Investigating Vector-based Detection of Code Clones Using BigCloneBench

Kazuki Yokoi†, Eunjong Choi†, Norihiro Yoshida† and Katsuro Inoue†

†Osaka University, Japan, {k-yokoi, inoue}@ist.osaka-u.ac.jp
‡Nara Institute of Science and Technology, Japan, choi@is.naist.jp
§Nagoya University, Japan, yoshida@ctrl.jp

Abstract—In a vector-based approach to detecting code clones from source code, all code fragments in the source are mapped to a vector space and then code fragments are detected as code clones if they are neighbors in the vector space. So far, our research group has developed a vector-based approach using TF-IDF and cosine similarity. For the improvement of the vector-based approach, we preliminarily investigated what kind of vectorization algorithms and similarity measurements are effective in terms of recall and detection time. In this paper, we present preliminary investigation results using BigCloneBench, a large-scale code clone benchmark.

Index Terms—code clone, software maintenance, information retrieval

I. MOTIVATION

Programmers often copy and paste code fragments so that they can reuse existing code fragments. This causes code clones (i.e., code fragments that are identical or similar code fragments to each other). Generally, a code clone is regarded as a factor that hinders software maintainability [1]. However, it is difficult for developers to manually identify all code clones from large-scale software systems.

To alleviate this problem, a multitude of code clone detection approaches have been proposed [2], [3], [4]. One of the representative approaches for detecting code clones is a vector-based approach. The advantage of this approach is that it detects syntactically dissimilar code clones. It also can detect clones at high speed by clustering similar code fragments. Our research group proposed a vector-based code clone approach that detects similar functions in the source code namely FLCCFinder (Function Level Code Clone Finder) [5]. FLCCFinder generates a feature vector from each function based on the occurrence of identifiers and reserved keywords using TF-IDF (Term Frequency/Inverse Document Frequency). It then clusters the generated vectors by means of locality-sensitive hashing. Finally, it detects clones based on the similarities between each pair of feature vectors using cosine similarity. However, FLCCFinder has possibilities that the generated vectors are high-dimensional vectors and vectors of polysemy and synonym are missed.

To improve FLCCFinder, we preliminary investigated what kind of vectorization algorithms and distance measures are effective in terms of recall and detection time. In this paper, we present preliminary investigation results using BigCloneBench [6], a large-scale code clone benchmark.

II. PRELIMINARILY INVESTIGATION

A. Vectorisation Algorithms and Similarity Measurements

Algorithms to vectorize documents are frequently used in the field of information retrieval. Note that, in this study, each function in the source code was regarded as a document and vectorized. We selected seven vectorization algorithms, namely BoW (Bag-of-Words), TF-IDF, LSI (Latent Semantic Indexing), LDA (Latent Dirichlet Allocation), Word2Vec [7], [8], FastText [9], Doc2Vec [10]. Among these algorithms, BoW and TF-IDF have a high possibility that generate high-dimensional vectors, whereas LSI and LDA represent documents as low-dimensional vectors by adopting dimension compression. Word2Vec and FastText generate vectors that can express the meaning of words using deep neural networks. In this study, we defined the average vectors of Word2Vec as WV-avg and the average vectors of FastText as FT-avg respectively. Doc2Vec extends Word2Vec to entire documents. Furthermore, to measure the similarity of vectors, we choose cosine similarity and WMD (Word Mover's Distance) [11]. In this study, the parameters for each algorithm are determined based on the default values of gensim [12], a python library that is used for implementing the algorithms in this study.

B. Research Questions and Results

To investigate the impact of vectorization algorithms and similarity measurements for detecting code clones, we set up two research questions (RQs) as follows.

1) RQ1: Does the recall of code clone detection vary with vectorization algorithms?: We set up this RQ to find out vectorization algorithms that can accurately detect code clones. To answer this RQ, we applied an approach using each selected vectorization algorithm with FLCCFinder to 25,000 open-source projects in BigCloneBench, consisting of 2.3 million Java source files (365MLOC), and compared the recall of each approach. In this study, we set the threshold of 0.9 for cosine similarity because this threshold achieved high accuracy in the previous study [5]. Note that we only used cosine similarity to measure the similarity of vectors because it takes approximately three months for WMD to detect code clones from target projects. The results of the comparison are shown in Table I. With respect to the definition of each clone type, please refer to [6].

https://radimrehurek.com/gensim/
2) RQ2: Does the selection of vectorization algorithms and distance scale affect detection speed?: We set up this RQ to identify the algorithms that detect code clones with high speed. To investigate the speed of calculation vector, we built 1MLOC dataset by randomly selecting files from BigCloneBench. We then measured the time to generate the vectors (i.e., generation time) and the time to calculate the similarity between one function and all other functions in the source code (i.e., similarity calculation time) with different algorithms and similarity measurements with FLCCFinder. The results are shown in Table II.

III. DISCUSSION

Regarding RQ1, T2 clones were detected with a recall of more than 0.9 in LSI, Doc2Vec, WV-avg and FT-avg. From these results, we can assume that T2 clones can be accurately detected by using dimensional compression and deep neural networks. Surprisingly, the recall of TF-IDF was lower than BoW except for T1 clones. This is because the weights of identifiers tend to be higher than reserved words in TF-IDF, and these code clones contain different identifier names. Due to the difference in the dimensional compression, LSI detects T2 clones with high recall, meanwhile LDA detects MT3 clones with high recall. Doc2Vec had relatively high recall in detecting T2 clones, however, it was not able to detect many Type-3 code clones such as ST3 and MT3 clones. WV-avg and FT-avg had the highest recall, but there is a possibility that these algorithms achieved the highest recall due to the large numbers of detected clones.

Regarding RQ2, BoW is the fastest in the generation time followed by LSI and TF-IDF. Meanwhile, algorithms using deep neural networks such as FastText are slower than other algorithms. Word2Vec achieved the highest speed among algorithms using deep neural networks. In the algorithms using dimensional compression, LDA is the slowest, whereas LSI is the fastest. In the algorithms using deep neural networks, WV-avg is the fastest. FT-avg is slower than WV-avg (Table I) even though they achieved the the same recall. In the similarity calculation time, algorithms using dimension compression showed the highest speed. On the other hand, WMD is much slower than the cosine similarity. In particular, the similar calculation time calculates the time to measures the similarity between one function and all other functions, that is, calculation of $O(n^2)$. However, actual clone detection requires the similar calculation of all functions and all functions, that is, calculation of $O(n^2)$. This revealed that it is not practical to use WMD for detecting code clones.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Numbers JP25220003, JP18H04094 and JP16K16034.

REFERENCES