# A Review of Compositional Model-based and Spectrum-based techniques for Improved Bug Localization

Amratanshu Shrivastava
Dept. of CSIS (Computer Science and Information Systems)
Birla Institute of Technology and Science
Pilani, Rajasthan, India
{f20170224@pilani.bits-pilani.ac.in}
{University ID: 2017A7PS022P}

***Abstract*—Software developers spend a proportionate amount of time in locating bugs in their source codes. Bug reports are filed and a large number of files are checked and processed in an attempt to find and rectify the errors. Numerous techniques have been proposed in the field of Information Retrieval to localize the bugs using text-based bug reports. Fault localization techniques can be classified into two categories – spectrum based and model-based (We focus on Vector Space Model for finding bugs). No single spectrum-based technique can optimally work on every dataset. Similarly, Vector Space Models (VSM) can vary drastically based on their tf-idf choices and no single VSM model can optimally work on every dataset. Therefore, compositions of various of these techniques is done and then an optimal solution is found using search-based algorithms (genetic algorithm and Simulated annealing). This is a review of a Compositional Vector Space Model [1] and a Search-based Fault Localization [2] method for improving bug localization.**

***Keywords—Bug localization, Vector Space Model, Search-based optimization problem***

## I. Introduction

A significant time in coding a final-working product required bug-fixing. Maintenance tasks can account for up to 90% of the overall cost of a software project [3]. Various techniques have been proposed to reduce the time and cost of finding the buggy files responsible. The larger the code-base, greater is the time required for locating the particular files responsible for errors and rectifying them. Comprehension costs need to be reduced and many techniques have been proposed to do the same.

Like all other Information Retrieval models, it is now a general fact that there is no perfectly capable model. A technique which is efficient and accurate on one dataset may prove to perform worse than the already established slower techniques, and vice versa. Therefore, composite models have been proposed which combine multiple versions of these techniques depending on the dataset and provide better results than a single one. From the permutations tried and tested, some produce better results and some don't. In the end, this becomes an optimization problem to find which composite model performs better than the others.

Based on the data used, fault localization techniques can be broken down into broad categories. This paper reviews a Compositional Vector Space Model [1] and a Search-based Fault Localization method [2] for improving the bug-locating abilities of an IR System. Both these compositional models use Search-based Optimization algorithms incorporating Genetic Algorithm [5] and Simulated Annealing [6] to find out the best approach possible.

## II. Compositional Vector Space Model

Bug reports are often human-readable versions of the errors represented in Natural Language. Extraction of data through Natural language data and applying the Vector Space Model on them has proven to perform better than many other non-VSM based Information Retrieval techniques. VSM model suffers from the fact that the Standard TF-IDF model can often be outperformed by other weighting schemes.

### *A. Vector Space Model - Background*

Vector Space Model finds similarity between documents using vectors of the terms forming the documents. Each value in the vector represents the weight of that term in the document. Higher values of weight imply that a term is more *important* for comparison than other terms. Frequently used terms like 'The' and 'A' are often useless in conveying the real information and purpose of a document/query. The formula used for calculating the weight of a term follows –

$$weight(t, d) = tf(t, d) \times idf(t, D)$$

where tf represents the term frequency of a term in that document and idf is the inverse document frequency. The standard method is to take tf as raw term frequency and to give more weight to rare terms, inverse document frequency is calculated as the logarithm of the reciprocal of the document frequency (df) [1]. Given a query, VSM first converts the documents and queries both into a bag of words i.e. the order does of appearance does not matter. Before reading from the corpus (documents), several pre-processing steps such as stemming, lemmatization, removal of stop-words and

punctuation are followed. Finally, cosine similarity between the documents and query vectors is calculated which is used to return the relevant documents.

Two phases are followed in finding the optimal combination of TF-IDF scoring schemes to locate bugs – Training and Deployment.

### B. Training Phase

The input to the model is Bug Reports along with the files that are modified to fix the bugs [1]. Bug reports are text documents. Apart from the textual description of the bug reports, various other parameters like Version history, similar reports and structure can also be taken into account to further improve the accuracy and relevance of the returned data i.e. the buggy files due to which a corresponding bug report was generated in the first place. In practice, this training data can be obtained from commit logs on platforms such as GitHub, past manual bug-localization or feature-localization efforts in one or more systems [1]. 15 VSM Models are used based on different weighting schemes and performed on the data. Next, in this work, we create a model that computes a weighted sum of the outputs of the individual models. Given the 15 vector space models, VSM 1, VSM 2, . . ., VSM 15 [1].

$$M_{Composite}(b,f) = 15\ \Sigma\ w_i \times VSM_i(b,f)$$

where $M_{Composite}(b,f)$ is the cosine similarity score between b and f, $w_i$ is the weight of the ith VSM and $VSM_i(b,f)$ is the cosine similarity score computed by vector space model $VSM_i$.

***Objective Function**:* After all the compositional VSMs are designed, we must define an objective function to optimize. 2 important criteria are – MAP and MRR values.

MAP or Mean Average Precision if the primary aim of the software designer is to locate all the buggy files after he goes through all bug reports one by one. MRR or Mean Reciprocal Rank is important if the focus is on finding only the first buggy file, after which eventual bugs can be located by manual error solving of this particular bug.

As a bug localization method, both these criteria are equally important for both types of programmers. The program should be able to maximize both of these. Hence, our objective function:

$$F = MAP + MRR$$

Finally, the optimization problem is solved using genetic algorithms (GA) [5] by choosing an appropriate Fitness function and strategy for proportion selection. Here, the Roulette wheel algorithm is used, which randomly selects chromosomes where the probability of a chromosome being selected is proportional to the chromosome's fitness score. [1]

### C. Deployment Phase

During this phase, the training data is read and worked on by our model. For the testing, the input data is the source code files and a bug report to be localized. Our model calculates the cosine similarities and ranks the source files by their cosine ranking. (files which most likely contain the bug)

## III. Spectrum-based fault localization

A common approach to fault localization is spectrum-based as opposed to other methods like Model-based. Spectrum-based localization technique involves the comparison of program traces (called traces) of correct failed executions to pick-out files which actually caused the bug in a testing set. The reason for adapting compositional spectrum models is the same as repeated in the previous sections regarding the lack of 'one model fits all' approach.

As in the previous section, Spectrum-based fault localization proposal also has 2 phases -

### A. Training Phase

The training data, unlike the buggy files in Vector Space Model, is a set of program spectra with the actual locations of the bugs. This can be obtained with the help of previous logs and corrections in the code corresponding to the bug. In the paper, all the 20 association measures and Tarantula [4] and Ochiai [10] are investigated and then finally combined.

Out of all the composition strategies, we combine the 22 versions using a linear model which takes in the weighted sum of individual fault localization measures.

Given 22 measures named as $M_1$, $M_2$, . . ., $M_{22}$ and a program element $e$, the suspiciousness score for $e$ assigned by the composite measure is defined as follows: [2]

$$M_{Composite}(e) = w_1 \times M_1(e) + w_2 \times M_2(e) + ... + w_{22} \times M_{22}(e)$$

Various compositions are formed by taking many permutations of the weights assigned between 0 and 1. But due to the simplicity of discrete search spaces, we use only 7 bits to assign discrete weights.

***Objective Function**:* The criteria to decide the efficiency and accuracy of our solution is associated with the number of source code files are needed to be investigated to find certain numbers of bugs. Therefore, the objective function taken up is the average proportion of program elements that need to be investigated to locate all bugs in a set of programs when lists of program elements sorted in the descending order according to their suspiciousness scores generated by the measure is traversed. [2]

Finally, the optimization problem is solved using search-based Genetic Algorithm (GA) [5] and Simulated Annealing [6].

When implementing the GA, a Simple and a further Enhanced approach is used.

- *Simple:* Randomly selected chromosomes for the next generation.
- *Enhanced:* Here, a threshold is set. If the objective function value of a chromosome is less than k away from that of the best chromosome in the current generation, then it is put in the next generation, otherwise it is randomly passed on. [2]

### B. Deployment Phase

During this phase, the composite methods learn and work on the learning datasets. Suspiciousness scores are assigned after working on the testing program spectra. Finally, a ranked list is returned.

## IV. Related work

A large number of attempts have been made in order to automate the process of localizing bugs. Antoniol et al [7] applied a Vector Space Model along with a probabilistic model to establish connections between the actual source code documents and the textual manuals. Saha et al. improve prior bug localization techniques by considering the structure of bug reports and source code files using *AmaLgam* [8], yet another existing state of the art system developed to achieve even better performance.

Spectrum-based techniques incorporate traces of execution and failure as previously stated. Poshyvanyk et al uses both NLP (Natural Language Processing) and Execution traces to return the buggy files [9]. Our Spectrum-based technique is just an improvement over the existing ones. The Compositional VSM Model is better over spectrum-based because the latter focuses only execution traces. Execution logs for programs are not easily available as training data. Many bug reports do not contain test cases and this makes it even harder to track errors [2].

## V. Future work

The proposed works in both the reviewed papers compose only a handful of Information Retrieval techniques. The plan of action for the future should be to incorporate techniques like BM25, LSA etc. Also, for combining different versions of a technique by varying its parameters (here, weights) should follow another method. In Search-based Fault localization, only Linear Modelling is followed. Other compositional models should also be looked at. Fine tuning of parameters, like weights in our Vector Space Model so that they fit perfectly with our individual training and test set should be the focus in the future.

## VI. Limitations

***Space Limitations*** **–** Generations of test sets, initiating the Vector Space Model for indexing, generating inverted lists for each term corresponding to their document, generating search spaces requires a significant amount of space. Therefore, these models have only been tested against the natural VSM, with standard tf-idf weighting against our composite VSM model. However, to develop a more accurate and well-working model, these restrictions should be surpassed.

***Efficiency*** **–** In both the papers, multiple iterations are involved when reading and working on the test set which could require hours of load time even on higher end machines with Intel processors. However, this is only required to be initiated once on the test data when setting things up in the training phase.

***Learned weights*** **–** GA works on each dataset separately and often, differently. The weights learned can drastically vary from one model to another. However, even if we use non-optimal weights, the learned model could improve on AmaLgam [8], a state-of-the-art approach.

***Validity issue*** **–** We look at the commit logs and make an assumption that the files changed in correspondence with a bug report will cause errors. Although this generalization is often true, a serious logical error may result in many files which previously did not contain the error. This may act as a severe threat to validity as numerous non-buggy files may be returned to the user as being potentially faulty.

## VII. Conclusion

Both these papers offer good insights into the problem of Bug Localization using Information Retrieval methods. Vector Space based and Spectrum based Compositional methods have fairly outperformed other native and singular techniques to do the same. Although these methods are still not safe from some limitations and nuances which might question their practical feasibility in the real world. We have seen some benefits of using Compositional Vector Space Models over Spectrum-based depending on the data they use to train. There

## VIII. References


[1] S. Wang, D. Lo and J. Lawall, "Compositional Vector Space Models for Improved Bug Localization," 2014 IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, 2014, pp. 171-180.

[2] S. Wang, D. Lo, L. Jiang, Lucia, and H. C. Lau, “Search-based fault localization,” in *ASE*, 2011, pp. 556–559

[3] T. Ball, M. Naik, and S. K. Rajamani, "From symptom to cause: Localizing errors in counterexample traces," in POPL, 2003.

[4] J. Jones and M. Harrold, “Empirical evaluation of the tarantula automatic fault-localization technique,” in ASE, 2005.

[5] J.Holland,AdaptationinNaturalandArtificialSystems:AnIntroductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Univ. of Michigan, 1975

[6] S. Kirkpatrick, C. Gelatt, and M. Vecchi, “Optimization by simulated annealing,” Science, vol. 220, pp. 671–680, 1983.

[7] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, “Recovering traceability links between code and documentation,” *IEEE Transactions of Software Engineering*, vol. 28, no. 10, 2002.

[8] S. Wang and D. Lo, “Version history, similar report, and structure: Putting them together for improved bug localization,” in *ICPC*, 2014

[9] D. Poshyvanyk, Y.-G. Gue´he´neuc, A. Marcus, G. Antoniol, and V. Rajlich, “Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval,” *IEEE Transactions of Software Engineering*, vol. 33, no. 6, pp. 420–432, 2007.

[10] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. van Gemund, “A practical evaluation of spectrum-based fault localization,” Journal of Systems and Software, 2009.