Review for *Cross-Language Code Search using Static and Dynamic Analyses*

Hongchen Cao
Shanghaitech University
Shanghai, China
caohch1@shanghaitech.edu.cn

I. Paper Summary

The paper purpose a cross-language technique called Code-to-Code Search Across Languages (COSAL) that uses both static and dynamic analyses to identify similar code. Code-to-code search means using a code query to search for similar code in a repository, which is a basic and useful function in modern code hosting platforms (e.g, Github). Code-to-code search is also critical in identifying code clones, code translation(e.g, translating python to java), and program repair.

Most of the existing tools for code-to-code search don't support cross-language search, and even those that do, utilize machine learning techniques that require massive labeled data and have high computational overhead.

The main challenge of cross-language code-to-code search is the syntactic and semantic differences between the languages. To overcome such a problem, the author use two static analysis (i.e., Token-Based Search and AST-Based Search) and one dynamic analysis (i.e., Input-Output Based Search) techniques. Based on the results of the three analyses, non-dominated sorting is applied to extract the final ranking results.

The author evaluates COSAL using a dataset that contains 43,146 Java and Python files from AtCoder and 55,499 Java files from BigCloneBench. The experimental results show that COSAL has better precision and recall compared to state-of-the-art within-language and cross-language code-to-code search tools. The author also evaluates three program analyses independently and the results indicate COSAL outperforms them. The improvement brought by the application of non-dominated sorting is also evaluated, comparison with a weighted aggregation of sorting measures shows that it improves the quality of results for the code-to-code search.

Further, although the experiments are limited to two languages (i.e., Java and Python), the authors discuss ways to generalize COSAL to more languages. The authors also explore how COSAL could work with an arbitrary open-source project since neither dataset (i.e., AtCoder and BigCloneBench) used in this work is particularly realistic. In addition, the authors discuss the limitations imposed by using only three program analysis techniques (i.e., Token-Based Search, AST-Based Search, and Input-Output Based Search) and insight ways to introduce more techniques into this tool.

II. Strengths and Weaknesses

A. Strengths

+ COSAL outperforms existing state-of-the-art methods in both single-language and cross-language code-to-code search without using high computational overhead machine learning techniques.

+ Application of non-dominated sorting is a novel and effective way to generate ranking results from multiple analyses.

+ Evaluation of *Single vs Multiple Search Similarity Measures* clearly shows the improvement brought by each analysis technique.

+ Comparative experiments with state-of-the-practice methods (i.e., GitHub and ElasticSearch) demonstrate the potential of this tool for real-world applications.

+ Three scalability explorations insight the searchers how to generalize and improve this tool, which may drive progress in the field of cross-language code-to-code search.

B. Weaknesses

- The way to extract non-language-specific tokens described in their Sec.III-A is naïve, which can lose lots of semantic information of the source code.

- The rule of generating the generic AST for AST-Based Search is a manual job and "based on our intuition", which greatly reduces the ability to generalize COSAL to other languages.

- Since COSAL consists of three analyses, the authors should perform ablation experiments to demonstrate the effect of each part on the performance of the tool.

- Although in their Sec.VI-A the authors perform the evaluation of *Single vs Multiple Search Similarity Measures*, the detailed reason why each part improves the performance and why the improvement of one part is smaller than the another is lacking.

III. Overall Evaluation

A. Soundness

The two static analysis methods and one dynamic analysis method used in COSAL are derived from direct use or improvement of previous work, and the authors clearly explain the rationale for using all three methods and provide a detailed description. The rationale and methods for applying non-dominated sorting are also fully explained by the authors.

Experiments on AtCoder and BigCloneBench, two datasets containing large amounts of Python and Java source code, also confirm this as a state-of-the-art and state-of-the-practice tool for cross-language code-to-code search.

The biggest concern is the ability to generalize to other languages of COSAL, which is highly important for its cross-language feature. Unfortunately, the dataset used for the experiments contains only two languages, Python and Java. Even though the authors discuss the basic idea of extending this tool to other languages in their Sec.VII-C-2, it is very general and vague. Especially the authors mention that "The generic AST is based on our intuition and chosen languages", which indicates that generalizing COSAL to other new languages needs a manual effort and the author only give limited standards and inspiration for how to design the generic AST.

B. Significance

Code-to-code search is useful in many subfields of program analysis, including identifying code clones, finding code translations in different languages, and program repair. It is also a technology in itself that can improve developer productivity and speed of learning a new programming language. This work accomplishes the task of cross-language code-to-code search by combining static, dynamic analysis, and non-dominated sorting. It outperforms previous machine learning-based approaches. This can enlighten people to discover more innovative approaches rather than just applying machine learning methods to program analysis scenarios to brush up the accuracy rate.

C. Novelty

According to the authors, COSAL is the first code-to-code search approach using non-dominated sorting over static and dynamic similarity measures. The main novelty is to combine static and dynamic analyses for cross-language code-to-code search and introduce non-dominated sorting into multi-source result ranking. This combination is indeed innovative and achieves good performance, but the design of each part itself is relatively simple. Most of the methods are derived from previous work and are not innovative. Therefore, the overall novelty of this paper is in proposing a novel framework for cross-language code-to-code search rather than in designing novel analysis algorithms.

D. Verifiability

The authors describe the datasets in detail used for the experiments, and for each experiment, the special preprocessing methods of the data are also given. COSAL is open-source and a basic tutorial is given. This is sufficient to allow all researchers to reproduce and validate the results of the paper.

IV. Discussion

A. Connection

Since I am a beginner in program analysis and do not have a great deal of knowledge about it, the most relevant part of this paper to the techniques I have learned is AST-Based Search. COSAL uses an AST-based static analysis technique. It converts the respective ASTs of different languages into generic ASTs and then uses the Zhang-Sasha algorithm to compute the similarity between ASTs. The algorithm computes the minimum number of edits required to transform an ordered labeled tree to another.

B. Brainstorming

1. AST-based Search can potentially be replaced by IR-based search. IR can more easily unify different programming languages into the same representation, while generic AST requires special transformation rules to be set manually.

2. graphs such as DFG, CFG, and CG can represent program semantics and other information very well [1], and it is easy to generate graphs with uniform rules even for different programming languages because they are based on IR. By calculating the similarity between these graphs [2]–[4], we may achieve better performance on cross-language code-to-code search tasks.

3. As stated in Sec. II-B, Token-based Search uses a very simple token generation method that loses context and some semantic information. A common practice in NLP is to perform vector embedding, which has proven to be very effective in program analysis areas such as code completion and code generation [5]. Although this may introduce machine learning techniques that increase the sensitivity to the dataset and make the generalization of the experimental results not guaranteed, it is a promising solution to improve the accuracy of this task.

References

[1] G. Gharibi, R. Tripathi, and Y. Lee, "Code2graph: automatic generation of static call graphs for python source code," in *Proceedings of ASE*, 2018, pp. 880–883.

[2] Y. Li, J. Jang, and X. Ou, "Topology-aware hashing for effective control flow graph similarity analysis," in *Proceedings of SecureComm*, vol. 304, 2019, pp. 278–298.

[3] C. Puodzius, O. Zendra, A. Heuser, and L. Noureddine, "Accurate and robust malware analysis through similarity of external calls dependency graphs (ECDG)," in *Proceedings of ARES*, 2021, pp. 57:1–57:12.

[4] G. Nikolentzos, P. Meladianos, and M. Vazirgiannis, "Matching node embeddings for graph similarity," in *Proceedings of AAAI*, 2017, pp. 2429–2435.

[5] F. Liu, G. Li, Y. Zhao, and Z. Jin, "Multi-task learning based pre-trained language model for code completion," in *Proceedings of ASE*, 2020, pp. 473–485.