Semantic Clone Detection: Can Source Code Comments Help?

Akash Ghosh
Tandy School of Computer Science
University of Tulsa
akashghosh@utulsa.edu

Sandeep Kaur Kuttal
Tandy School of Computer Science
University of Tulsa
sandeep-kuttal@utulsa.edu

***Abstract*—Programmers reuse code to increase their productivity, which leads to large fragments of duplicate or near-duplicate code in the code base. The current code clone detection techniques for finding semantic clones utilize Program Dependency Graphs (PDG), which are expensive and resource-intensive. PDG and other clone detection techniques utilize code and have completely ignored the comments - due to ambiguity of English language, but in terms of program comprehension, comments carry the important domain knowledge. We empirically evaluated the accuracy of detecting clones with both code and comments on a JHotDraw package. Results show that detecting code clones in the presence of comments, Latent Dirichlet Allocation (LDA), gave 84% precision and 94% recall, while in the presence of a PDG, using GRAPLE, we got 55% precision and 29% recall. These results indicate that comments can be used to find semantic clones. We recommend utilizing comments with LDA to find clones at the file level and code with PDG for finding clones at the function level. These findings necessitate a need to reexamine the assumptions regarding semantic clone detection techniques.**

I. Introduction

"Don't reinvent the wheel, just realign it." A common practice for programmers to increase their productivity is copying an existing piece of code and changing it to suit a new context or problem. This reuse mechanism promotes large fragments of duplicate or near-duplicate code in the code base [2]. These duplicates are called code clones. Research shows that about 7% to 23% of software systems contain duplicated codes [9]–[12].

In software engineering, many techniques [1] have been proposed to detect code clones based on token similarity (e.g., CCFinder [18], CloneMiner [19] and CloneDetective [17]), Abstract Syntax Tree(e.g., CloneDR [13], Deckard [14]) and Program Dependency Graph (e.g., [3], [6], [7], [15], [16]). One of the most challenging types of clones to find are semantic clones - code fragments that are functionally similar but may be syntactically different. Techniques based on Program Dependency Graph (PDG) are one of the most notable mechanisms to detect semantic code clones [3] as it abstracts many arbitrary syntactic decisions that a programmer made while constructing a function. However, PDG-based techniques are computationally expensive, as they require resource-intensive operations to detect the clones.

Current code clone detection techniques do not include source comments. From a program comprehension point of view, these comments carry important domain knowledge and also might assist other programmers to understand the code. One of the reasons, to ignore code comments is due to the ambiguity of the English language. For humans, it is easy to comprehend the similarity or difference between words or topics, but a machine may treat the words differently. However, with recent advancement in machine learning and natural language processing tools we hope to detect clone sets by using LDA.

In this paper, we investigated:

- **RQ1:** Does the use of comments help in detecting semantic clones in the code base?
- **RQ2:** Does a PDG based technique, which just uses code for detection of semantic clones, perform equivalently to an LDA based technique, which uses comments?

II. Methodology

A. Dataset

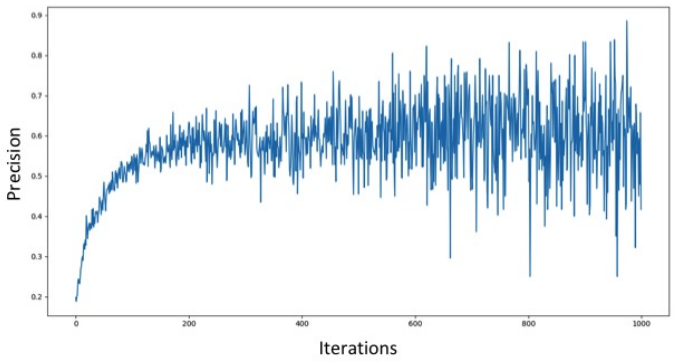
In this work, JHotDraw-a java package-has been used which contains 310 java source files with 27kLOC. JHotDraw [8] has been widely used in clone detection studies [5].

B. Procedure

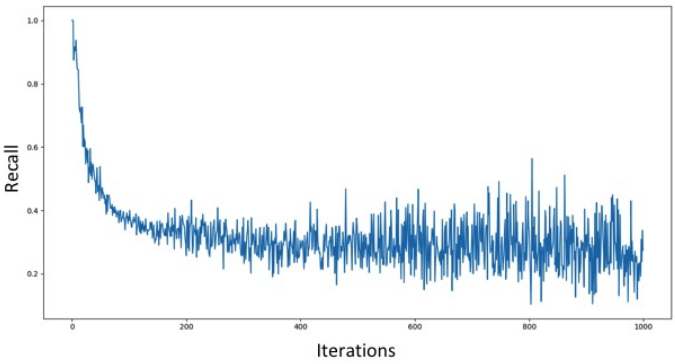
1) PDG: GRAPLE [3], [4], an existing PDG based clone detection tool, was used to identify clones within the Java package and JPDG to create an undirected graph (vertex-edge, veg) for the whole package. The tool generates a JSON file with edges and vertices in the form of a dictionary. This veg file was then used as an argument along with min-support, sample-size, min-vertices, and selection probability for GRAPLE. The clone sets were generated with and without the selection probabilities with support=5, sample-size=100, and min-vertices=8.

2) LDA: Python 3.6 and Regex Expression were used to extract the comments from the source files. All comments were included, except the copyright comments, since it does not contain any information related to the functionality of the source code. Once the comments were extracted, it was normalized by cleaning the stop words and the punctuations. With this normalized texts, a dictionary was created which was used to create the Doc-Term matrix. The LDA model was trained using the corpus and dictionary mentioned above. Then the passes and iterations were set to a specified value. The

978-1-5386-4235-1/18/ $31.00 © 2018 IEEE



(a) Precision.



(b) Recall.

Fig. 1. Precision and Recall.

comment files were passed as an argument to the model to generate the relative topics. Once all the clone sets were generated, we calculated the precision - $|D_{reported} \cap D_{actual}| \div |D_{reported}|$ and recall - $|D_{reported} \cap D_{actual}| \div |D_{actual}|$.

$D_{reported}$ is the set of multi-sets reported by the model and D_{actual} is the ground truth which contains 52 clone sets built manually in 45 hours.

III. Results

A. RQ1: Can code comment help?

To understand whether comments can assist in detecting code clones, the model was trained and the outputs (clone sets) were analyzed in two different ways.

1) Way 1: The LDA model was trained using the files as the corpus. With topic limit set to 100, we were able to extract 66 clone sets (274 files). The precision and recall found are mentioned in Table 1.

2) Way 2: To understand how the clone sets varied in terms of precision and recall, the model was trained over a range of 1 to 1000 topics. The parameters were set at 1000 iterations with 50 passes. Table 1 shows the best precision found with topic set to 975, which generated 7 clone sets with 21 files. From Fig:1 it is evident that with increased iterations fewer clone sets were found. Also, as the number of iterations increased the precision increased as well with a global maxima at 975. However, the recall decreased.

To further add, the best clone sets in comparison with the ground truth were the clones sets generated by topic number 975. The clone sets were manually analyzed to check the authenticity, it was observed that the matched clone sets i.e $|D_{reported} \cap D_{actual}|$ have high similarities in terms of object or instance creation.

B. RQ2: PDG vs LDA: code vs comments?

Further, to compare a PDG based technique with LDA, we used GRAPLE [3]. We evaluated GRAPLE with and without the selection probability P_i, the later was used to avoid the "Curse of Dimensionality".

1) Without Pr: In this evaluation technique the sample-size were varied multiple times, setting it from 20 to 200, but in most of the cases very small increase in clone sets were observed. Precision and recall mostly varied between 50% to 55%. Table 1 depicts the precision and recall for sample-size 100 with min-vertices 8 and support set to 5.

2) With Pr: Using the selection probabilities and with the above mentioned specification, we generated 80 clone sets. Table 1 shows the 22 clone sets were found while using selection probabilities, and 17 clone sets were found without using selection probabilities. Comprehensive clone sets were reported by the model with probability in expense of 30 hours and 74 GB of memory. However, the model without probability reported 17 clone sets in 4.5 secs and consumed 481.5 MB. Moreover, 16 out of 17 clone sets which were reported by model without probability were also reported by the model with probability.

Evidently, the precision and recall for LDA are better than PDG. Upon analyzing the clone sets returned by PDG and LDA, it was observed that LDA was able to find more clone sets. In addition, LDA quickly found the clones based on similar comments compared to PDG, which took hours. Our dataset consists of 27 KLOC, so for such packages PDG based techniques can perform decently, but for larger sizes as noticed by [3] they can deplete the resources.

TABLE I
Precision and Recall for LDA and PDG.

		#clonesets	Recall	Precision
LDA	***Way 1***	66	94.86	84.21
	Way 2	7	28.61	88.57
PDG	***Without Pr.***	17	27.84	52.94
	With Pr.	22	28.7	55.39

IV. Conclusion

Our results show that comments can be utilized with LDA and are equivalent to sophisticated PDG based techniques. One approach would be using comments with LDA to detect clone sets at the file level, as this process is less resource-intensive, and applying PDG based code detection techniques at the function level. Our study provides the very first evidence that comments which are underrated in clone detection research can be utilized effectively.

REFERENCES

[1] C.K. Roy, M.F. Zibran, and R. Koschke, "The vision of software clone management: Past, present, and future (Keynote paper)", in Proceedings of Software Maintenance, Re-engineering and Reverse Engineering, pp.18-33, 2014.

[2] J. Howard Johnson, "Visualizing textual redundancy in legacy source", in Proceedings of Centre for Advanced Studies on Collaborative, pp.32, 1994.

[3] TAD Henderson and A Podgurski, "Sampling code clones from program dependence graphs with GRAPLE, in Proceedings of International Workshop on Software Analytics, pg 47-53, 2016.

[4] H. Cheng, X. Yan, and J. Han, "Mining Graph Patterns. In Frequent Pattern Mining", in Managing and Mining Graph Data, pp.307-338, 2010.

[5] Y. Lin, Z. Xing, Y. Xue, Y. Liu, X. Peng, J. Sun, and W. Zhao, "Detecting differences across multiple instances of code clones", in Proceedings of International Conference on Software Engineering, pp.164-174, 2014.

[6] J. Krinke, "Identifying similar code with program dependence graphs", in Proceedings of Working Conference on Reverse Engineering, pp.301-309, 2001.

[7] R. Komondoor and S. Horwitz, "Using Slicing to Identify Duplication in Source Code", in Proceedings of International Symposium on Static Analysis, pp.40-56, 2001.

[8] JHotDraw: http://www.jhotdraw.org/

[9] B. Baker, "On Finding Duplication and Near-Duplication in Large Software Systems", in Proceedings of Working Conference on Reverse Engineering, pp.86-95, 1995.

[10] I. Baxter, A. Yahin, L. Moura and M. Anna, "Clone Detection Using Abstract Syntax Trees", in Proceedings of International Conference on Software Maintenance, pp.368-377, 1998.

[11] C. Kapser and M. Godfrey, "Supporting the Analysis of Clones in Software Systems: A Case Study", Journal of Software Maintenance and Evolution: Research and Practice - IEEE International Conference on Software Maintenance, Vol.18 (2), pp.61-82, 2006.

[12] J. Mayrand, C. Leblanc and E. Merlo, "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics", in Proceedings of Proceedings of International Conference on Software Maintenance, pp.244-253, 1996.

[13] F. Al-Omari, I. Keivanloo, C. K. Roy, and J. Rilling, "Detecting clones across microsoft .net programming languages", in Proceedings of Working Conference on Reverse Engineering, pp.405-414, 2012.

[14] S. Bazrafshan, and R. Koschke, "An empirical study of clone removals", in Proceedings of International Conference Software Maintenance, pp.50-59, 2013.

[15] D. Chatterji, J. C. Carver, and N. A. Kraft, "Cloning: The need to understand developer intent", in International Workshop on Software Clones, pp. 14-15, 2013.

[16] J. R. Cordy, "Comprehending reality: Practical barriers to industrial adoption of software maintenance automation", in International Workshop on Program Comprehension, pp.196-206, 2003.

[17] N. Bettenburg, W. Shang, W. Ibrahim, B. Adams, Y. Zou, and A. Hassan, "An empirical study on inconsistent changes to code clones at the release level", in Working Conference on Reverse Engineering, pp.760-776, 2012.

[18] S. Bouktif, G. Antoniol, M. Neteler, and E. Merlo, "A novel approach to optimize clone refactoring activity", in Proceedings of Conference on Genetic and Evolutionary Computation, pp.1885-1892, 2006.

[19] E. Adar and M. Kim, "SoftGUESS: Visualization and exploration of code clones in context", in Proceedings of International Conference on Software Engineering, pp.762-766, 2007.