A Large-Scale Comparative Evaluation of IR-based Tools for Bug Localization

SHAYAN AKBAR
AVI KAK
PURDUE UNIVERSITY
[AN MSR 2020 PRESENTATION]
We focus on the problem of IR-based bug localization

- Given a large corpus of source code files belonging to a software repository and a bug report, find a list of source code files ranked based on their relevancy to the bug report
Contents

• Motivation / Introduction
• The three generations of IR-based bug localization
• Eight retrieval algorithms selected from the three generations to study
• Bugzbook --- A large, diverse bug localization dataset
• Results
• Conclusion
Why yet another study on bug localization?

• **Because previous studies** …
  • mostly relied only on Java software repositories
  • are based on datasets with only few thousand bug reports
  • conducted experiments only on very few software projects

• **We created Bugzbook --- a new large and diverse bug localization dataset containing over 20000 bug reports**
  • These bug reports belong to 29 different software projects
  • The projects belong to Java, C/C++ and Python programming languages
We experimented with past bug localization algorithms on Bugzbook

- We divided the past 15 years of studies in IR-based bug localization into three generations
- Each generation tells a story of what happened in the field of bug localization during that generation
- We compiled a list of the prominent papers published in each generation
- From all the three generations we selected eight algorithms and conducted experiments on Bugzbook dataset to evaluate their relative performances
The three generations of IR-based bug localization (spanning studies from last 15 years)
1\textsuperscript{st} generation [2004 – 2011]

- Most of the early works in bug localization relied on using traditional BoW (Bag of Words) based IR tools to rank files based on their relevancy to the bug report

- Tradition BoW IR techniques:
  - UM (Unigram Model)
  - TFIDF (Term Frequency Inverse Document Frequency)
  - DLM (Dirichlet Language Model)
  - VSM (Vector Space Model)
  - LSI (Latent Semantic Indexing)
  - ...
2nd generation [2011 – 2016]

• **Software-centric information modelled in bug localization systems**

• Information derived from:
  • bug report history
  • version history
  • structure of bug reports
  • structure of code files
  • ...

• BugLocator, BLUiR, DHbpD, BRTracer, BLIA, LOBSTER, and Amalgam are some of the tools developed during this generation
• **Term-term proximity, order and semantic relationships modelled in bug localization systems**

  • **Proximity**: Code files that contain bug report terms in similar proximities as in the bug report itself are more likely to be relevant to the bug report.
  
  • **Order**: Code files should be ranked higher in the list if they contain bug report terms in the same order as they appear in the bug report text.
  
  • **Semantics**: In addition to exact matching between terms in the code files and bug report, terms should also be matched based on semantics.
From the three generations we selected eight retrieval algorithms to study

1. **TFIDF** (Term Frequency Inverse Document Frequency)
2. **DLM** (Dirichlet Language Model)
3. **BugLocator**
4. **BLUiR** (Bug Localization Using Information Retrieval)
5. **MRF-SD** (Markov Random Fields – Sequential Dependence)
6. **MRF-FD** (Markov Random Fields – Full Dependence)
7. **PWSM** (Per Word Semantic Model)
8. **SCOR** (Source Code Retrieval with Semantics and Order)
1st Generation Tools (Simple BoW Models)

- **TFIDF**
  - works by combining frequencies of query terms in the file (TF) and the inverse document frequencies of query terms in the corpus (IDF) to determine the relevance of a file to a query

\[
\text{score}_{\text{tfidf}}(f, Q) \propto \text{tf}(q_i, f) \times \text{idf}(q_i)
\]

- **DLM**
  - a probabilistic model that estimates a smoothed first order probability distribution of the query terms in the file to produce the relevance score for a file to a query

\[
\text{score}_{\text{dlm}}(f, Q) \propto \text{tf}(q_i, f) \times \text{cf}(q_i)
\]
2\textsuperscript{nd} Generation Tools (Software-centric information)

- **BugLocator**
  - takes into account the history of the past bug reports and leverages bug reports that have been previously fixed to improve bug localization

- **BLUiR**
  - extracts code entities such as classes, methods, variables, and comments from code files to help in localizing a buggy file
3rd Generation Tools (Modelling Proximity and Order)

- **MRF-SD**
  - measures the probability distribution of the frequencies of pairs of consecutively occurring query terms appearing in the file to compute the relevance score for the file to a query
  
  \[
  \text{score}_{sd}(f, Q) \propto tf(q_{i}q_{i+1}, f) \times cf(q_{i}q_{i+1})
  \]

- **MRF-FD**
  - is a term-term dependency model that considers frequencies of all pairs of query terms appearing in the file to determine the relevance of the file to the query
  
  \[
  \text{score}_{fd}(f, Q) \propto tf(q_{i}q_{j}, f) \times cf(q_{i}q_{j})
  \]
3rd Generation Tools (Modeling semantics)

- In order to discuss how to model semantics in retrieval process first we need to discuss semantic word embeddings
3rd Generation Tools
(word2vec for constructing semantic word embeddings)

word2vec neural network

Millions of source code files

Semantic word vectors

SCOR Word Embeddings

https://engineering.purdue.edu/RVL/SCOR_WordEmbeddings/
3rd Generation Tools (Modelling Semantics as in SCOR)

**Query Embeddings**
- Each row is a dense N-dim vector for a query term learned using word2vec

**File Embeddings**
- Each row is a dense N-dim vector for a file term learned using word2vec

**Match Layer 1 (ML1)**
- Each cell is cosine similarity between a query term \( q_i \) and a file term \( t_j \)
- Convolve with the \( 2 \times 2 \) kernel

**Match Layer 2 (ML2)**
- Each cell represents similarity between 2 consecutive query terms \( q_i, q_{i+1} \) and file terms \( t_j, t_{j+1} \)

**Best-matching vector**
- Each cell represents cosine similarity between 2 consecutive query terms and their 2 consecutive best matched file terms

**Best-of-best vector**
- Retain cosine similarities for top \( \xi_1 \) query terms pairs from best-matching vector

**score_{pirm}(f, Q)**
- Relevance score based on single term

**score_{ordm}(f, Q)**
- Relevance score by comparing 2 consecutive query and file terms
3\textsuperscript{rd} Generation Tools (Modelling Semantics)

- **PWSM:**
  - uses embeddings derived from word2vec algorithm to model term-term contextual semantic relationships in retrieval process
  \[
  \text{score}_{\text{pws}m}(f,Q) = \alpha \cdot \text{score}_{\text{dlm}}(f,Q) + \beta \cdot \text{score}_{\text{pws}m}(f,Q)
  \]

- **SCOR:**
  - combines MRF based term-term dependency modeling with semantic word embeddings as made possible by word2vec algorithm to improve bug localization
  \[
  \text{score}_{\text{scor}}(f,Q) = \alpha \cdot \text{score}_{\text{dlm}}(f,Q) + \beta \cdot \text{score}_{\text{sd}}(f,Q) + \\
  \mu \cdot \text{score}_{\text{pws}m}(f,Q) + \Omega \cdot \text{score}_{\text{ords}m}(f,Q)
  \]
Components built into each of the eight algorithms

<table>
<thead>
<tr>
<th></th>
<th>TFIDF</th>
<th>DLM</th>
<th>BugLocator</th>
<th>BLUiR</th>
<th>MRF SD</th>
<th>MRF FD</th>
<th>PWSM</th>
<th>SCOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>BoW</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Order</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Semantics</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Stack Trace</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Structure</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Bug History</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
We experimented with these eight algorithms on Bugzbook (features of Bugzbook dataset)

- **Bugzbook contains over 20000 bug reports from 29 projects belonging to Java, C/C++, and Python projects**
- **We processed over 4 million source code files belonging to all the 29 projects**
  - Out of 29 projects, 23 projects belong to Apache community (Ambari, Spark, Camel, …)
  - We downloaded Apache bug reports from well-managed Jira platform
  - Also present in Bugzbook are bug reports from other large-scale projects (OpenCV, Pandas, …)
  - The bug reports for non-Apache projects were downloaded from GitHub
How Bugzbook compare with other datasets?

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of projects</th>
<th>Number of bug reports</th>
</tr>
</thead>
<tbody>
<tr>
<td>moreBugs</td>
<td>2</td>
<td>~400</td>
</tr>
<tr>
<td>BUGLinks</td>
<td>2</td>
<td>~4000</td>
</tr>
<tr>
<td>iBUGS</td>
<td>3</td>
<td>~400</td>
</tr>
<tr>
<td>Bench4BL</td>
<td>46</td>
<td>~10000</td>
</tr>
<tr>
<td>Bugzbook</td>
<td>29</td>
<td>~20000</td>
</tr>
</tbody>
</table>
## Bugzbook stats
(number of bug reports)

### Apache Java projects

<table>
<thead>
<tr>
<th>Project</th>
<th>#bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ambari</td>
<td>2253</td>
</tr>
<tr>
<td>Bigtop</td>
<td>5</td>
</tr>
<tr>
<td>Camel</td>
<td>2308</td>
</tr>
<tr>
<td>Cassandra</td>
<td>514</td>
</tr>
<tr>
<td>CXF</td>
<td>1795</td>
</tr>
<tr>
<td>Drill</td>
<td>800</td>
</tr>
<tr>
<td>Hbase</td>
<td>2476</td>
</tr>
<tr>
<td>Hive</td>
<td>2221</td>
</tr>
<tr>
<td>JCR</td>
<td>457</td>
</tr>
<tr>
<td>Karaf</td>
<td>390</td>
</tr>
<tr>
<td>Mahout</td>
<td>162</td>
</tr>
<tr>
<td>Math</td>
<td>17</td>
</tr>
<tr>
<td>OpenNLP</td>
<td>84</td>
</tr>
<tr>
<td>PDFBox</td>
<td>1163</td>
</tr>
<tr>
<td>PIG</td>
<td>47</td>
</tr>
<tr>
<td>SOLR</td>
<td>471</td>
</tr>
</tbody>
</table>

### Other Java projects

<table>
<thead>
<tr>
<th>Project</th>
<th>#bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark</td>
<td>185</td>
</tr>
<tr>
<td>Sqoop</td>
<td>201</td>
</tr>
<tr>
<td>Tez</td>
<td>177</td>
</tr>
<tr>
<td>Tika</td>
<td>183</td>
</tr>
<tr>
<td>Wicket</td>
<td>567</td>
</tr>
<tr>
<td>WW</td>
<td>87</td>
</tr>
<tr>
<td>Zookeeper</td>
<td>20</td>
</tr>
</tbody>
</table>

### C/C++, Python projects

<table>
<thead>
<tr>
<th>Project</th>
<th>#bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrome</td>
<td>147</td>
</tr>
<tr>
<td>OpenCV</td>
<td>8</td>
</tr>
<tr>
<td>Pandas</td>
<td>179</td>
</tr>
<tr>
<td>Tensorflow</td>
<td>10</td>
</tr>
</tbody>
</table>

### Total

- #projects: 29
- #bugs: 21253

Eclipse and Chrome bug reports taken from BUGLinks dataset
AspectJ bug reports taken from iBUGS dataset
Bugzbook construction

- Gather raw bug reports and code files
  Download bug reports from Jira and GitHub, and source code files from project archives

- Filter raw bug reports (remove duplicates, etc.)
  Remove bug reports with duplicate tags. Also, remove reports that are not tagged as “bug”.

- Link bug reports with respective code files
  Use commits to identify which bugs it fixes. Also, use modified associated with the commits as relevant files for the respective bug.

- Match bug reports with respective versions
  Bug reports come with a tag that tells which version was affected by the bug

- Manual verification of dataset
  Verify randomly chosen bugs for each project. Check if bug ID, title, description, fixed files are correctly recorded in Bugzbook.
Results on Bugzbook dataset
Highlights of the results

• **Experiment 1** *(Performance of retrieval algorithms belonging to different generations)*
  • Results on Java only projects
  • Results on C/C++ and Python projects
  • Results on all projects

• **Experiment 2** *(Effect of semantic word embeddings)*
  • Can we cross-utilize word embeddings across different programming languages?
  • Does changing the size of word vectors and the word embedding algorithm affect retrieval?
Highlights of the results (contd.)

• 3rd generation tools are the most effective in terms of retrieval precision

• SCOR word embeddings (trained on Java language) can be used for searching in C/C++ and Python projects

• The size (500, 1000, 1500 dimensions) and type (word2vec, GloVe, FastText) of word embeddings do not affect the performance of retrieval
Results for Java projects

SCOR, a third generation tool, outperforms all other algorithms.
Results for C/C++ and Python projects

SCOR and MRF SD, both third generation tools, outperforms all other algorithms
Results for all projects

SCOR, a third generation tool, outperforms all other algorithms.
Cross-utilization of word embeddings

• SCOR word embeddings were trained on Java-only projects

• The following 5 Java projects present in Bugzbook dataset were not present in the SCOR dataset:
  • AspectJ, Bigtop, OpenNLP, PDFBox, and Drill

• Therefore, the word embedding algorithm was not trained on these 5 projects

• However, the bug localization precision on these 5 datasets increases when semantic word embeddings based retrieval algorithms --- PWSM and SCOR --- are used for retrieval.

• Therefore, the SCOR word embeddings are generic enough to be applied for retrieval in new projects

• We also observe the same results on the following C/C++ and Python projects
  • Chrome, OpenCV, Pandas, and Tensorflow
Effect of changing word vector size and word embedding algorithm

- We observe that the word vector size does not affect the retrieval precision of SCOR when tested on 4000 bug reports of Eclipse dataset

<table>
<thead>
<tr>
<th>SCOR500</th>
<th>SCOR1000</th>
<th>SCOR1500</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3191</td>
<td>0.3204</td>
<td>0.3193</td>
</tr>
</tbody>
</table>

SCOR500 means word vectors having 500 dimensions used for retrieval

- We also trained GloVe and FastText (the two semantic word embedding algorithms that compete with word2vec) on the same SCOR dataset, and used the resulting word vectors for retrieval with SCOR

- We notice that the change in word embedding algorithm does not affect the retrieval performance of SCOR

<table>
<thead>
<tr>
<th>SCOR (word2vec)</th>
<th>SCOR (glove)</th>
<th>SCOR (fasttext)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3204</td>
<td>0.3192</td>
<td>0.3182</td>
</tr>
</tbody>
</table>

SCOR (glove) means GloVe word embeddings (size=500) were used for retrieval
Conclusion

• The third generation tools are far superior in performance than the first and second generation tools

• SCOR semantic word embeddings are very generic:
  • They can be used to perform bug localization in those Java projects on word2vec was not trained on
  • Also, they are very effective for bug localization in non-Java projects

• It’s time to merge techniques from 2nd and 3rd generations to develop hybrid approaches for IR-based bug localization
Thank you …

Questions?

Dataset will be made available at:

https://engineering.purdue.edu/RVL/Bugzbook/