FAIRSECO impact portal - An Efficient Research Software impact measurement portal for a Software Ecosystem

Research Aim

This proposal aims to develop a portal to find the impact of software available in the database using suitable metrics, and the reused codes in the database are identified using search-based algorithms.

Keywords: Software impact measurement, search-based algorithms, impact metrics

Introduction

Reusing source code or research software is common in society. As the importance of plagiarism prevention increases significantly, source code reuse detection or source code similarity detection gained considerable attention in research. Manually extracting reused codes from a massive database is a time-consuming and challenging task. So there is a need for automatic tools that accurately find reused codes or similar copies of the codes. Traditional methods, such as text-based comparisons or token-based source code detection, involve $O(n^2)$ pairwise comparisons, whereas tree-based algorithms require $O((nm)^2)$ comparisons[10]. These methods are not scalable for comparing several projects in a massive database. So, search-based algorithms like hashing[10], Genetic Algorithms (GA)[11], and Particle Swarm Optimization (PSO)[11] will play a vital role in finding source code similarity in a massive database. Searching reused source code in a database helps to find or report uncited reused source codes. Since software developers contribute to scientific progress by writing software, getting credit for such contributions is still not commonplace in many scientific domains[2]. Thus, Research software impact is helpful in getting recognition and credit for academic contributions to software and securing continued financial support for the future. It also measures the success of software.

Research impact is often measured using quantitative methods such as citation counts, the h -index, and journal impact factors. But the number of citations does not give a clear picture of the influence and quality of a research product. Several other metrics are also available to solve this issue; Co-citation networks, Eigen factor scores, etc. Our aim is to suggest a new metric or find a suitable metric for research software impact measurement.

The research questions of this study are listed below:

Research Questions?

1. Are the reused source codes or software cited or not?

2. What are the metrics which influence the impact of Research software?

The general objective of the proposed study is to explore the impact of research software. The specific **objectives** are:

1. To identify reused codes in the database
2. To explore the impact of research software using new metrics or existing metrics

In this proposal, we used search-based algorithms to identify reused codes in the database(current database) and explore the impact of research software using new metrics including the location of cited persons, co-citation and much more.

Literature Review

Source code similarity Detection:

Source code similarity detection is mainly divided into two- Attribute counting and structure metrics. Among these structure metrics are widely used in research that includes text-based, tree-based, and graph-based code similarity detection approaches[4].

Text-based approaches convert the source code into tokens and then find similarity using similarity metrics such as Levenstein, Jaccard, TF-IDF, etc[5]. Tree-based methods construct parse trees [6] or Abstract syntax trees [7] and find similarities by matching subtrees or vectors formed from the main tree structures. Whereas graph-based methods construct Program dependency graphs (PDG) [8] and Control flow graphs[9], calculate similarity by matching isomorphic sub-graphs and paths in the graph. In this, we discussed only source code similarity detection in the same programming languages.

Research Software Impact measurement:

To date, there is no proper research done on research software impact measurement. Any academic software producer like the Netherlands eScience Center (NLeSC) necessary to make their performance indicators observable. Since **Increased visibility and software impact measurements are incentives for partners to join**[1].

Methodology

The proposed approach is divided into 3 parts: 1) Source code similarity detection 2) Impact measurement of research software and 3) Evaluation of the impact measurement metrics

The proposed plan is depicted in Figure 1. The database has a number of source codes which are collected from different sites or repositories. These codes are compared for similarity in the database using various Search based algorithms and the final results are compared for selecting the best model for the existing problem.Then in the second stage various metrics(listed in the Figure 1) are considered for finding impact of research software. Finally, these metrics are evaluated to find the best impact metric.

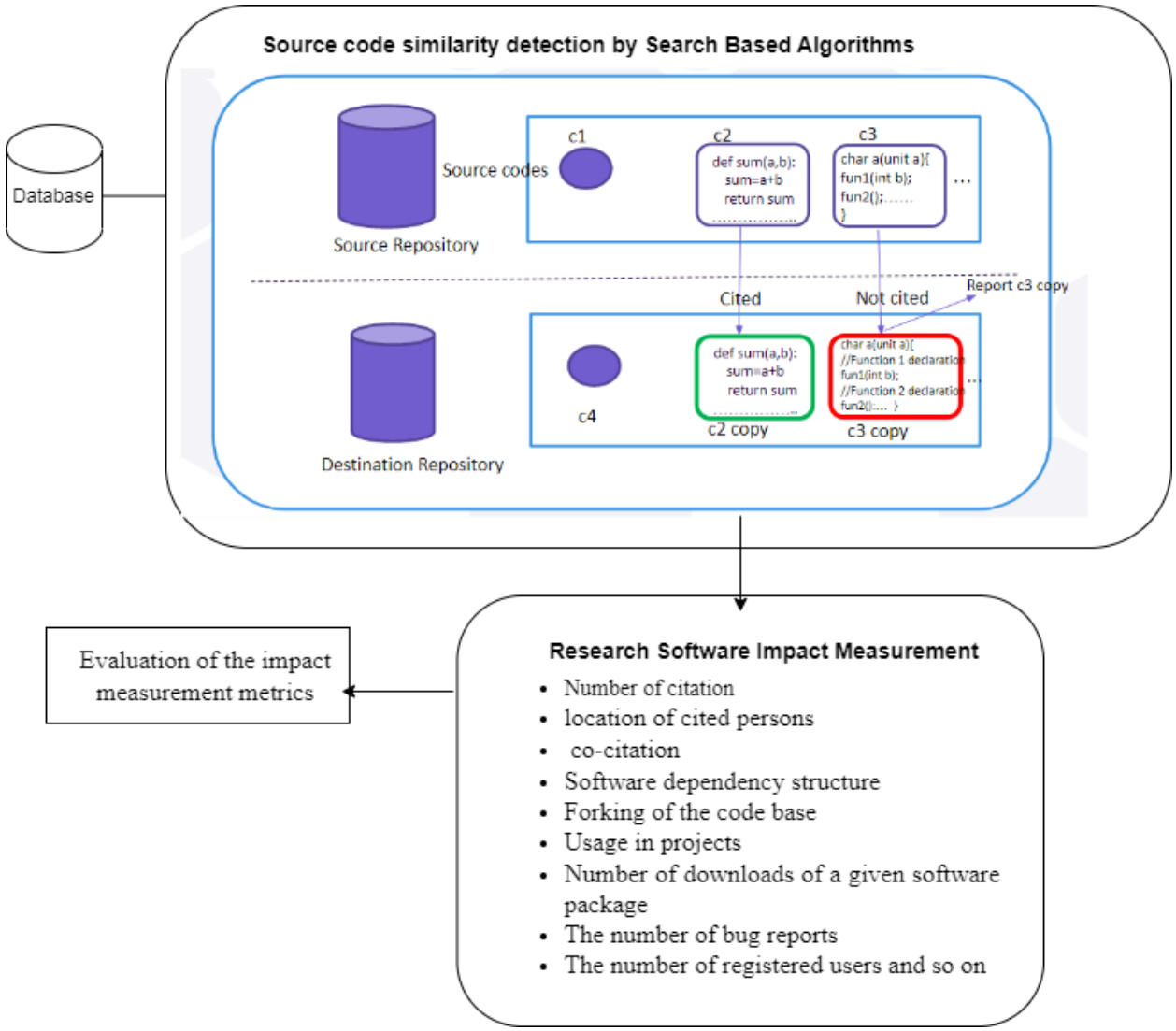

Figure 1 : Proposed methodology

Conclusion

The search based algorithms found to be helpful in the current research. It will solve pair wise comparison problems in a large database. In the proposed methodology, we planned to apply widely used search based algorithms such as Hashing, GA and PSO to the source code similarity detection. There are many different metrics available to measure software impact: number of citations, co-citation, software dependency structure, forking of the code base, usage in projects, number of downloads of a given software package, the number of bug reports, the number of registered users[3], and so on[3]. It is also important to find the metric which is more effective in measuring impact of research software.

Challenges:

1. The single impact metric itself doesn't give a clear picture of the quality of the software product. So a combination of two or more metrics can be used to get a better score for impact measurement.

2. Need to select hashing algorithm which minimizes false positives

References:

[1] van Hage, Willem & Maassen, Jason & Van Nieuwpoort, Rob. (2016). Lightning talk: Software Impact Measurement at the Netherlands eScience Center.

[2] Nick Barnes, David Jones, Peter Norvig, Cameron Neylon, Rufus Pollock, Joseph Jackson, Victoria Stodden, Peter Suber, "Science Code Manifesto", 2013. url:http://sciencecodemanifesto.org/

[3] J. H. Spaaks et al., "Painting the Picture of Software Impact with the Research Software Directory," 2018 IEEE 14th International Conference on e-Science (e-Science), 2018, pp. 23-24, doi: 10.1109/eScience.2018.00013.

[4] Zhang, F., Li, G., Liu, C., & Song, Q. (2020). Flowchart-based cross-language source code similarity detection. Scientific Programming, 2020.

[5] B. Muddu, A. Asadullah, and V. Bhat, "CPDP: a robust technique for plagiarism detection in source code," in Proceedings of the 2013, 7th International Workshop on Software Clones ICSC, pp. 39–45, San Francisco, CA, USA, May 2013.

[6] J. W. Son, S. B. Park, and S. Y. Park, "Program plagiarism detection using parse tree kernels," in Proceedings of the 9th Pacific Rim, International Conference on Artificial Intelligence PRICAI, pp. 1000–1004, Guilin, China, August 2006.

[7] T. Guo, G. Dong, H. Qin, and B. Cui, "Improved plagiarism detection algorithm based on abstract syntax tree," in Proceedings of the 2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies (EIDWT), pp. 714–719, Xi an, China, September 2013.

[8] J. Krinke, "Identifying similar code with program dependence graphs," in Proceedings of the Eighth Working Conference on Reverse Engineering WCRE, pp. 301–309, Stuttgart, Germany, February 2001

[9] H. Lim, H. Park, S. Choi, and T. Han, "A method for detecting the theft of Java programs through analysis of the control flow information," Information and Software Technology, vol. 51, no. 9, pp. 1338–1350, 2009

[10] Chilowicz, M., Duris, E., & Roussel, G. (2009, May). Syntax tree fingerprinting for source

code similarity detection. In 2009 IEEE 17th international conference on program comprehension (pp. 243-247). IEEE.

[11] Rathee, A., & Chhabra, J. K. (2019). A multi-objective search based approach to identify reusable software components. Journal of Computer Languages, 52, 26-43.