Statistical Debugging Using a Hierarchical Model of Correlated Predicates

Saeed Parsa, Maryam Asadi-Aghbolaghi, and Mojtaba Vahidi-Asl

Department of Software Engineering,
Iran University of Science and Technology, Tehran, Iran
{parsa,m_vahidi_asl}@iust.ac.ir, m_asadi@comp.iust.ac.ir

Abstract. The aim of statistical debugging is to identify faulty predicates that have strong effect on program failure. In this paper predicates are fitted into a linear regression model to consider the vertical effect of predicates on each other and on program termination status. Prior approaches have merely considered predicates in isolation. The proposed approach in this paper is a two-step procedure which includes hierarchical clustering and the Lasso regression method. Hierarchical clustering builds a tree structure of correlated predicates. The Lasso method is applied on the clusters in some specified levels of the tree. This makes the method scalable in terms of the size of a program. Unlike other statistical methods which do not provide any context of the failure, the predicates contained in the group that is provided by this method can be used as the bug signature. The method has been evaluated on two well-known test suites, Space and Siemens. The experimental results reveal the accuracy and precision of the approach comparing with similar techniques.

Keywords: Automatic Bug Finding, Statistical Debugging, Hierarchical Clustering, Lasso Method, Fault Relevant Predicates.

1 Introduction

Although great efforts are made by software companies to remove software faults during in-house testing process, they cannot confidently claim that their deployed software is free of bugs [6]. Generally, a number of latent faults manifest themselves during operations in the hands of end users [1]. This has motivated many researchers during past few years to seek for techniques which automate the process of bug finding [8][12]. Among fault localization techniques, statistical debugging methods have achieved great success [2][3][4][5]. Statistical debugging approaches apply program logs representing the value of the predicates at failing and passing runs to find the fault relevant ones [5].

The majority of statistical debugging techniques cannot detect specific bugs caused by undesired interactions between predicates because they only consider predicates in isolation [4][5]. The situation becomes worse when bug(s) is located in multiple and probably far apart statements. Furthermore, many of these techniques merely provide the debugger with a single line of code or predicate while assuming that he/she has "*perfect bug understanding* " [21]. However, if the programmers are provided with a group of highly fault relevant predicates, they can find the origin of the bug and correct it, easier. To consider the simultaneous impact of predicates on each other and on the program termination status a logistic regression model [9] could be best applied [13]. To this end, ridge regression method [19] and a combination of ridge and lasso regression methods are applied [20]. For large programs with huge number of predicates ridge regression cannot provide interpretable models [10] while lasso models do. However, lasso does not identify correlated predicates. To resolve the difficulty, the predicates could be clustered according to their correlation.

In this paper, inspired from [22], a combination of a hierarchical clustering technique [24] and lasso method is presented. The hierarchical clustering provides a nested structure of correlated predicates. A distinct lasso model is then built for each level of the hierarchy to detect the fault relevant predicates. Finally we apply the majority voting technique on selected predicates of all involving levels to find out which predicate has got the highest suspiciousness score in the majority of levels. With this strategy, we improve the accuracy and precision of the method in finding and ranking fault relevant predicates. The clustering technique also helps us to identify correlated faulty predicates as the bug signature.

The remaining part of this paper is organized as follows. In section two, an overview of the method is described. The experiments and results are shown in section three. Finally concluding remarks and future works are depicted in section four.

2 The Method Overview

In this section, the main idea of the proposed technique is described in detail. The main phase of the technique is done in four stages: 1) preprocessing stage 2) constructing the hierarchical tree 3) building the lasso model and 4) identifying the fault relevant predicates.

2.1 The Preprocessing Phase

In order to construct statistical model from runtime behavior of a program, we should execute it several times with different test cases to collect the value of predicates in each run. To achieve this, it is required to insert some extra code before each predicate which is called instrumentation [6]. In this work, we have considered both function calls and branch statements to design the predicates [12].

2.2 Building the Nested Structure of Correlated Predicates

In order to construct the nested correlated structure of predicates, some initial steps should be performed [23]. First instrumented program is run by different test cases and the number of times each predicate has been evaluated as *True* in a particular run is collected and logged into a database. Then the data is normalized for clustering. The clustering is made according to the *Pearson* correlation coefficient [24] between two predicate values: $P_m=\{p_{m1}, p_{m2}, \ldots, p_{mN},\}$ and $P_n=\{p_{n1}, p_{n2}, \ldots, p_{nN},\}$ where p_{ij} depicts the number of times that predicate p_i is evaluated as *True* in the execution j and N is the number of executions. The Pearson coefficient is computed as follows:

$$\gamma_{m,n} = \frac{1}{N}\sum_{i=1..N}\left(\frac{p_{mi}-\overline{P_m}}{\sigma_{P_m}}\right)\left(\frac{p_{ni}-\overline{P_n}}{\sigma_{P_n}}\right) \tag{1}$$

The $\overline{P_m}$ and $\overline{P_n}$ in (1) are the average of values in P_m and P_n, respectively. σ_{P_m} and $\sigma_{P_n}$are the standard deviation of the corresponding predicates. The result would be a correlation matrix which represents the correlation between predicates. To achieve this, the *Average Linkage Clustering* [10] could be applied on the matrix. The algorithm for constructing the hierarchy is iterative: At first stage, for each actual predicate of the program, as the leaf of the tree, a predicate vector is assigned. Using these vectors, the correlation matrix is built according to the *Pearson* coefficient in (1). At this point, the highly correlated predicates are identified and clustered to form pseudo-predicates as the parents of leaves. Again a vector is allocated to each pseudo-predicate which contains the average value of all predicate vectors which are included in it. In a similar manner, we compute the *Pearson* correlation coefficient between this pseudo-predicate and all other predicates or pseudo-predicates. In this stage, the second level of the hierarchy from the bottom is constructed. The process continues until all pseudo-predicates and the remaining actual predicates are clustered to form the root of the hierarchy. The final result is a tree structure of correlated predicates.

2.3 Fitting Predicates to Lasso Model

The least absolute shrinkage and selection operator (lasso), first proposed by Tibshirani [11], shrinks some coefficients and tends to give zero to the remaining coefficients. The aim is to choose the coefficients, $\hat{\beta}_j$, *j=1, 2,... n*, that minimize the following equation, over all β_j, *j=1, 2, ... n*:

$$.\ \beta^{lasso} = \arg\min_{\beta}\sum_{i=1}^{N}(y_i - \beta_0 - \sum\nolimits_{j=1}^{n} p_{ij}\beta_j)^2 \quad \text{subject to} \quad \sum_{j=1}^{n}|\beta_j| \le s \tag{2}$$

The parameter *s* in (2) behaves as a constraint and if it is chosen adequately small, some of the coefficients may become zero. Therefore, lasso could be considered as a continuous subset selection technique which makes it appropriate for identifying bug relevant predicates. The loss function in (2) could be also converted to:

$$||y - P\beta||_2^2 + \lambda||\beta||_1 \tag{3}$$

Where *y* is the vector of program termination status in different runs, β is predicate coefficient matrix, *P* is the predicate (or pseudo-predicate) matrix and λ is the shrinkage tuning parameter. Finding an appropriate value for λ is a challenging problem, because with inadequate value we may lose fault relevant predicates. Lasso combines good features of naïve subset selection and ridge regression techniques and eliminates their drawbacks. It improves prediction accuracy that naïve subset selection suffers from and gives an interpretable model which is the main drawback of ridge regression.

As shown in previous section, the nested structure of predicates could be viewed as a tree. For each program, according to its size and complexity, a predefined number of levels of the tree are specified for which the lasso model should be constructed. At each specified level, the program runs are split into two categories of train and test sets. In order to find an appropriate shrinkage parameter for level *x*, first the lasso model is built with different values for λ. An appropriate shrinkage parameter is the one for which the corresponding lasso model constructed from the test set has the best prediction power.

2.4 Ranking Predicates

As mentioned in the previous section, for each specified level an individual lasso model is constructed. The predicates with non-zero coefficients are scored according to their weights. Using a majority voting technique, the predicates are ranked based on their score in different models (i.e. levels). Since the bug is not necessarily included in the predicate, we should scrutinize the code manually to find the cause of failure. Typically, the statistical debugging technique evaluates their effectiveness on the amount of manual code inspection by the user.

3 Experimental Results

In this section, the evaluation results of our approach on Siemens test suite and Space program are compared with some well known proposed techniques for software debugging. Siemens suite is available at [16]. The hierarchical clustering is performed using [23][24] and the lasso model is constructed using [14][15]. Among all built levels for a program, one third of levels in the middle are chosen to construct the lasso model. The cross validation involves almost 10% of a given program runs as the test set and 90% of it is the train set. In table 1, the number of localized bugs in terms of manually code inspection for each program in Siemens suite is presented. As shown in the table, the results are very interesting and manifest the high effectiveness of the proposed approach. Some versions have not been included in the experiments due to the segmentation faults, lack of failing test cases, and existence of a fault in the header file and none convergence of the lasso model. The proposed approach is compared with some other techniques in table 2. In overall the proposed technique has outperformed all other debugging techniques in terms of manual code inspection to locate the origin of failure.

In order to show the scalability of our approach, the proposed method is applied to Space [2]. Among 19 versions that we have evaluated the method, 16 faults were located by less than 0.5% of code examination and overally less than 10% lines of code examination is required for all 19 faults of this program. Tarantula [2] could locate 12 faults of Space by less than 1% of code examination.

Table 1. The result of proposed technique on Siemens programs

Code inspection	Schedule	Schedule2	Print_tokens	PrintTokens2	Replace	Totinfo	Tcas	Total
< 1 %	2	2	3	5	13	10	5	40
< 10 %	5	7	4	7	18	16	32	89
< 20 %	5	8	5	9	23	21	38	109
used versions	5	8	5	9	23	21	38	109
All versions	9	10	7	10	32	23	41	127

Table 2. A complete comparison with some outstanding debugging techniques

Percentage of Examined Code	Proposed Method	Tarantula [2]	SOBER [3]	Liblit05 [7]	Context Aware[4]	CT [17]	Argus [8]	NN (perm)[18]
< 1 %	40	17	10	11	38	6	0	0
< 10 %	89	68	52	68	67	34	74	18
< 20 %	109	75	85	96	73	49	98	28
< 30 %	109	87	91	102	-	67	107	41

4 Concluding Remarks and Future Work

In this paper a new statistical approach for software debugging has been proposed. Two important issues are considered in this paper: 1) eliminating all irrelevant and redundant predicates to obtain a small subset of faulty predicates and 2) finding a group of fault relevant predicates as the context of bug for better understanding the origin of failure and fixing the bug. The approach constructs a tree view structure according to the correlation among predicates. Then, an average linkage clustering technique is applied on correlation matrix of predicates. The lasso model is constructed for some specified levels and high ranked predicates are identified and reported as faulty predicates. Since the clustering method groups correlated predicates, the result of applying lasso on generated predicates is more accurate and efficient. The grouping effect of the method helps the programmer to identify groups of fault relevant predicates as bug signatures.

Our experiments has been done on Siemens test suite and Space program. The results are promising and show the effectiveness of the proposed techniques in comparison with other debugging techniques. For future work, we aim to extend the work on larger and more complex programs with multiple bugs. We may also consider the static structure of the program in identifying correlated predicates.

References

1. Liblit, B.: Cooperative Bug Isolation. PhD thesis. University of California, Berkeley (2004)
2. Jones, J.A., Harrold, M.J.: Empirical evaluation of the tarantula automatic fault localization technique. In: 20th IEEE/ACM International Conference on Automated Software Engineering, pp. 273–282. ACM Press, Long Beach (2005)
3. Liu, C., Yan, X., Fei, L., Han, J., Midkiff, S.P.: Sober: Statistical model-based bug localization. In: 10th European Software Eng. Conf./13th ACM SIGSOFT Int'l Symposium Foundations of Software Engineering, pp. 286–295. ACM Press, Lisbon (2005)
4. Jiang, L., Su, Z.: Context-aware statistical debugging: from bug predictors to faulty control flow paths. In: Twenty-Second IEEE/ACM International Conference on Automated Software Engineering, pp. 184–193. ACM Press, Atlanta (2007)
5. Arumuga Nainar, P., Chen, T., Rosin, J., Liblit, B.: Statistical debugging using compound Boolean predicates. In: International Symposium on Software Testing and Analysis, pp. 5–15. ACM Press, London (2007)
6. Zeller, A.: Why Programs Fail: A Guide to Systematic Debugging. Morgan Kaufmann, San Francisco (2006)

7. Liblit, B., Naik, M., Zheng, A., Aiken, A., Jordan, M.: Scalable Statistical Bug Isolation. In: Int'l Conference Programming Language Design and Implementation, Chicago, pp. 15–26 (2005)
8. Fei, L., Lee, K., Li, F., Midkiff, S.P.: Argus: Online statistical bug detection. In: Baresi, L., Heckel, R. (eds.) FASE 2006. LNCS, vol. 3922, pp. 308–323. Springer, Heidelberg (2006)
9. Chatterjee, S., Hadi, A., Price, B.: Regression Analysis by Example, 4th edn. Wiley Series in Probability and Statistics, New York (2006)
10. Hastie, T.J., Tibshirani, R.J., Friedman, J.: The Elements of Statistical Learning: Data Mining Inference and Prediction. Springer, New York (2001)
11. Tibshirani, R.: Optimal Reinsertion: Regression shrinkage and selection via the lasso. J. R. Statist. Soc. 58, 267–288 (1996)
12. Zheng, A.X., Jordan, M.I., Liblit, B., Naik, M., Aiken, A.: Statistical debugging: simultaneous identification of multiple bugs. In: 23rd International Conference on Machine Learning, pp. 1105–1112. ACM Press, NY (2006)
13. Liblit, B., Aiken, A., Zheng, X., Jordan, M.I.: Bug isolation via remote program sampling. In: ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation, pp. 141–154. ACM Press, San Diego (2003)
14. Friedman, J., Hastie, T., Tibshirani, R.: Lasso: Glmnet for Matlab - Lasso (L1) and elastic-net regularized generalized linear models
15. Friedman, J., Hastie, T., Tibshirani, R.: Lasso: Glmnet for R (2011), `http://cran.r-project.org/web/packages/glmnet/index.html`
16. Software-artifact infrastructure repository, `http://sir.unl.edu/portal`
17. Cleve, H., Zeller, A.: Locating causes of program failures. In: 27th International Conf. on Software Engineering, St. Louis Missouri, pp. 342–351 (2005)
18. Renieris, M., Reiss, S.: Fault localization with nearest neighbor queries. In: 18th IEEE International Conference on Automated Software Engineering, Montreal, pp. 30–39 (2003)
19. Parsa, S., Vahidi-Asl, M., Arabi, S.: Finding Causes of Software Failure Using Ridge Regression and Association Rule Generation Methods. In: Ninth ACIS International Conference on Parallel/Distributed Computing, Phuket, pp. 873–878 (2008)
20. Parsa, S., Arabi, S., Vahidi-Asl, M.: Statistical Software Debugging: From Bug Predictors to the Main Causes of Failure. In: Software Metrics and Measurement: SMM 2009 in Conjunction with the Second International Conference on Application of Digital Information and Web Technologies, London, pp. 802–807 (2009)
21. Cheng, H., Lo, D., Zhou, Y., Wang, X.: Identifying Bug Signatures Using Discriminative Graph Mining. In: International Symptoms on Software Testing and Analysis, pp. 141–151. ACM Press, Chicago (2009)
22. Park, M., Hastie, T., Tibshirani, R.: Averaged gene expressions for regression. Biostatistics Journal, 212–227 (2007)
23. Eisen, M.: Hierarchical Clustering: Cluster and TreeView are an integrated pair of programs for analyzing and visualizing the results of complex microarray experiments, `http://rana.lbl.gov/EisenSoftware.htm`
24. Eisen, M., Spellman, P., Brown, P., Botstein, D.: Cluster analysis and display of genomewide expression patterns. Proceedings of the National Academy of Sciences of the United States of America 95, 14863–14868 (1998)