Automatic Formative Feedback Generation for Programming Tasks Using Program Repair Techniques

Jheison Morales-Vásquez
Facultad de Ingeniería
Universidad Nacional de Colombia
Bogotá, Colombia
Email: jhmoralesva@unal.edu.co

Rafael Ruiz-Niño
Facultad de Ingeniería
Universidad Nacional de Colombia
Bogotá, Colombia
Email: ramruizni@unal.edu.co

***Abstract*—The abstract goes here.**

1. Introduction

Improving the speed at which human beings acquire new knowledge is a fundamental challenge in society development. Various techniques such as Formative Learning[1] have generated many benefits including Learning Progressions, Learning Goals and Criteria for Success, Descriptive Feedback, Self and Peer Assessment and Collaboration between individuals.

However, many of these tasks are difficult to automate. In the specific field of learning about software development, instruments such as Automatic Program Repair[2] have been explored for the mechanical repairing of software bugs without the intervention of a human programmer.

This paper proposes a machine learning model for automatically generating formative feedback for programming tasks using a state-of-the-art code repair technique. It is limited to adapting a state-of-the-art repair model for use in generating formative feedback focused on basic python programming tasks.

The model employs repositories such as CodeXBlue[9], MBPP[10] and BIFI[7] to find optimal solutions about how to refine this process in the future.

1.1. Main Concepts

This section describes some findings found in the scientific literature on automatic formative feedback and automatic code repair, which are two areas of research that converge in the proposed project. The key concepts used in this document are defined below:

Formative Feedback: aimed at helping students to improve their work, is an important factor in learning. [7]

Automatic Program Repair (APR): consists of automatically finding a solution to software bugs without human intervention. [8]

In the next subsection, the relationship between both fields of research is explained in more depth.

1.1.1. Automatic Formative Feedback Generation using APR.

This section describes some findings found in the scientific literature on automatic formative feedback and automatic code repair, which are two areas of research that converge in the proposed project.

As can be seen in Figure 1, there is a variety of techniques that can be used to deliver feedback automatically to the students. The different approaches range from the most basic such as delivering predefined tracks, which are the basis of what can be group as knowledge-driven feedback, to more sophisticated models based on natural language processing using machine learning.

The use of machine learning for the automatic delivery of formative feedback is a very active field of study in recent years, however, the use of other approaches such as automatic repair of programs, which emerges as an independent branch of research, is striking. as an alternative to improve the software development cycle in the industry.

Machine repair also involves various techniques to perform the repair task, ranging from the use of genetic programming (the original approach) to the more recent use of machine learning models, especially those used in neural machine translation.

That is why they are highlighted in red, those topics that converge and that are presented to us as an interesting field of work.

Although APR is originally intended to fix bugs in productive code bases, it has proven to be an interesting way of providing feedback. In Figure 2, a timeline with some representative works of the use of automatic repair to deliver formative feedback is presented in summary form. The work of Yi and his team was identified (In our best knowledge) as the first attempt as one of the first attempts to apply APR to deliver feedback with promising results [9].

The model known as CLARA [3] presents an interesting advance in the field of automatic feedback and has become the baseline for subsequent works that seek to propose improvements in the feedback task for repair. Although the CLARA approach based on the use of the answers of the students who managed to pass the programming tasks successfully to try to predict the correctness of those who did not do well seems reasonable, it may not be adequate

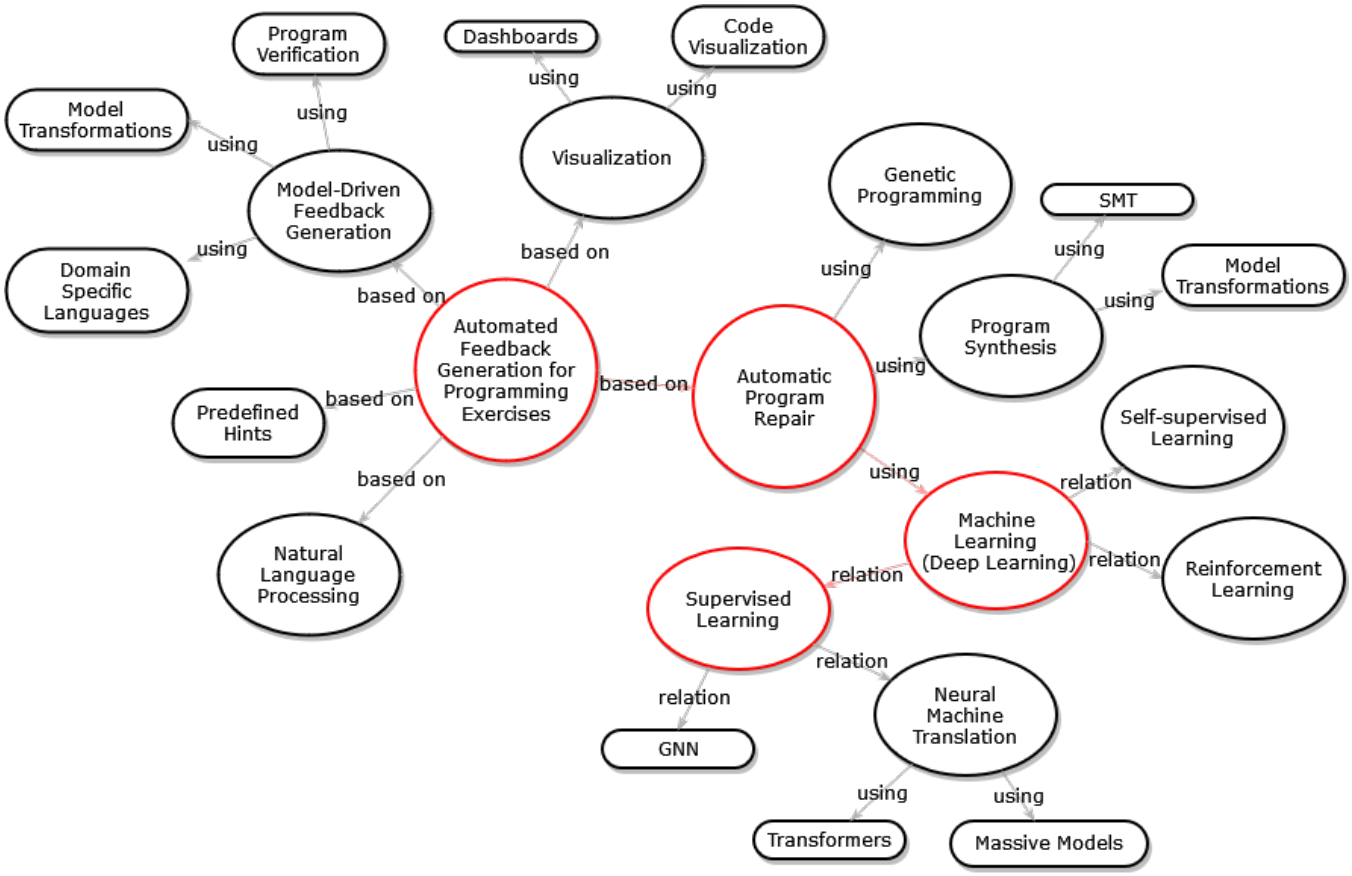

Figure 1. Relationship between formative feedback and program repair and its most relevant approaches.

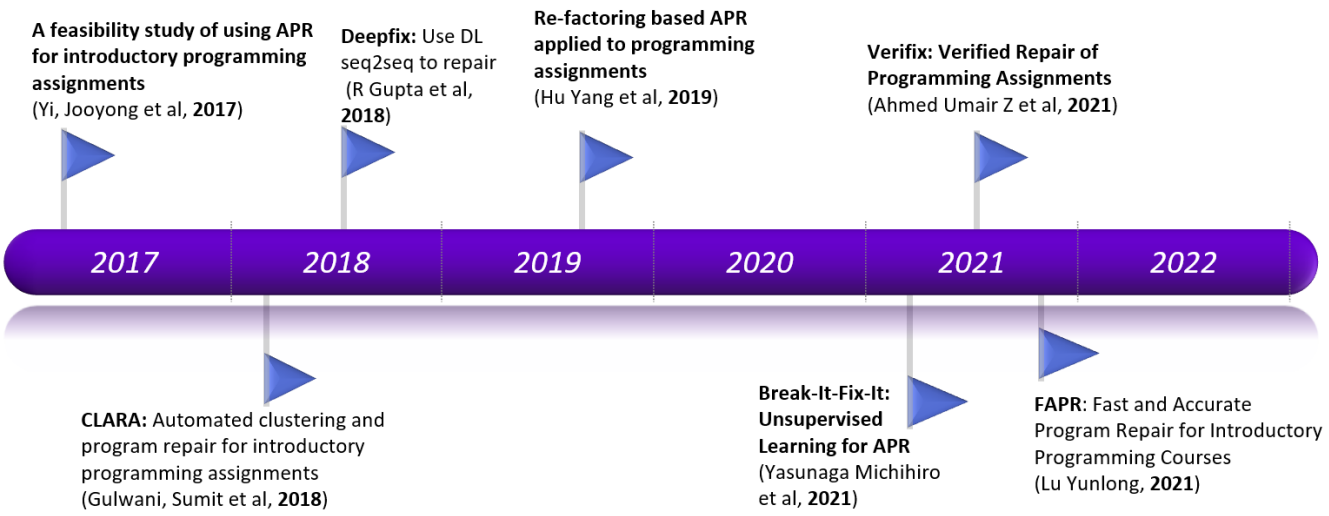

Figure 2. Some interesting approaches using program repair to give formative feedback in the last 5 years. Based on: [1] [2] [3] [4] [5] [6]

to use solutions (although correct) of students who are not programming experts because they are learners.

One of the challenges when building data sets to train learning-based repair models is to properly build the code with errors, code correct pairs. Although mining Open Source repositories is a reasonable alternative, manually

locating the commits where the bug was introduced and in which it was fixed requires considerable manual effort. One of the approaches used by the community to simplify this task is to introduce synthetic errors into correct code to train fixers. It is possible that the distribution of synthetic errors does not approximate the distribution of errors presented in a real project.

In this context, the BIFI [10] proposal is striking, in which it seeks to train a component that injects errors using snipped records of real code with errors. The next section goes into more detail about this approach.

2. Case study

A common approach to perform learning-based repair tasks is to use neural translation techniques, in which pairs are needed (Buggy Code, Correct Code). To build that kind of data set, there are two options: first, the researchers can mine repositories and manually identify which commit introduced a bug and which commit resolved it, or create training data consisting of (bad, good) pairs by corrupting good examples using heuristics. The second approach may seem attractive, especially to build large synthetic datasets from smaller sets, however, fixers trained on this synthetically-generated data do not extrapolate well to the real distribution of bad inputs [10]. To solve this problem BIFI propose: first, use the critic to check a fixer's output on real bad inputs and add good (fixed) outputs to the training data and train a breaker to generate realistic bad code from good code.

This proposal seems interesting to create synthetic datasets that allow evaluating repair models from smaller datasets or datasets that were not originally intended for repair tasks. For this reason we have decided to use this work as a basis for this project. In the next section, other possible datasets or benchmarks that can be used in conjunction with BIFI to achieve the proposed objective are explored.

3. Datasets

To meet the proposed objective, the availability of the code and data sets of the research works consulted in the literature review were used as a criterion. then it extended the search to databases of data sets using terms such as "programming assignments" "basic programming problems" as keywords in the search equation.

As a result of the search, the data sets presented in Table 1 were identified. In this table we can see CodeXBlue [6], although it was not part of the original literature review, is a benchmark whose adoption by the community has been increasing and is useful for comparing models that perform some type of task related to understanding and generating code such as repair of programs (in this case based on java).

For its part, Google has published the "Mostly Basic Python Problems Dataset" [2] in which beginner-level problems in Python are presented, although it was originally intended as a dataset for generating code from natural language, it can be useful to evaluate the models of repair programs in repair tasks to deliver formative feedback.

TABLE 1. SUMMARY OF USEFUL DATASETS TO PERFORM THE TASK PROPOSED IN THIS PROJECT.

Name	Description	Purpose in this project
CodeXBlue [6]	It includes models (based on CodeBERT and GPT) and data sets for diverse task (including program repair)	Benchmark for repair task
MBPP [2]	It consists of around 1,000 crowd-sourced Python programming problems, designed to be solvable by entry level programmers. Each problem consists of a task description, code solution and 3 automated test cases.	Benchmark for feedback generation
BIFI [10]	3 million Python3 snippets from GitHub, divided in two groups: Bad code, Correct code.	Dataset for training/validate models

One of the reasons for choosing BIFI is its replicability (its source code, like the dataset used, are publicly accessible) and its innovative approach that allows smaller datasets to be expanded, which makes it ideal to be used in conjunction with MBPP.

This proposal then consists of replicating the BIFI work and using it with the MBPP dataset to evaluate its performance (accuracy, precision, etc.), repairing assignments of basic programming problems. Our idea is to produce an experimental set where BIFI is the baseline to propose repair models for automatic feedback generation that try to improve the registered metrics.

Acknowledgments

The authors would like to thank...

References

[1] U. Z. Ahmed, Z. Fan, J. Yi, O. I. Al-Bataineh, and A. Roychoudhury, "Verifix: Verified repair of programming assignments," 6 2021. [Online]. Available: http://arxiv.org/abs/2106.16199

[2] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, and C. Sutton, "Program synthesis with large language models," 8 2021. [Online]. Available: http://arxiv.org/abs/2108.07732

[3] S. Gulwani, I. Radiček, and F. Zuleger, "Automated clustering and program repair for introductory programming assignments," *ACM SIGPLAN Notices*, vol. 53, pp. 465–480, 6 2018.

[4] R. Gupta, S. Pal, A. Kanade, and S. Shevade, "Deepfix: Fixing common programming errors by deep learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 1, pp. 1345–1351, 2017.

[5] Y. Hu, U. Z. Ahmed, S. Mechtaev, B. Leong, and A. Roychoudhury, "Re-factoring based program repair applied to programming assignments," 2019. [Online]. Available: https://github.com/githubhuyang/refactory

[6] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tufano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. Liu, "Codexglue: A machine learning benchmark dataset for code understanding and generation," 2021. [Online]. Available: http://arxiv.org/abs/2102.04664

[7] H. Keuning, J. Jeuring, and B. Heeren, "A systematic literature review of automated feedback generation for programming exercises," *ACM Transactions on Computing Education*, vol. 19, pp. 1–43, 1 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3231711

[8] M. Monperrus, "Automatic software repair: A bibliography," *Proceedings - International Conference on Software Engineering*, vol. Part F1371, p. 1219, 2018, al ser una bibliografia responde la mayoría de preguntas orientadoras, permite enteder el panorama de ASR en general. Define conceptos clave y hace una distinción entre reparación de estado y de comportamiento.

[9] J. Yi, U. Z. Ahmed, A. Karkare, S. H. Tan, and A. Roychoudhury, "A feasibility study of using automated program repair for introductory programming assignments," vol. Part F130154. Association for Computing Machinery, 8 2017, pp. 740–751.

[10] M. Yasunaga and P. Liang, "Break-it-fix-it: Unsupervised learning for program repair," 2021. [Online]. Available: http://arxiv.org/abs/2106.06600