Suitability of genetic programming for software reliability growth modeling

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

Abstract

Genetic programming (GP) has been found to be effective in finding a model that fits the given data points without making any assumptions about the model structure. This makes GP a reasonable choice for software reliability growth modeling. This paper discusses the suitability of using GP for software reliability growth modeling and highlights the mechanisms that enable GP to progressively search for fitter solutions.

1. Introduction

In our modern society, we see software playing an important role in various domains, for example, air traffic control and automotive industry. The criticality of these applications demands that the software is reliable because software failures can have dire consequences. It is, therefore, imperative that the reliability of the software is determined before making it operational.

Deciding upon when to release the software is also important because releasing software that contains errors will result in high failures costs whereas, on the other hand, prolonged debugging and testing increases development costs. Reliability growth model is an important criterion, which helps in making an informed decision about when to release the software. A software reliability growth model (SRGM) describes the mathematical relationship of finding and removing faults to improve software reliability. A SRGM performs curve fitting of observed failure data by a pre-specified model formula, where the parameters of the model are found by statistical techniques like maximum likelihood method [11]. The model then estimates reliability or predicts future reliability by different forms of extrapolation [10]. After the first software reliability growth model was proposed by Jelinski and Moranda in 1972 [6], there have been numerous reliability growth models following it. These models come under different classes [9], e.g. exponential failure time class of models, Weibull and Gamma failure time class of models, infinite failure category models and Bayesian models. These models are based on prior assumptions about the nature of failures and the probability of individual failures occurring [7]. Some of these models are complex with many parameters. Without extensive mathematical background, practitioners cannot determine when a model is applicable and when it diverges from reality. There is no reliability growth model that can be generalized for all possible software projects, although there is evidence of models that are better suited to certain types of software projects [7]. Under this scenario what becomes significantly interesting is to have modeling mechanisms that can exclude the assumptions of the model and is based entirely on the fault data. This kind of adaptive model-building system can evolve a model from the actual characteristics of the given data sets. In this respect, genetic programming (GP) can be used as an effective tool because, being a non-parametric method, GP does not conceive a particular structure for the resulting model and GP also does not make any assumptions about the distribution of data.

2 Using GP for reliability growth modeling

GP can be used for modeling software reliability growth e.g. [4, 12, 14, 1]. The evolution of software reliability growth models using GP is an example of a symbolic regression problem. Symbolic regression is an error-driven evolution as it aims to find a function, in symbolic form, that fits (or approximately fits) data from an unknown curve [8]. In simpler terms, symbolic regression finds a function whose output matches some target values. GP is well suited for symbolic regression problems, as it does not make any assumptions about the structure of the function.

There are five preparatory steps for GP [3]:

1. Specifying the set of terminals.
2. Specifying the set of functions.
3. Specifying the fitness measure.

978-0-7695-3428-2/08 $25.00 © 2008 IEEE
DOI 10.1109/CSA.2008.13

4. Specifying the parameters for controlling the run.
5. Specifying the termination criterion and designating the result of run.

The set of terminals and functions make up a variety of programs in the population being searched by GP. For symbolic regression, the set of functions may consist of arithmetic functions while the terminal set may consist of independent variables and random constants. The specification of fitness measure specifies the desired objective of GP run. For symbolic regression problems, the fitness measure includes summing the errors measured for each record in the data set [13]. The specification of control parameters administers the GP run which includes setting different parameters like population size and probabilities of performing the genetic operations. The fifth preparatory step specifies the termination criterion (e.g. a maximum number of generations) and selection of an individual as a result of the run.

3 Suitability of GP

The suitability of GP for modeling software reliability growth is based on the identification of building blocks and progressively improving overall fitness.

According to Koza, the GP population contains building blocks, which could be any GP tree or sub-tree in the population. According to the building block hypothesis, good building blocks improve the fitness of individuals that include them and these individuals have greater chance to be selected for reproduction. Therefore, good building blocks get combined to form better individuals [2]. This hypothesis appears suited to adaptive model-building system that can be used for predicting software reliability growth.

The evolution of better individuals using GP is shown in Figure 1. The fitness of a GP solution is the sum of absolute differences between the obtained and expected results in all fitness cases. Suppose that during the fourth generation of the GP run, two tree solutions have evolved (Figure 1a and 1b) which contain different building blocks for an optimum solution. For tree 1 (Figure 1a), the sum of absolute differences between the obtained and expected results in all fitness cases was 31.34, while for tree 2 (Figure 1b), the fitness measure was 28.9. By combining these two trees, two new trees could emerge (Figure 1c & Figure 1d). The first tree (Figure 1c) has a better fitness of 27.8 than any of its parents, while the second tree (Figure 1d) produced a higher fitness of 39.

In order to evolve a general function based on the fitness cases, the search space of solutions can get complex. This increase in complexity helps the GP programs to be able to comply with all the fitness cases [13]. Evolutionary algorithms have been found to be robust for complex search spaces. In our work [1], we have used genetic programming

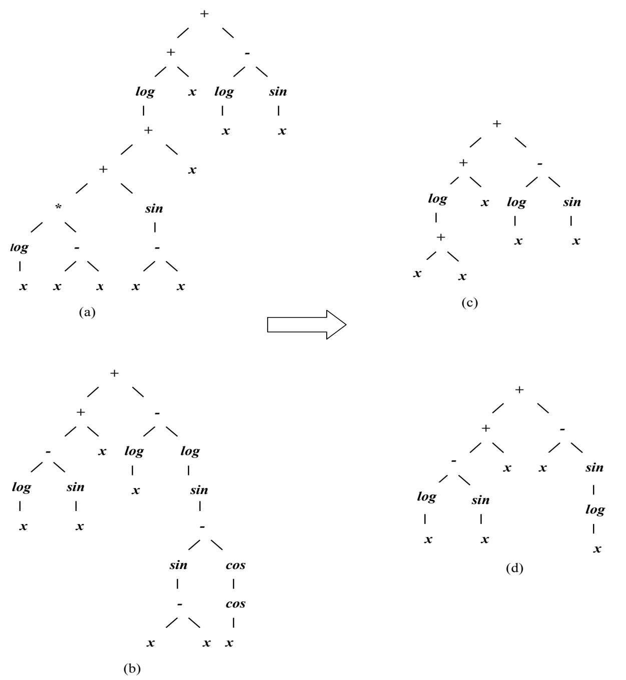

Figure 1. Combination of trees containing building blocks.

to evolve software reliability growth model because the suitability of genetic programming has already been proven for symbolic regression and curve fitting problems. Being a stochastic search technique, the different runs of GP would result in different trajectories [13]. Figure 3 shows how the GP algorithm is searching the program space of solutions to track the model to approximate. Figure 2 shows the Pareto front when modeling software reliability growth for one of the data sets. The Pareto front shows the set of solutions for which no other solution was found which both has a smaller tree and better fitness [5]. The Pareto front in the Figure 2 also shows how the fitness of different solutions fluctuates as the number of nodes increases during the course of generations.

4 Conclusions

GP is based on the exploration of space of computer programs. This exploration starts from random programs and through genetic operations of crossover, mutation and reproduction; a new generation of programs is evolved. This process is repeated until the termination criterion is satisfied. Through visualization of space of solutions, we can understand how GP is searching the space of solutions. GP is suited to symbolic regression problems and software reliability growth modeling is one instance of it. The experi-

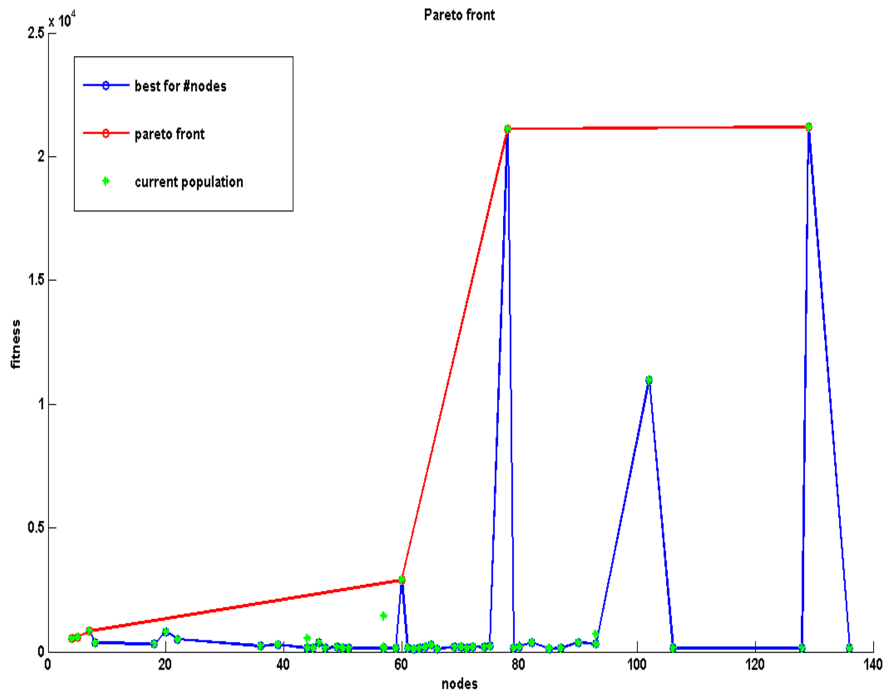

Figure 2. Visualization of Pareto front for one set of industrial fault count data.

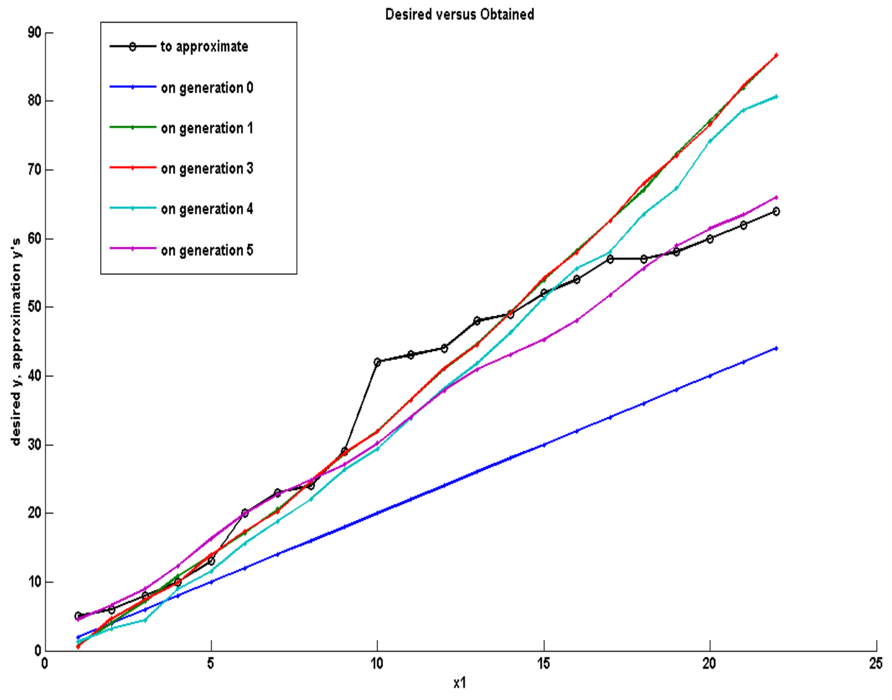

Figure 3. Several approximations to the original fault count data in different generations.

ments of using GP for software reliability growth modeling have indicated positive results, which warrant further investigation with larger real-world industrial data sets.

5 Acknowledgments

We thank Sara Silva for making available GPLAB, the genetic programming toolbox for MATLAB.

References

[1] W. Afzal and R. Torkar. A comparative evaluation of using genetic programming for predicting fault count data. Accepted at ICSEA'08: The 3rd International Conference on Software Engineering Advances, Sliema, Malta.

[2] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic programming - an introduction*. Morgan Kaufmann Publishers, Inc., 1998.

[3] E. K. Burke and G. Kendall. *Search methodologies*. Springer Science and Business Media, New York, USA, 2005.

[4] E. Costa, S. Vergilio, A. Pozo, and G. Souza. Modeling software reliability growth with genetic programming. In *ISSRE '05: Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*, Washington, USA, 2005. IEEE Computer Society.

[5] GPLAB—A genetic programming toolbox for MATLAB. `http://gplab.sourceforge.net`.

[6] Z. Jelinski and P. Moranda. Software reliability research. In *Statistical Computer Performance Evaluation, Ed. W. Freiberger*. Academic Press, USA, 1972.

[7] N. Karunanithi and Y. Malaiya. Neural networks for software reliability engineering. In *Handbook of software reliability engineering, Editor M. R. Lyu*, Hightstown, NJ, USA, 1996. McGraw-Hill, Inc.

[8] J. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, 1992.

[9] M. Lyu. *Handbook of software reliability engineering*. IEEE Computer Society and McGraw-Hill, 1996.

[10] M. R. Lyu. Software reliability engineering: A roadmap. In *FOSE '07: 2007 Future of Software Engineering*, Washington, DC, USA, 2007. IEEE Computer Society.

[11] I. J. Myung. Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology*, 47(1), 2003.

[12] E. Oliveira, A. Pozo, and S. Vergilio. Using boosting techniques to improve software reliability models based on genetic programming. In *ICTAI '06: Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*, Washington, USA, 2006. IEEE Computer Society.

[13] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008.

[14] Y. Zhang and H. Chen. Predicting for MTBF failure data series of software reliability by genetic programming algorithm. In *Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications*, Washington, USA, 2006. IEEE Computer Society.