Does Principal Component Analysis Improve Cluster-Based Analysis?

Joan Farjo, Rawad Abou Assi, Wes Masri, and Fadi Zaraket
Department of Electrical and Computer Engineering
American University of Beirut
Beirut, Lebanon
{jmf09, ria21, wm13, fz11}@aub.edu.lb

***Abstract*- Researchers in the dynamic program analysis field have extensively used *cluster analysis* to address various problems. Typically, the clustering techniques are applied onto execution profiles having high dimensionality (i.e., involving a large number of profiling elements), sometimes in the order of thousands or even hundreds of thousands. Our concern is that the high number of profiling elements might diminish the effectiveness of the clustering process, which led us to explore the use of dimensionality reduction techniques as a preprocessing step to clustering.**

Specifically, in this work, we used *PCA* (*Principal Component Analysis*) as a dimensionality reduction technique and investigated its impact on two cluster-based analysis techniques, one aiming at identifying coincidentally correct tests, and the other at test suite minimization. In other words, we tried to assess whether PCA improves cluster-based analysis. Our experimental results showed that the impact was positive on the first technique, but inconclusive on the second, which calls for further investigation in the future.

***Keywords*-** *PCA (Principal Component Analysis), cluster analysis, dimensionality reduction, test suite minimization, coincidental correctness.*

I. Introduction

Cluster analysis has been used in several areas of dynamic software analysis, such as test suite minimization, fault localization [8][10], and application-based intrusion detection [18]. The clustering techniques are applied onto execution profiles comprising profiling elements that varied in terms of complexity, e.g., statements, edges, def-uses, information flow pairs [17], slice pairs [11][19], and paths [21]. Also, typically these execution profiles exhibit high dimensionality, i.e., include thousands or even hundreds of thousands of profiling elements. Our concern is that the high number of profiling elements might diminish the effectiveness of the clustering process, which led us to explore the use of dimensionality reduction techniques as a preprocessing step to clustering.

The goal of this work is to investigate the impact of dimensionality reduction on cluster-based dynamic program analyses, and specifically, the impact of *PCA* (*Principal Component Analysis*) [23] on two cluster-based analysis techniques, one aiming at identifying coincidentally correct tests, and the other at test suite minimization. In other words, we tried to answer the following question: "*Does Principal Component Analysis Improve Cluster-Based Analysis?*".

We first describe PCA and the basis behind it (Section II). Then we describe our two experimental studies (Section III and IV). Finally, Section V presents our conclusions and future work.

II. Principal Component Analysis

Principal Component Analysis (*PCA*) is an unsupervised and linear technique that reduces the dimensionality of a data set (possibly involving correlated variables) to a new set involving uncorrelated variables. The generated uncorrelated variables are called *principal components* (*PC*s). The obtained set has the *PC*s ordered by the fraction of the total information/variation each retains. That is, the first PC captures as much of the variability present in the data set as possible, the second PC also captures as much of the variability but under the constraint of being uncorrelated with the previous (first) PC, and similarly for the subsequent PCs. PCA is typically used in situations where high dimensionality data needs to be reduced, therefore, after applying it, only specific PCs are considered and the remaining ones ignored according to the criteria adopted for choosing the eigenvectors as will be explained in the following section. Also, PCA can be viewed as a technique for removing redundant information from a data set [22][6]. This redundancy is likely due to the fact that some variables measure the same or related constructs. In the context of software analysis, this type of redundancy is prevalent mostly due to the transitivity relationships induced by control and data dependences.

A. Details

PCA transforms the original data of dimension n to a new coordinate system such that the greatest variance by any projection of the data lies on the first coordinate or PC_1, the second greatest variance on the second coordinate or PC_2, and so on [6]. The number of PCs (or eigenvectors) extracted in a PCA is equal to the number of observed variables being analyzed, i.e., the new coordinate system is also of dimension n [6][23]. And only the first few components account for significant amounts of variance.

PC_1 accounts for a maximal amount of total variance, which means that it will be correlated with at least some (or many) of the observed variables. PC_2 will account for a maximal amount of variance in the data set that was not accounted for by PC_1, meaning that PC_2 will be correlated with some of the observed variables that did not display strong correlations with PC_1. In addition, PC_2 will be uncorrelated with PC_1[6][5]. Each remaining PC accounts for a maximal amount of variance in the observed variables that was not accounted for by the preceding components, and is uncorrelated with all of the preceding components, having in mind that now each lower component accounts for smaller variance.

There are several criteria for choosing the number of eigenvectors (PCs) to retain, including, the *eigenvalue-one* criterion [9], the *scree test* [3], the *J-measure*, the *SEPCOR-algorithm*, the *proportion of variance accounted for* criterion [6], and the *cumulative percent of variance accounted for* criterion [6][5] known also as the *m-method*, which we use in this work. The *m-method* retains enough components so that the cumulative percent of variance accounted for is equal to some minimal value. For example, if PC_1, PC_2, PC_3, PC_4, accounted for 45%, 38%, 10%, and 5% of the total variance, respectively. It suffices to only retain PC_1, PC_2, and PC_3 if the minimal value sought was 90%.

B. PCA Steps

Below are the main steps for conducting PCA [23]:

a. Compute the mean for each dimension and subtract it from the data of that dimension in order to obtain the *MeanAdjustedData* with a mean of 0
b. Build the covariance matrix
c. Extract the eigenvectors (PCs) and eigenvalues of the covariance matrix. An eigenvalue represents the amount of variance that is accounted for by a given eigenvector
d. Order the eigenvectors in decreasing order of their eigenvalues
e. Choose the eigenvectors of interest according to the *cumulative percent of variance accounted for* criterion, to form the *FeatureVector* (a matrix with the chosen eigenvectors in the columns)
f. Compute the transpose of the *FeatureVector* and multiply it with the transposed *MeanAdjustedData*

$$FinalData = FeatureVector^T \times MeanAdjustedData^T$$

g. *FinalData* represents the original data in terms of the eigenvectors we chose

III. Experimental Study-I

A. Identifying Coincidentally Correct Tests

The recognized conditions for failure to be observed are [24][2]: 1) the defect got executed, 2) the program has transitioned into an infectious state, and 3) the infection has propagated to the output. *Coincidental correctness* (*CC*) [13][14][7][25][15] arises when the program produces the correct output while condition (1) is met but neither (2) nor (3) occurred. Identifying CC tests in order to cleanse test suites from coincidental correctness was shown to enhance *coverage-based fault localization* (*CBFL*) [1][12][8][10].

To help describe our experiment, we first define few terms. A test suite T comprises a set of passing tests T_P and a set of failing tests T_F, where T_P might be composed of a subset of coincidentally correct tests T_{CC} and another subset of true passing tests T_{trueP}, i.e., passing tests that did not execute the fault. The aim is to identify T_{CC} given T_F and T_P so that the tests in T_{CC} would be moved from T_P to T_F or discarded in order to enhance the effectiveness of CBFL.

A simple technique for identifying T_{CC} conjectures that coincidentally correct tests are similar to the failing tests, and thus should cluster together. Therefore, given a single fault program and an associated test suite in which tests are categorized as passing and failing. Creating two clusters out of the test suite should ideally lead to one cluster comprising T_{trueP} and another comprising T_F and T_{CC}.

B. Experimental Setup and Results

In this experiment we contrast the performance of the above cluster-based technique from when it is applied to the original data set to when PCA is first used to reduce the dimensionality of the data set. Note that *K-means* clustering is used in this experiment where k is set to 2. Table 1 shows information about the used programs and the composition of their test suites. This study was conducted using 10 seeded versions from the Siemens benchmark. Since our available profiling tools only supported Java programs, the programs were converted to Java as part of previous work [14]. Note that the execution profiles in this study included all of the following three profiling types: basic blocks, branches, and def-use pairs.

Table 1 also summarizes our results for when PCA was used and when it was not used. For each subject it shows: 1) the average % of false positives (FP), 2) the average % of false negatives (FN), and 3) the average % of false alarms (FA=FN+FP). Note that FN assesses whether or not we are successfully identifying all of the CC tests, and FP assesses whether we are erroneously categorizing tests as CC. It's worth mentioning that the *K-means* algorithm we used is deterministic since the two selected initial means correspond to the passing and failing test cases that are separated by the largest measured distance.

Table 1. Subject Programs and Results

Program	|T|	|T_F|	|T_P|	|T_{CC}|	Without PCA			With PCA		
					FN	FP	FA	FN	FP	FA
*print_tokens*_v7	4070	28	4042	357	0%	87%	87%	22%	8%	30%
*print_tokens2*_v4	4055	332	3723	1099	0%	95%	95%	11%	40%	51%
*tot_info*_v22	1052	23	1029	843	0%	49%	49%	0%	10%	10%
*replace*_v28	2843	18	2825	801	55%	47%	102%	56%	46%	102%
*schedule*_v2	2650	210	2440	1382	0%	50%	50%	8%	25%	33%
*schedule*_v3	2650	159	2491	1199	0%	62%	62%	7%	38%	45%
*schedule*_v4	2650	294	2356	1481	0%	43%	43%	8%	14%	22%
*schedule*_v8	2650	31	2619	1311	0%	58%	58%	7%	39%	46%
*tcas*_v16	1597	70	1527	1497	46%	0%	46%	46%	0%	46%
*tcas*_v25	1597	3	1594	396	0%	40%	40%	0%	40%	40%

It could be observed from the results in Table 1 (the FA columns in particular) that for 7 out of the 10 subject programs PCA considerably improved the CC identification process, and for the remaining 3 subject programs it had no positive or negative impact. It is also worth noting that in 2 out of the 3 latter cases (i.e., for *tcas_v16* and *tcas_v25*), the programs are relatively small and exhibit simple structure.

IV. Experimental Study-II

A. Cluster-based Test Suite Minimization

Test suite minimization involves selecting a subset of tests *T'* from an existing test suite *T* in order to reduce the cost of the testing process. An effective minimization technique would yield a *T'* that is manageable in size and that reveals all (or most of) the defects revealed by *T*.

Coverage based test suite minimization techniques analyze the execution profiles of a program in order to construct a *T'* such that all the profiling elements covered by *T* are also covered by *T'* [20][16]. Distribution-based test suite minimization [20] techniques select test cases based on how their execution profiles are distributed in the multidimensional profile space. In this experiment, we use a distribution-based technique comprising the following steps:

a) For each pair of tests (execution profiles), a metric that represents their degree of dissimilarity is computed. We used the following dissimilarity (binary) metric [4]:

$$\delta\_\text{binary}_{ij} = \sqrt{\sum_{r=1}^{k} | \, b_{ir} - b_{jr} \, |}$$

where *i* and *j* are test cases, *r* is a profiling element, and *k* is the total number of distinct profiling elements induced by the test suite. Using this dissimilarity metric, the smaller the number of common elements executed by *i* and *j*, the larger the value of the metric.

b) Based on the computed metrics, *K-means* clustering is applied to partition the population into *k* clusters.

c) One test is randomly selected from each cluster similar to what is done in [4]. This one-per-cluster sampling technique economically exercises each program behavior represented by a cluster, and it also favors the selection of unusual executions, which tend to be placed in isolated clusters.

d) The *k* selected tests represent the tests in *T'*.

B. Experimental Setup and Results

This experiment compares the performance of two variants of the above cluster-based test suite minimization technique. In the first, the original execution profiles are used, whereas in the second PCA is first applied.

Table 2 describes the subject programs, original test suites, and minimized test suites using PCA and without using it. In order to account for the variability of the samples, the algorithm's steps were applied 1000 times and the results were averaged. Therefore, the reported |*T'*| correspond to when *T'* reveals 100%, on average, of the defects revealed in *T*. Note that each test suite contained no more that 5% failures.

Two points are worth pointing out in regard to our use of PCA. First, enough components (eigenvectors) are retained so that the cumulative percent of variance accounted for is equal to 90% (see Section II.A). Second, Table 2 reports two values for |*T'*|: a) the smallest/earliest |*T'*| that revealed 100% of the defects on average, denoted by "Smallest"; b) the smallest |*T'*| that revealed 100% of the defects on average with the additional constraint that it was not followed by a larger |*T'*| that revealed less than 100%, denoted by "Stable". Also, for the PCA based technique, the total initial number of eigenvectors and the number of eigenvectors (PCs) retained to attain 90% variability, are reported.

Note that the *Space* program, which is written in C, was profiled using GCov as opposed to our own profiling tools which only support Java programs.

Table 2. Subject Programs and Results

Program	**# Faults**	**|T|**	**|T'|: NonPCA**		**|T'|: PCA**			
			Smallest	**Stable**	**tot. #Eig**	**#Eig**	**Smallest**	**Stable**
Space	5	2000	300	400	3164	25	300	300
JTidy	4	2000	200	200	5110	20	250	250
print_tokens2	7	3691	550	2550	278	8	500	3300
tot_info	6	952	150	900	271	8	850	950
schedule	3	1406	750	750	238	14	450	550

Based on Table 2 we make the following observations:

1) For *JTidy* and *print_tokens2*, PCA had a clear negative impact

2) For *tot_info*, PCA had a minor negative impact

3) And for *Schedule* and *Space*, it had a clear positive impact

That is, the impact of PCA on cluster-based test suite minimization was inconclusive.

V. Conclusions and Future Work

The aim of this work was to use *PCA* (*Principal Component Analysis*) as a dimensionality reduction technique and investigate its impact on two cluster-based analysis techniques, one aiming at identifying coincidentally correct tests, and the other at test suite minimization. Our experimental results showed that the impact was positive on the first technique, but inconclusive on the second.

In the future, we will experiment with different criteria to choose our eigenvectors, and we will investigate variations of PCA (e.g., Kernel PCA and Probabilistic PCA) and other dimensionality reduction techniques.

Acknowledgment

This research was supported in part by NSF grant no. 0819987.

References

[1] Abou-Assi R. and Masri W. Identifying Failure-Correlated Dependence Chains. *First International Workshop on Testing and Debugging*, TeBug/ICST 2011, Berlin, March 2011.

[2] Ammann P. and Offutt J. Introduction to Software Testing. Cambridge University Press, 2008.

[3] Cattell, R. B. (1966). The scree test for the number of factors. Multivariate Behavioral Research, 1, 245-276.

[4] Dickinson W., Leon D., and Podgurski A. Finding Failures by Cluster Analysis of Execution Profiles. ICSE 2001: 339-348.

[5] Fodor I. K. A survey of dimension reduction techniques. Center for Applied Scientific Computing, Lawrence Livermore National Laboratory. June 2002.

[6] Hatcher, Larry. 1994. *A Step-by-Step Approach to Using SAS for Factor Analysis and Structural Equation Modeling*. Cary, NC: SAS Institute Inc.

[7] Hierons R. M. Avoiding coincidental correctness in boundary value analysis. ACM TOSEM. Volume 15, Issue 3 (July 2006). Pages: 227 - 241.

[8] Jones J., Harrold M. J., and Stasko J. Visualization of Test Information to Assist Fault Localization. ICSE 2001,467-477.

[9] Kaiser, H. F. (1960). The application of electronic computers to factor analysis. Educational and Psychological Measurement, 20, 141-151.

[10] Liblit B., Aiken A., Zheng A., and Jordan M. 2003. Bug Isolation via Remote Program Sampling. Proc. ACM SIGPLAN, PLDI 2003, pp. 141-154, 2003.

[11] Masri, W. Exploiting the Empirical Characteristics of Program Dependences for Improved Forward Computation of Dynamic Slice. Empirical Software Engineering (ESE) (Springer), 2008 13:369-399.

[12] Masri W. Fault localization based on information flow coverage. Softw. Test., Verif. Reliab. 20(2): 121-147 (2010).

[13] Masri W., Abou-Assi R. Cleansing Test Suites from Coincidental Correctness to Enhance Fault-Localization. Third International Conference on Software Testing, Verification and Validation, ICST 2010, Paris, April, 2010.

[14] Masri W., Abou-Assi R., El-Ghali M., and Fatairi N. An Empirical Study of the Factors that Reduce the Effectiveness of Coverage-based Fault Localization. International Workshop on Defects in Large Software Systems, DEFECTS, Chicago, IL, 2009.

[15] W. Masri, R. Abou Assi, F. Zaraket, and N. Fatairi. Enhancing Fault Localization via Multivariate Visualization, Regression/ICST 2012, Montreal, Canada, April 2012.

[16] Masri W. and El-Ghali M. Test Case Filtering and Prioritization Based on Coverage of Combinations of Program Elements. Seventh International Workshop on Dynamic Analysis, WODA, Chicago, IL, 2009.

[17] Masri W., Halabi H. An algorithm for capturing variables dependences in test suites. Journal of Systems and Software (JSS) 84(7): 1171-1190 (2011).

[18] Masri, W. and Podgurski, A. Application-Based Anomaly Intrusion Detection with Dynamic Information Flow Analysis. Computers & Security (Elsevier). Vol. 27 (2008), pages 176-187.

[19] Masri, W. and Podgurski, A. Algorithms and Tool Support for Dynamic Information Flow Analysis. Information and Software Technology (IST) (Elsevier). Vol. 51 (Feb. 2009), pages 385-404.

[20] Masri W., Podgurski A. and Leon D. An Empirical Study of Test Case Filtering Techniques Based On Exercising Information Flows. IEEE Transactions on Software Engineering, July, 2007, vol. 33, number 7, page 454

[21] Reps, T., Ball, T., Das, M., and Larus, J., The use of program profiling for software maintenance with applications to the Year 2000 Problem. *Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Zurich, 1997

[22] Shlens J. A Tutorial on Principal Component Analysis. Center for Neural Science, New York University. April 22, 2009.

[23] Smith L. I. A tutorial on Principal Components Analysis, February 26, 2002.

[24] Voas J. 1992. PIE: A Dynamic Failure-Based Technique. IEEE Trans. Software Eng. 18(8): 717-727 (1992).

[25] Wang X., Cheung S.C., Chan W.K., Zhang Z. 2009. Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization. ICSE 2009, pp. 45-55