Genetic Improvement in the Shackleton Framework for Optimizing LLVM Pass Sequences

Shuyue Stella Li
sli136@jhu.edu
Department of Computer Science,
Johns Hopkins University, USA

Hannah Peeler∗
hpeeler@utexas.edu
Arm Ltd.
USA

Andrew N. Sloss†
andrew@sloss.net
Arm Ltd.
USA

Kenneth N. Reid
ken@kenreid.co.uk
Department of Animal Science,
Michigan State University, USA

Wolfgang Banzhaf
banzhafw@msu.edu
Department of CSE, Michigan State
University, USA

ABSTRACT
Genetic Improvement is a search technique that aims to improve a given acceptable solution to a problem. In this paper, we present the novel use of genetic improvement to find problem-specific optimized LLVM Pass sequences. We develop a Pass-level edit representation in the linear genetic programming framework, Shackleton, to evolve the modifications to be applied to the default optimization Pass sequences. Our GI-evolved solution has a mean of 3.7% runtime improvement compared to the default LLVM optimization level `-O3' which targets runtime. The proposed GI method provides an automatic way to find a problem-specific optimization sequence that improves upon a general solution without any expert domain knowledge. In this paper, we discuss the advantages and limitations of the GI feature in the Shackleton Framework and present our results.

CCS CONCEPTS
- Computing methodologies → Genetic programming; - Software and its engineering → Compilers.

KEYWORDS
Evolutionary Algorithms, Genetic Programming, Genetic Improvement, Compiler Optimization, Parameter Tuning, Metaheuristics

ACM Reference Format:
Shuyue Stella Li, Hannah Peeler, Andrew N. Sloss, Kenneth N. Reid, and Wolfgang Banzhaf. 2022. Genetic Improvement in the Shackleton Framework for Optimizing LLVM Pass Sequences. In Proceedings of The Genetic and Evolutionary Computation Conference 2022 (GECCO '22). ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3520304.3534000

∗At time of publication, no longer affiliated with Arm. Reachable at hpeeler@utexas.edu
†Now at University of Washington, Seattle, WA, USA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GECCO ’22, July 9–13, 2022, Boston, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9268-6/22/07...$15.00
https://doi.org/10.1145/3520304.3534000

1 INTRODUCTION
Genetic Improvement (GI) [6] automatically improves upon a given solution using Genetic Programming (GP), a powerful search algorithm that can efficiently find the near-optimal solution in a large search space[1, 5]. This approach is inspired by the process of natural selection [3], in which fitness advantage guides the passing of genetic information to the next generation[4]. Linear Genetic Programming (LGP) [2] is a special application in which the genetic information of each individual codes for active elements in the population represented in a sequential order. The Shackleton Framework1 is a generalized LGP framework that allows the use of GP on any user-defined object types and fitness metrics [9].

In the GI feature of the Shackleton Framework (Shackleton-GI), each modification from the starting solution is represented as a sequence of operations to be applied to that solution. The use-case of interest in our experiments is the optimization of LLVM Compiler Optimization Pass (Pass) sequences [7]. LLVM is a collection of modular and reusable (language/target independent) compiler technologies. Different compile-time optimizations can be specified using Passes, which mutate the program in order to optimize some metric (e.g. runtime)[10]; a sequence of Passes can be specified at compilation to achieve a particular optimization goal. In LLVM, there are a number of default optimization levels, -Ox, that contain encoded sequences of 10 to 90 Passes. The default LLVM optimization level `-O3' (-O3) enables optimizations that primarily target the program runtime [7, 8]. Shackleton-GI evolves a series of insertion, deletion, and replacement edit operations, which produces a more powerful optimization Pass sequence when applied to a solution to a general problem (the sequence of -O3 Passes in our case).

2 METHODS
Shackleton [9] is a flexible LGP framework, in which various types of objects can be treated as genes and optimized using Genetic Algorithm (GA). In Shackleton-GI, we develop a Pass-level edit representation in which individuals consist of ‘genes’ that are edits. An edit has a type field, a position field, and a value field. Given the source code of a target program, Shackleton-GI generates a sequence of edits that will be used to modify a starting sequence of Passes.

1https://github.com/ARM-software/Shackleton-Framework
2https://llvm.org
Therefore, finding the absolute optimum for a given source code is computationally impossible with existing methods. -O3 is carefully designed to reduce the runtime of a general target program. Hence, it gives a good starting point for the search and significantly reduces the size of the search space. The Pass-level edit representation in Shackleton-GI effectively searches near this initial starting point and does not contain problem-specific optimizations. Shackleton-GI is a novel application of GI and a first step in exploring a flexible use case of Shackleton. Future directions in the development of Shackleton-GI are: First, measuring fitness of individuals with CPU time by altering the threading design to avoid fluctuations caused by resource sharing on the same computing cluster. Second, our experiments used the optimal hyperparameter values found by [9] in a LGP (non-GI) environment; additional hyperparameter tuning might result in further runtime improvements as this is a different use case. Further investigation into other optimization objectives (e.g. peak memory consumption, I/O energy), other GI algorithms, and a wider range of test problems would also be interesting areas of future research.

5 ACKNOWLEDGEMENTS

This work was generously funded by MSU as well as the John R. Koza Endowment. We also gratefully acknowledge The Institute of Cyber-Enabled Research (ICER) at MSU for providing the hardware infrastructure that made the required computation possible.

REFERENCES

[1] Wolfgang Banzhaf, Peter Nordin, Robert Keller, and Frank Francone. 1998. Genetic Programming- An Introduction. Morgan Kaufmann, San Francisco.
[2] Markus Brameier and Wolfgang Banzhaf. 2007. Linear Genetic Programming. Springer, New York.
[3] Charles Darwin. 1859. On the Origin of Species by Means of Natural Selection. John Murray, London.
[4] John Holland. 1975. Adaptation in Natural and Artificial Systems. MIT Press, Cambridge.
[5] John R Koza. 1992. Genetic Programming. MIT Press, Cambridge.
[6] Wülliam B Langdon and Mark Harman. 2014. Optimizing existing software with genetic programming. IEEE Transactions on Evolutionary Computation 19, 1 (2014), 118–135.
[7] Chris Lattner. 2006. Introduction to the llvm compiler infrastructure. In Itanium conference and expo.
[8] Chris Lattner. 2008. LLVM and Clang: Next generation compiler technology. In BSD Conference BSDCan 2008 (University of Ottawa, Canada).
[9] Hannah Peeler, Shuyue Stella Li, Andrew N Sloss, Kenneth N Reid, Yuan Yuan, and Wolfgang Banzhaf. 2022. Optimizing LLVM Pass Sequences with Shackleton: A Linear Genetic Programming Framework. arXiv preprint arXiv:2201.13305 (2022).
[10] Suyog Sarda and Mayur Pandey. 2015. LLVM essentials. Packt Publishing Ltd.