A practical tool for detecting cross-language code pairs with similar control structures

Feng DAI
The University of Tokyo
Tokyo, Japan
daifeng@csg.ci.i.u-tokyo.ac.jp

Shigeru Chiba
The University of Tokyo
Tokyo, Japan
chiba@acm.org

ABSTRACT

Detecting cross-language code pairs with similar control structures is of practical importance in software maintenance. In this paper, we present a novel tool for fast and accurate detection of structurally-similar cross-language code pairs. We manage to obtain a much faster speed compared with the state-of-the-art technique and keep very close accuracy.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**.

KEYWORDS

cross-language code pairs, code representation, software engineering, similar code pairs

ACM Reference Format:
Feng DAI and Shigeru Chiba. 2024. A practical tool for detecting cross-language code pairs with similar control structures. In *The 39th ACM/SIGAPP Symposium on Applied Computing (SAC '24), April 8–12, 2024, Avila, Spain.* ACM, New York, NY, USA, Article 4, 3 pages. https://doi.org/10.1145/3605098.3636134

1 INTRODUCTION

Similar code pairs have been known as a source of software-maintenance problem [8] for decades. Recently, detecting similar code pairs in different programming languages is attracting more interests as software development is getting polyglot. We are particularly interested in cross-language code pairs with similar control structures as they are more practical.

We develop a novel tool for fast and accurate detecting cross-language code pairs with similar control structures. Our tool exploits neural-network models to pre-determine a few most-possible candidates, and further calculates tree-based editing-distances deterministically to select the most similar code fragment pair in control structures. To promote accuracy, we develop a new code representation technique called two-level generic AST representation for neural-network models. We discover that constructing **two** different kinds of generic ASTs can significantly increase the success rate of pre-determination, and thus improve final accuracy.

We also create a code-pair dataset for evaluating detection of cross-language code pairs with similar control structures. To the best of our knowledge, we are the first to create a dataset specially targeting at structurally-similar code pairs.

SAC '24, April 8–12, 2024, Avila, Spain

ACM ISBN 979-8-4007-0243-3/24/04.
https://doi.org/10.1145/3605098.3636134

2 MOTIVATION

Detecting cross-language code pairs with similar control structures is crucial in code refactoring and software maintenance when software development is polyglot. One scenario is modern web-service development where different platforms require different languages. When two platforms implement the same logic, structurally-similar cross-language code pairs may appear and result in code duplication. For example, in Fig. 1, two clients implement the grouping-posts feature because the server only provides an API to return a list of the user posts. It leads to code duplication. For maintenance efficiency and system stability, this pair of code fragments should be grouped into a common API and moved to the server side.

Our goal is to find such code pairs. It is similar to traditional cross-language *code clone detection*, but we are more focused on similar control structures while traditional cross-language clone detection mainly considers the same functionality and ignores detailed implementations. In fact, it would be dangerous to ignore detailed structures in real-world development. For example, if two functions use different algorithms, merging them into the server side might be dangerous. On the contrary, if two functions have similar structures, they would implement the same or similar functionality with a high possibility. Therefore, we are more interested in structural similarity.

Tools [1, 4, 10, 12, 13, 15, 16] for traditional cross-language code clone detection are not satisfying for structural similarity detection. They are either not specialized for structurally-similar cross-language code pairs, or not applicable for large structurally-similar cross-language code pair detection. [11] notices the importance of similar control structures and provides a successful technique for detecting similar structures between different languages. The technique transforms an AST in a source language into *a generic AST*, which is a language-agnostic intermediate tree-representation among multiple languages, and calculates *tree-based editing distances* to discover similar trees or similar control structures in source code. It can encompass syntactic features from different languages while ignoring syntactical differences. This work uses a simple design and a straight-forward node-to-node mapping, and proves to be accurate enough in finding cross-language code pairs with similar control structures. But its detection speed is very slow since computing editing distances between generic ASTs is heavily compute-intensive. Therefore, it is unrealistic to be used in large-scale detection.

```python
def group_posts:
    res = {}
    for post in posts:
        bucket = res.setdefault(post.owner, [])
        bucket.append(post)
    return res
```

(a) Python code

```java
public Map<String, List<Post>> groupPosts(List<Post> posts){
    Map<String, List<Post>> grouped = new HashMap<>();
    for (Post post: posts){
        if (!grouped.containsKey(post.getOwner())){
            grouped.put(post.getOwner(), new ArrayList<Post>());
        }
        grouped.get(post.getOwner()).add(post);
    }
    return grouped;
}
```

(b) Java code

Figure 1: A structurally-similar code pair in web-services

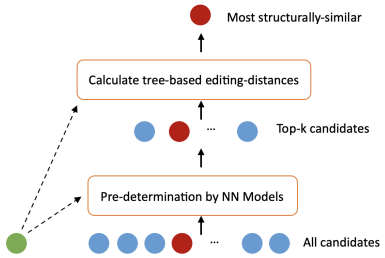

Figure 2: System overview

3 SYSTEM DESIGN

To address the detection speed issue of [11] in Section 2, we adopt a two-stage structure and use neural network models to pre-determine a few most-possible candidates, instead of calculating editing-distances of all candidates directly. After obtaining a few most-possible candidates, we further calculate their tree-based editing distances with the target to choose the most structurally-similar one deterministically. The system overview is shown in Fig. 2.

To further improve accuracy, we develop a new code representation technique called *two-level generic AST* to generate inputs and improve encoding quality for models. We design two different kinds of generic ASTs, and focus on different abstraction aspects of syntactic features. We use Java and Python as an example. *Coarse-grained generic ASTs* consider high-level structural features. It is a rough intersection of syntactic features from two languages. It ignores language-specific syntactic features, and keeps essential nodes that refer to key control structures, such as an *If* statement. In this paper, we borrow the design of coarse-grained generic ASTs from [11] with a few changes based on our understanding. *Fine-grained generic ASTs* consider low-level semantic features. It is a rough union of syntactic features from two languages because it preserves both common and language-specific features. For example, type declaration is enforced in Java but not in Python. Therefore, type declaration is a language-specific feature for Java and is kept in fine-grained generic ASTs. We also keep node properties such as identifier names to keep more semantics. The design of fine-grained generic AST is original.

We use path-based encoding [3] to transform ASTs into model inputs, and use an LSTM-based (Long short-term memory) [9] encoder-decoder model [5] to vectorize the ASTs. The models are trained to predict the method name of the given AST representing the method body. Thus, the training involves no parallel labels. After training, the vector representation from the encoder's output is used as the representation of the given AST to calculate similarity whether it is coarse-grained or fine-grained. Two models are trained separately for two kinds of generic ASTs, and two similarity scores are calculated. We add them to calculate the final similarity score between two code fragments.

4 EXPERIMENTS

We evaluate both accuracy and speed of our tool against other baselines [6, 7, 11]. We use Java-small [2] and py150k [14] datasets for training.

Table 1: Comparison between our tool and other approaches

Methods	AR	25% Quan.	50% Quan.	SR@1	SR@3	MRR	Time
Token-based Jaccard similarity algorithm	67	1	15	0.34	0.41	0.391	0.36s
Tree-based editing distance algorithm	**11**	**1**	**1**	**0.51**	**0.64**	**0.605**	131s
CodeBert	95	27	73	0.03	0.06	0.068	0.018s
Unixcoder	63	5	31	0.16	0.21	0.223	0.022s
Two-level generic ASTs	29	2	9	0.22	0.36	0.321	0.048s
Our tool (Top-15)	**27**	**1**	**1**	**0.50**	**0.56**	**0.545**	**6.55s**

The evaluation metrics include average rank (AR), quantile rank, SuccessRate@k, mean reciprocal rank (MRR) and average inferring time. In experiment settings, there is one and only one ground truth among the candidates in a single detection. The ground truth is ranked by sorting all candidates according to their similarity scores with the target. The inferring time of one detection is the time to calculate the similarity scores between the target and all candidates.

Current dataset [11, 13] is aimed for traditional cross-language *code clone detection* and is not suitable for evaluation of structurally-similar code pair detection. One of the authors organized a new dataset based on current dataset and manage to get 307 pairs of code fragments written in Java and Python. All of them have the same functionality with similar control structures. We make this dataset available in public.[1]

The results are shown in Table 1. Our tool can successfully pick up the most similar code fragment considering 50% quantile rank. In all accuracy metrics, our tool leads a large advance over all other approaches except editing-distance algorithm. Compared with tree-based editing-distance algorithm, our tool keeps a very close accuracy, but is 20 times faster in speed. The results show that our tool improves the accuracy a lot and is useful in practise.

5 CONCLUSION

In this paper, we present a tool to detect structurally-similar cross-language code pairs between Java and Python. Our tool can have very close accuracy as the state-of-the-art approach, but a much faster speed.

ACKNOWLEDGMENTS

This work is partly supported by JSPS KAKENHI JP20H00578. Thanks to Dr. Tetsuro Yamazaki, Prof. Soramichi Akiyama, Senxi Li and Yilin Zhang for all the help.

[1] https://zenodo.org/record/7824909

REFERENCES

[1] Farouq Al-omari, Iman Keivanloo, Chanchal Roy, and Juergen Rilling. 2012. Detecting Clones Across Microsoft .NET Programming Languages. *Proceedings - Working Conference on Reverse Engineering, WCRE*, 405–414. https://doi.org/10.1109/WCRE.2012.50

[2] Miltiadis Allamanis, Hao Peng, and Charles Sutton. 2016. A Convolutional Attention Network for Extreme Summarization of Source Code. In *International Conference on Machine Learning*. PMLR, 2091–2100.

[3] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. Code2vec: Learning Distributed Representations of Code. *Proc. ACM Program. Lang.* 3, POPL, Article 40 (jan 2019), 29 pages. https://doi.org/10.1145/3290353

[4] Xiao Cheng, Zhiming Peng, Lingxiao Jiang, Hao Zhong, Haibo Yu, and Jianjun Zhao. 2017. CLCMiner: Detecting Cross-Language Clones without Intermediates. *IEICE Transactions on Information and Systems* E100.D (02 2017), 273–284. https://doi.org/10.1587/transinf.2016EDP7334

[5] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1724–1734. https://doi.org/10.3115/v1/D14-1179

[6] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, Online, 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

[7] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified Cross-Modal Pre-training for Code Representation. 7212–7225. https://doi.org/10.18653/v1/2022.acl-long.499

[8] Aakanshi Gupta and Bharti Suri. 2018. *A Survey on Code Clone, Its Behavior and Applications*. 27–39. https://doi.org/10.1007/978-981-10-4600-1_3

[9] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[10] N.A. Kraft, B.W. Bonds, and R.K. Smith. 2008. Cross-Language Clone Detection. In *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE'08)* (San Francisco, CA, USA). 54–59.

[11] George Mathew and Kathryn T. Stolee. 2021. Cross-Language Code Search Using Static and Dynamic Analyses. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 205–217. https://doi.org/10.1145/3468264.3468538

[12] Kawser Wazed Nafi, Tonny Shekha Kar, Banani Roy, Chanchal K. Roy, and Kevin A. Schneider. 2019. CLCDSA: Cross Language Code Clone Detection using Syntactical Features and API Documentation. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering*. 1026–1037. https://doi.org/10.1109/ASE.2019.00099

[13] Daniel Perez and Shigeru Chiba. 2019. Cross-language Clone Detection by Learning over Abstract Syntax Trees. In *Proceedings of the 16th International Conference on Mining Software Repositories* (Montreal, Quebec, Canada) *(MSR '19)*. IEEE Press, Piscataway, NJ, USA, 518–528. https://doi.org/10.1109/MSR.2019.00078

[14] Veselin Raychev, Pavol Bielik, and Martin Vechev. 2016. Probabilistic Model for Code with Decision Trees. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications* (Amsterdam, Netherlands) *(OOPSLA 2016)*. Association for Computing Machinery, New York, NY, USA, 731–747. https://doi.org/10.1145/2983990.2984041

[15] Chenning Tao, Qi Zhan, Xing Hu, and Xin Xia. 2022. C4: Contrastive Cross-Language Code Clone Detection. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension* (Virtual Event) *(ICPC '22)*. Association for Computing Machinery, New York, NY, USA, 413–424. https://doi.org/10.1145/3524610.3527911

[16] Tijana Vislavski, Gordana Rakić, Nicolás Cardozo, and Zoran Budimac. 2018. LICCA: A tool for cross-language clone detection. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering*. 512–516. https://doi.org/10.1109/SANER.2018.8330250