of using GP for symbolic regression problems are [20]:

1. GP, being a non-parametric method, does not conceive a particular structure for the resulting function. Therefore, the evolved model truly represents the data, be it linear or non-linear.
2. The model and the associated coefficients are evolved based on the fault data collected during the initial test phase.
3. The equations are derived according to the fitness evaluation criterion of the individuals only, since GP does not make any assumptions about:
   (a) The distribution of the data.
   (b) Relationship between independent and dependent variables.
   (c) The stochastic behavior of software failure process.
   (d) The nature of software faults.

3. Related work

Studies reporting the use of GP for software fault prediction are few and recent. Costa et al. [6] presented results of two experiments exploring GP models based on time and test coverage. The authors compared the results with other traditional and non-parametric artificial neural network (ANN) models. For the first experiment, the authors used 16 data sets containing time-between-failure (TBF) data from projects related to different applications. The models were evaluated using five different measures, four of these measures represented different variants of differences between observed and estimated values. The results from the first experiment, which explored GP models based on time, showed that GP adjusts better to the reliability growth curve. Also GP and ANN models converged better than traditional reliability growth models. GP models also showed the lowest average error in 13 out of 16 data sets.

For the second experiment, which was based on test coverage data, a single data set was used. This time the Kolmogorov-Smirnov test was also used for model evaluation. The results from the second experiment showed that all metrics were always better for GP and ANN models. The authors later extended GP with boosting techniques for reliability growth modeling [19] and reported improved results. A similar study by Zhang and Chen [25] used GP to establish software reliability model based on mean time between failures (MTBF) time series. The study used a single data series and used six different criteria for evaluating the GP evolved model. The results of the study also confirmed that in comparison with the ANN model and traditional models, the model evolved by GP had higher prediction precision and better applicability.

Our research using GP extends these previous studies. We focus on using *cumulative* fault count data for modeling and investigate different ways to adapt the use of modeling in current trend of *multi-release* software development. We focus on using proven experimental design practices in our research work. We intend to increase the comparison groups and also make use of larger, real-world data sets to question the generalizability of our results.

3.1. Authors' contribution and preliminary work

In the preliminary stage of our research, we evaluated the use of GP for fault predictions in two studies ([3, 1]). In the very first study [3], we evaluated the results of using GP for modeling weekly fault count data of three industrial projects in terms of goodness of fit and predictive accuracy. The results found were statistically significant in favor of GP. We later extended the scope and included comparisons with three traditional reliability growth models [1]. In terms of evaluating

[1] A sequence of random variables with different variances.

model validity, three measures were used; two of them showed favorability of GP model, while the goodness of fit of the GP evolved model was also found to be either equivalent or better than the traditional models. Lastly, the predictions of the GP evolved model was found to be less biased than traditional models. We later on, in [2], highlighted the underlying mechanisms that allows GP to progressively search for fitter solutions.

4. Methodology

The overall methodology is discussed in terms of data requirements, GP design and statistical hypothesis testing.

4.1. Fault count data sets

Fault count data sets are required to train the GP evolved models and to evaluate their applicability using various evaluation measures. The fault count data sets resembles a time-series, with faults aggregated either on weekly or monthly basis. The week/month number can be regarded as the independent variable (being controllable) and the corresponding count of faults as the dependent variable in which the effect of the treatment is measured. The data sets needs to be split in to training and test sets. We resort to a typical mechanism, with first $\frac{2}{3}$ of data in each data set for building the model and later $\frac{1}{3}$ of the data for evaluating the model. Such a choice of split preserves the chronological time series occurrence of faults.

4.2. GP design

The representation of solutions in the search space is a symbolic expression in the form of a parse tree, which is a structure having functions and terminals. The quality of solutions is measured using an evaluation function. A natural evaluation measure for symbolic regression problems is the calculation of the difference between the obtained and expected results in all fitness cases, $\sum_{i=1}^{n} \mid e_i - e_i^{'} \mid$ where e_i is the actual fault count data, $e_i^{'}$ is the estimated value of the fault count data and n is the size of the data set used to train the GP models. Various variation operators can be used to grow or shrink a variable length parse tree. Similarly, there are various selection mechanisms that can be used to determine individuals in the next generation. The effectiveness of these operators is problem-dependent [16]. In our experiments, we have used cross-over with branch swapping by randomly selecting nodes of the two parent trees. We have also used mutation in which a random node from the parent tree is substituted with a new random tree created with the available terminals and functions. A small proportion of individuals were also copied into the next generation without any action of operators. The selection mechanism selected a random number of individuals from the population and chose the best of them; if two individuals were equally fit, the one having the less number of nodes was chosen as the best.

4.3. Statistical hypothesis testing

It is important to test results for statistical significance because it is not reliable to draw conclusions merely on observed differences in means or medians because the differences could have been caused by chance alone [17]. Prior to applying statistical testing, suitable accuracy indicators are required. However, there is no consensus with regards as to which accuracy indicator is the most suitable for the problem at hand. Commonly used indicators suffer from different limitations (for details see [7, 22]). One intuitive way out of this dilemma is to employ more than one accuracy indicator, so as to better reflect on a model's predictive performance in light of different limitations of each accuracy indicator. This way the results can be better assessed with respect to each accuracy indicator and we can better reflect on a particular model's reliability and validity. However, reporting multiple measures that are all based on a basic measure like mean relative error (MRE) would not be useful because all such measures would suffer from common disadvantage of being unstable (see [7]). In [18], measures for the following characteristics are proposed: Goodness of fit (Kolmogorov-Smirnov test), Model bias (U-plot), Model bias trend (Y-plot) and Short-term predictability (prequential likelihood). These measures, although providing a thorough evaluation of a model's predictions, lacks a suitable measure for variable-term predictability.

In [8, 15], average relative error is used as a measure of variable term predictability. To our knowledge, we are not aware of any critique of such an approach for variable-term predictability. As an example of applying multiple measures, one of our recent studies [1] used measures of prequential likelihood, Braun statistic and adjusted mean square error for evaluating model validity. Additionally we examined the distribution of residuals from each model to measure model bias. Lastly, the Kolmogorov-Smirnov test was applied for evaluating goodness of fit. More recently, analyzing distribution of residuals is proposed as an alternative measure [13, 22]. It has the convenience of applying significance tests and visualizing differences in absolute residuals of compet-

ing models using box plots.

5. Conclusions

This paper presented the synopses of the research conducted so far that evaluates the use of genetic programming for predicting fault count data. Initial studies have produced better or comparable results to traditional models (see [1]). This encourages further testing the use of GP for larger data sets and to increase comparisons with other machine learning/traditional approaches. Future work includes evaluating the use of GP for *cross-release* predictions using extensive data sets covering both commercial and open source software systems. Going further, the applicability of the approach will be assessed in an on-going project in an industrial context.

References

[1] W. Afzal and R. Torkar. A comparative evaluation of using genetic programming for predicting fault count data. In *Proceedings of the Third International Conference on Software Engineering Advances*. IEEE Computer Society, 2008.

[2] W. Afzal and R. Torkar. Suitability of Genetic Programming for Software Reliability Growth Modeling. In *The 2008 International Symposium on Computer Science and its Applications*. IEEE Computer Society, 2008.

[3] W. Afzal, R. Torkar, and R. Feldt. Prediction of fault count data using genetic programming. In *Proceedings of the 12th IEEE International Multitopic Conference*. IEEE, 2008.

[4] L. C. Briand, V. R. Basili, and W. M. Thomas. A pattern recognition approach for software engineering data analysis. *IEEE Trans. Softw. Eng.*, 18(11):931–942, 1992.

[5] V. Challagulla, F. Bastani, I.-L. Yen, and R. Paul. Empirical assessment of machine learning based software defect prediction techniques. *10th International Workshop on Object-Oriented Real-Time Dependable Systems*, 2005.

[6] E. Costa, S. Vergilio, A. Pozo, and G. Souza. Modeling software reliability growth with genetic programming. *International Symposium on Software Reliability Engineering*, 2005.

[7] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit. A simulation study of the model evaluation criterion MMRE. *IEEE Transactions on Software Engineering*, 29(11), 2003.

[8] K. Gao and T. Khoshgoftaar. A comprehensive empirical study of count models for software fault prediction. *IEEE Transactions on Reliability*, 56(2), June 2007.

[9] A. L. Goel. Software reliability models: Assumptions, limitations, and applicability. *IEEE Transactions on Software Engineering*, SE-11(12):1411–1423, 1985.

[10] A. Gray and S. MacDonnell. A comparison of techniques for developing predictive models of software metrics. *Information and Software Technology*, 39(6):425–437, 1997.

[11] T. Khoshgoftaar, N. Seliya, and N. Sundaresh. An empirical study of predicting software faults with case-based reasoning. *Software Quality Control*, 14(2), 2006.

[12] T. M. Khoshgoftaar and N. Seliya. Tree-based software quality estimation models for fault prediction. In *METRICS '02: Proceedings of the 8th International Symposium on Software Metrics*, Washington, DC, USA, 2002. IEEE Computer Society.

[13] B. Kitchenham, L. Pickard, S. MacDonell, and M. Shepperd. What accuracy statistics really measure. *IEE Proceedings Software*, 148(3), Jun 2001.

[14] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, 2008.

[15] Y. Malaiya, N. Karunanithi, and P. Verma. Predictability measures for software reliability models. *COMPSAC 90*.

[16] Z. Michalewicz and D. Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlag, second edition, 2004.

[17] I. Myrtveit and E. Stensrud. A controlled experiment to assess the benefits of estimating with analogy and regression models. *IEEE Transactions on Software Engineering*, 25(4), July-Aug. 1999.

[18] A. Nikora and M. Lyu. An experiment in determining software reliability model applicability. *ISSRE*, 1995.

[19] E. Oliveira, A. Pozo, and S. R. Vergilio. Using boosting techniques to improve software reliability models based on genetic programming. *IEEE International Conference on Tools with Artificial Intelligence*, 2006.

[20] R. Poli, W. Langdon, N. McPhee, and J. Koza. Genetic Programming: An Introductory Tutorial and a Survey of Techniques and Applications. Technical Report CES-475, ISSN: 1744-8050, 2007.

[21] M. Reformat, W. Pedrycz, and N. Pizzi. Software quality analysis with the use of computational intelligence. *Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on*, 2:1156–1161, 2002.

[22] M. Shepperd, M. Cartwright, and G. Kadoda. On building prediction systems for software engineers. *Empirical Software Engineering*, 5(3), 2000.

[23] M. Shepperd and G. Kadoda. Comparing software prediction techniques using simulation. *IEEE Transactions on Software Engineering*, 27(11):1014–1022, 2001.

[24] A. Wood. Software reliability growth models: assumptions vs. reality. In *ISSRE '97: Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering*, Los Alamitos, CA, USA, 1997. IEEE Computer Society.

[25] Y. Zhang and H. Chen. Predicting for MTBF failure data series of software reliability by genetic programming algorithm. *6th International Conference on Intelligent Systems Design and Applications (ISDA '06)*, 2006.