Compositional Vector Space Models for Improved Bug Localization

Shaowei Wang, David Lo, and Julia Lawall

School of Information Systems, Singapore Management University

2018-4-29

Lottie Lu, @Computer Graphix Lab, University of Seoul

Contents

1.ABSTRACT

IR techniques using for Bug Localization. **VSM** with standard TF-IDF, **outperform** nine IR techniques. But, **Multiple VSM variants** with different weighting schemes relative performance differs for different software systems.

We propose to compose various **VSM variants**, and a **GA based approach** to explore the space of possible compositions and then evaluated the approach on thousands of bug reports.

2.Background

A: **Bug Localization**

- **Bug Localization** : Link a particular bug report to the files using

  information retrieval(IR) techniques, which is a textual document.

- **Vector Space Model:** (VSM), each document is represented as a **vector of values**. Each value in the vector represents the **weight of a term** in the document. **Assign weights** use the concepts of tf-idf.

Term frequency and Inverse document frequency (tf-idf).

- The standard tf-idf scheme assigns a weight to a term **t** in a document **d** according to the formula:

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

Where t, d, D, tf (t, d), idf (t,D) correspond to a term, a document, a corpus (i.e., a set of documents), the frequency of t in d, and the inverse document frequency of t in D, respectively.

Search-based Algorithms

- Present a particular family of search-based algorithms: genetic algorithms (GA). A GA aims to maximize an objective function. Figure 1 shows the pseudocode of the one we use.
- The time complexity of our GA is given by:

$$O(N_I \times (N_C \times P_C \times O(cross) + N_C \times P_M \times O(mut) + O(sel)))$$

Procedure Genetic Algorithm

Inputs: N_C: Number of chromosomes
N_I: Number of iterations
P_C: Crossover probability
P_M: Mutation probability

Outputs: Best solution found

Method:

1: Let P = Initial population with N_C members
2: Evaluate P's members and find the best solution so far
3: Repeat N_I times
4: P = Selection(P)
5: P = Crossover(P, P_C)
6: P = Mutation(P, P_M)
7: Evaluate P and update the best solution so far
8: **Output** the best solution found

Fig. 1. Genetic Algorithm: Pseudocode

3. Variants of the TF-IDF Weighting Scheme Search-Based Composition Engine

- The tf-idf weight for a term in a document is the product of its **term frequency score** and its **inverse document frequency score**.
- **Inverse document frequency:** (idf) is a measure of whether a term is common or rare in the documents of a corpus.

- There are many **variants of the standard tf-idf** weighting scheme, depending on how the tf and idf are measured.

TABLE II. VARIANTS OF TF AND IDF

Term frequency	
$tf_n(t,d)$ (natural)	$\lvert\{t \mid t \in d\}\rvert$
$tf_l(t,d)$ (logarithm)	$1 + log(tf_n)$
$tf_L(t,d)$ (Log ave)	$\frac{1+log(tf_n(t,d))}{1+log(ave_{t\in d}(tf_n(t,d)))}$
$tf_a(t,d)$ (augmented)	$0.5 + \frac{0.5\times tf_n(t,d)}{max_t(tf_n(t,d))}$
$tf_b(t,d)$ (boolean)	$\begin{cases} 1 & if\ \ tf_n(t,d) > 0 \\ 0 & otherwise \end{cases}$
Document frequency	
$idf_n(t,D)$ (no)	1
$idf_l(t,D)$ (standard)	$log\frac{\lvert D\rvert}{df_t}$
$idf_r(t,D)$ (ratio)	$\max\{0, log\frac{\lvert D\rvert - df_t}{df_t}\}$

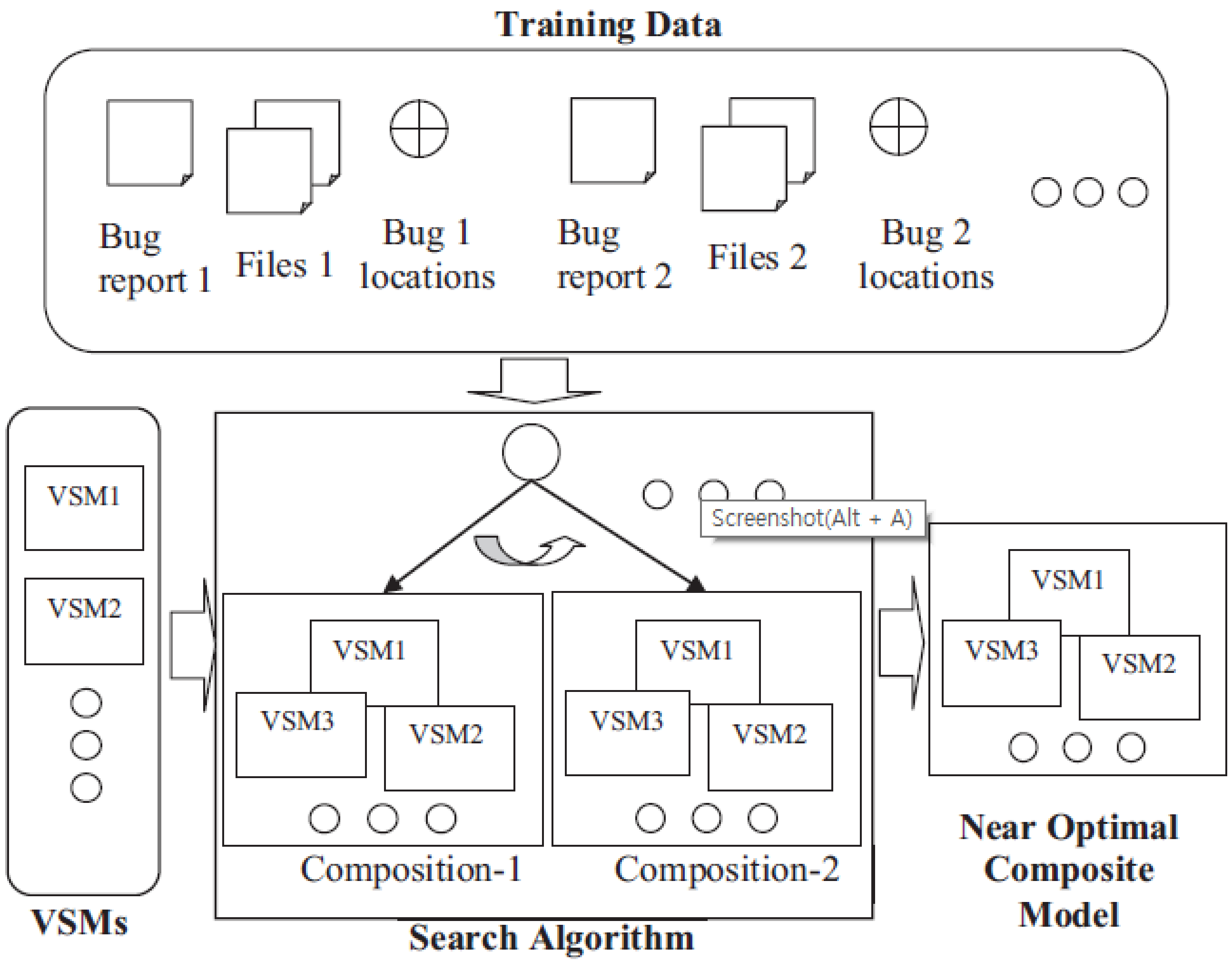
TABLE III. VARIANTS OF THE TF-IDF WEIGHTING SCHEME. THE TF AND IDF VARIANTS ARE DESCRIBED IN TABLE II.

Name	**Equation**
tf_n-idf_n	$tf_n(t,d) \times idf_n(t,D)$
tf_n-idf_l	$tf_n(t,d) \times idf_l(t,D)$
tf_n-idf_r	$tf_n(t,d) \times idf_r(t,D)$
tf_l-idf_n	$tf_l(t,d) \times idf_n(t,D)$
tf_l-idf_l	$tf_l(t,d) \times idf_l(t,D)$
tf_l-idf_r	$tf_l(t,d) \times idf_r(t,D)$
tf_L-idf_n	$tf_L(t,d) \times idf_n(t,D)$
tf_L-idf_l	$tf_L(t,d) \times idf_l(t,D)$
tf_L-idf_r	$tf_L(t,d) \times idf_r(t,D)$
tf_a-idf_n	$tf_a(t,d) \times idf_n(t,D)$
tf_a-idf_l	$tf_a(t,d) \times idf_l(t,D)$
tf_a-idf_r	$tf_a(t,d) \times idf_r(t,D)$
tf_b-idf_n	$tf_b(t,d) \times idf_n(t,D)$
tf_b-idf_l	$tf_b(t,d) \times idf_l(t,D)$
tf_b-idf_r	$tf_b(t,d) \times idf_r(t,D)$

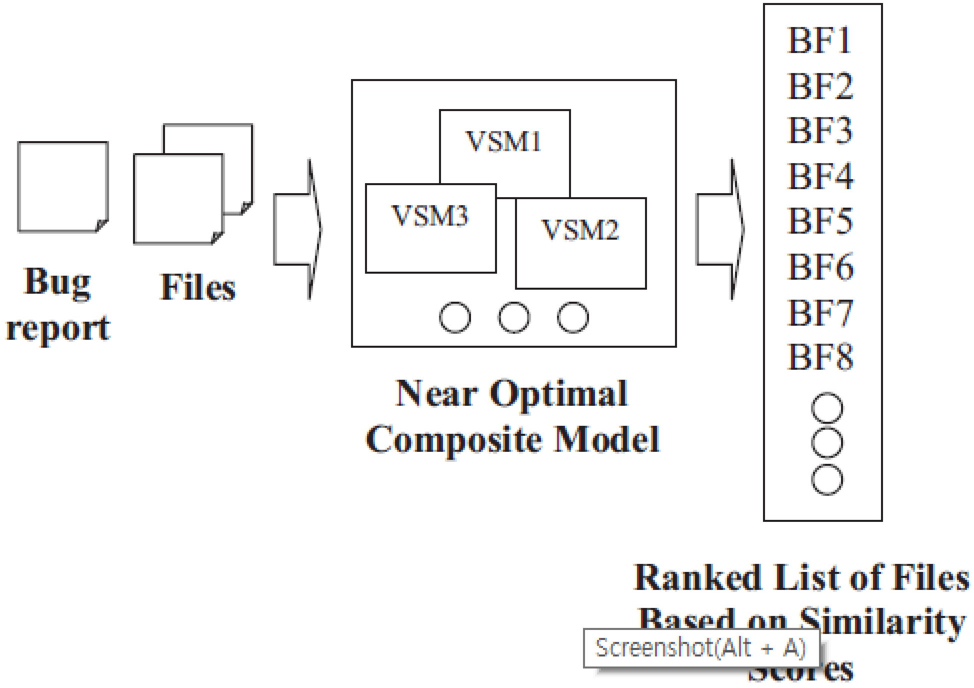
4.Search-Based Composition Engine

- Our search-based bug localization process is composed of two phases:
- **A. Training Phase**
- **B. Deployment Phase**

The two phases are illustrated in Figure 2.



(a) Training Phase



(b) Deployment Phase

Fig. 2. Proposed Framework: Training and Deployment Phase

Objective Functions

- Search algorithms require an objective function to measure how good a candidate solution is. The goal of a genetic algorithm is to **maximize** the value of a given objective function.
- Before defining the objective function for GA, first introduce two evaluation **metrics** that are commonly used to measure the effectiveness of bug localization techniques:

  Mean Average Precision (MAP) 평균 정밀도

  Mean Reciprocal Rank (MRR) 평균 상호 순위

- **Mean Average Precision (MAP)**: MAP emphasizes all of the buggy files instead of only the first one. MAP is computed by taking the mean of the average precision scores across all bug reports.

$$AP = \sum_{k=1}^{M} \frac{P(k) \times pos(k)}{\#buggy\ files},$$

- **Mean Reciprocal Rank (MRR):** The reciprocal rank for a bug report is the reciprocal of the position of the first buggy file in the returned ranked files. MRR is the mean of the reciprocal ranks over a set of bug reports Q

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

5-A. AmaLgam

- AmaLgam which is a state-of-art **bug localization approach** incorporating three components to localize bugs in systems: version history, structure, and similar bug reports:
  - **Version history component**: Input commit logs collected from the version control system and outputs a list of files with their suspiciousness scores. $score_H(b, f)$
  - **Structure component**: Input the source code corpus and a given bug report and returns a list of files with their suspiciousness scores. $score_S(b, f)$
  - **Similar report component:** considers historical bug reports that have already been fixed. $score_R(b, f, B)$

5-B. Compositional Model: *AmaLgam*$_{composite}$

- Present one strategy for combining our compositional VSM with AmaLgam.
- We combine the VSM models with different tf-idf weighting schemes, and three components of AmaLgam as follows.
- Given a bug **report b** and a set of historical fixed bug **reports B**, we compute the suspiciousness score $M_{Composite}$ (b, f) of file f as follows:

$$\sum_{i=1}^{15} w_i \times VSM_i(b,f) + \sum_{J}^{J \in H,R,S} w_j \times score_J(b,f)$$

6.Empirical Evaluation

A. Experimental Setting

- 1) Datasets: We use three datasets containing a total of 3,459 bug reports from three popular **open source projects**, AspectJ, Eclipse, and SWT.

TABLE IV. DATASET DETAILS

Project	Description	Period	#Fixed Bugs	#Source Files
AspectJ	Aspect-oriented extension of Java	07/2002-10/2010	286	6485
Eclipse	Open source IDE	10/2004-03/2011	3075	12863
SWT	Open source wid-get toolkit	10/2004-04/2010	98	484

A. Experimental Setting

- 2) **Effectiveness Calculation**: We use the components of our objective function, MAP and MRR, to evaluate the effectiveness of our solution. We also use **Hit@N.**
- **Hit@N**: This metric calculates the number of bug reports where one of its buggy files appears in the top N ranked files. Given a bug report, if at least one of its relevant files is in the top N ranked files, we consider the report is successfully located.

- **VSM$_{\text{natural}}$**: VSM with the standard tf-idf weighting scheme
- **VSM$_{\text{composite}}$**: Standard tf-idf weighting scheme combining our compositional VSM

- **AmaLgam$_{\text{composite}}$**: Combined the 15 VSMs with the 3 components of AmaLgam
- **AmaLgam$_{\text{natural}}$**: Natural AmaLgam

TABLE V. PERFORMANCE COMPARISONS. $AmaL = AmaLgam$. $AmaL_{compo.} = AmaLgam_{composite}$.

Project	Approach	Hit@1	Hit@5	Hit@10	MAP	MRR
AspectJ	$VSM_{natural}$	25 (8.7%)	43 (15.0%)	65 (22.3%)	0.05	0.13
	$VSM_{compo.}$	33 (11.5%)	55 (19.2%)	67 (23.4%)	0.07	0.16
	$AmaL$	127 (44.4%)	187 (65.4%)	209 (73.1%)	0.33	0.54
	$AmaL_{compo.}$	**145 (50.7%)**	**211 (73.8%)**	**227 (79.4%)**	**0.43**	**0.61**
Eclipse	$VSM_{natural}$	116 (3.8%)	456 (14.8%)	709 (23.1%)	0.01	0.01
	$VSM_{compo.}$	116 (3.8%)	544 (17.7%)	845 (27.5%)	0.01	0.01
	$AmaL$	1060 (34.5%)	1775 (57.7%)	2059 (67.0%)	0.35	0.45
	$AmaL_{compo.}$	**1108 (36.1%)**	**1905 (62.0%)**	**2187 (71.2%)**	**0.39**	**0.48**
SWT	$VSM_{natural}$	12(50.7%)	37 (73.8%)	49 (79.4%)	0.21	0.24
	$VSM_{compo.}$	14 (50.7%)	40 (73.8%)	53 (79.4%)	0.23	0.26
	$AmaL$	61 (62.2%)	80 (81.6%)	**88 (89.8%)**	0.62	**0.71**
	$AmaL_{compo.}$	**62 (63.2%)**	**83 (82.6%)**	**88 (89.8%)**	**0.63**	**0.71**

Conclusion

- In this paper, we build a solution that combines 15 VSMs with different tf-idf weighting schemes into an improved **composite model**, constructed using a **genetic algorithm**.
- We have evaluated our approach on 3,459 bug reports from AspectJ, Eclipse, and SWT and demonstrate that our approach can achieve **better performance**. Compared with $VSM_{natural}$, averaging across the 3 datasets, our approach, VSMcomposite, **improves** $VSM_{natural}$ in terms of Hit@5, MAP, and MRR by 18.4%, 20.6%, and 10.5% respectively.
- We have also combined the 15 VSMs with the 3 components of AmaLgam, which is the state-of-the-art bug localization technique. Compared with AmaLgam, averaging across the 3 datasets, $AmaLgam_{composite}$ can **improve** AmaLgam in terms of Hit@5, MAP, and MRR by 8.0%, 14.4% and 6.5% respectively.

THANK YOU