Program Transformation Landscapes for Automated Program Modification Using Gin

Justyna Petke
University College London
UK
j.petke@ucl.ac.uk

Brad Alexander
University of Adelaide
Australia
bradley.alexander@adelaide.edu.au

Earl T. Barr
University College London
UK
e.barr@ucl.ac.uk

Alexander E.I. Brownlee
University of Stirling
UK
alexander.brownlee@stir.ac.uk

Markus Wagner
Monash University
Australia
markus.wagner@monash.edu

David R. White
University of Sheffield
UK

ABSTRACT

Automated program modification underlies two successful research areas — genetic improvement and program repair. Under the generate-and-validate strategy, automated program modification transforms a program, then validates the result against a test suite. Much work has focused on the search space of application of single fine-grained operators — COPY, DELETE, REPLACE, and SWAP at both line and statement granularity. This work explores the limits of this strategy. We scale up existing findings an order of magnitude from small corpora to 10 real-world Java programs comprising up to 500k LoC.

We systematically study the APM landscape, asking five research questions: (1) What is the relative effectiveness of the conventional edit operators: COPY, DELETE, REPLACE and SWAP? (2) How effective is DELETE when used alone? (3) Which is more effective: line or statement granular CDRS edits? (4) How much does effectiveness drop with the number of edits in a patch? and (5) What is the correlation between subject's features and its *plasticity* — the likelihood that applying APM to it will be effective?

We decisively show that the grammar-specificity of statement granular edits pays off: its pass rate triples that of line edits and uses 10% less computational resources. We confirm previous findings that DELETE is the most effective operator for creating test-suite equivalent program variants. We go farther than prior work by exploring the limits of DELETE's effectiveness by exhaustively applying it. We show this strategy is too costly in practice to be used to search for improved software variants.

We further find that pass rates drop from 12–34% for single statement edits to 2–6% for 5-edit sequences, which implies that further progress will need human-inspired operators that target specific faults or improvements.

A program is *amenable to automated modification* to the extent to which automatically editing it is likely to produce test-suite passing variants. We are the first to systematically search for a code measure that correlates with a program's amenability to automated modification (*i.e.* its plasticity). We found no strong correlations, leaving the question open.

To summarise, our key contributions are:

- We formalise the cost of automated program modification;
- We show that exhaustively applying DELETE generates a smooth search space, suggesting that uniformly sampling DELETE applications may measure test suite adequacy and increase plasticity.
- We provide conclusive evidence that statement granular edits are more effective than line;
- To spur future work, we propose plasticity, the problem of identifying code amenable to APM, conduct preliminary experiments that show how hard it is, and provide two lists of methods: those particularly amenable and those particularly resistant to APM.

CCS CONCEPTS

• **Software and its engineering → Search-based software engineering**.

KEYWORDS

Automated Program Modification , Genetic Improvement , Automated Program Repair , Search-Based Software Engineering

ACM Reference Format:
Justyna Petke, Brad Alexander, Earl T. Barr, Alexander E.I. Brownlee, Markus Wagner, and David R. White. 2023. Program Transformation Landscapes for Automated Program Modification Using Gin . In *Proceedings of 46th International Conference on Software Engineering (ICSE 2024)*. ACM, New York, NY, USA, 1 page. https://doi.org/XXXXXXX.XXXXXXX

1 PUBLICATION INFORMATION

The manuscript[1] was accepted to the Empirical Software Engineering (EMSE) journal on May 23rd. This was a journal first submission. The paper has not been presented at, and is not under consideration for, journal-first programs of other conferences. All our scripts and data are available in the following repository: https://github.com/automatedprogrammodification/automatedprogrammodification/.

ACKNOWLEDGMENTS

This work was funded by the UKRI EPSRC grants EP/P023991/1 and EP/J017515/1; Carnegie Trust grant RIG008300; Australian Research Council projects DE160100850, DP200102364, and DP210102670, and by gifts from Facebook and Google. For the purpose of open access, the authors have applied a Creative Commons Attribution (CC BY) license to any Accepted Manuscript version arising.

[1]https://link.springer.com/article/10.1007/s10664-023-10344-5