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.

GECCO '22, July 9–13, 2022, Boston, USA

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 Framework[1] 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[2] 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.

[1]https://github.com/ARM-software/Shackleton-Framework
[2]https://llvm.org

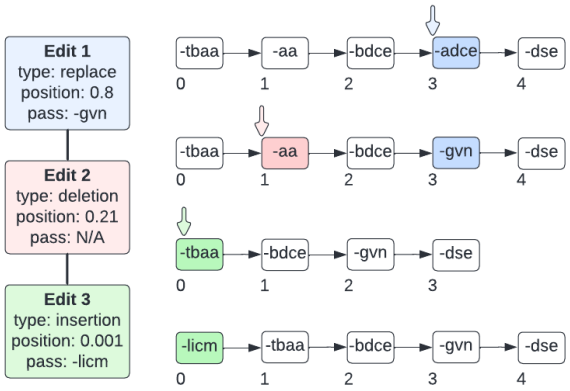

Figure 1: Sample Edit Representation of GI. Three different edits (left) are applied to a Pass sequence (right): insertion (1), deletion (2), and replacement (3). Position field: relative position between 0 and 1; value field: Pass name.

A demonstration of the process is shown in Figure 1. An 'individual' with three edits is applied to an initial sequence of five Passes. The modified Pass sequence is used as the compiler optimization arguments. A user-provided source program is compiled with these arguments and the average runtime over 40 runs is recorded as the fitness of the 'individual' of 3 edits. This minimizes the effect of runtime inconsistency due to system fluctuations and any unusual halting in the user-provided source program.

For more detail on the source program choice please see [9]. In our experiments, the source program used is the Backtrack Algorithm for the Subset Sum Problem (SSP)[3]. The to-be-modified sequence is -O3. There are a number of hyperparameters required for Shackleton, and we used the optimal hyperparameter combination found in [9]. The experiments were conducted on HPCC nodes running CentOS Linux version 7 and Clang version 8.0.0.

3 RESULTS AND DISCUSSION

Eight repeated trials were run with the same hyperparameter values. Fitness across generations for two sample trials are plotted in Figure 2, with horizontal lines as the baseline runtime. Figure 2(a) shows a converging pattern that starts at a high runtime then decreases; Figure 2(b) shows a high quality initialization, and stays within the same range during the entire evolutionary process. Both scenarios outperforms the baselines. Shackleton-GI outperforms -O3 on the SSP is 3.7% with a standard deviation of 0.8768. The p-value for the left-tail test when the null hypothesis for the mean percent improvement of 0 is 0.000012%. This shows the robustness of the algorithm and its readiness to be experimented with in production.

The search space for Pass sequences is in the order of 10^{167} (using 120 different Passes in sequences of approximately 80 Passes long). 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 is able to find a local minimum tailored to the specific source program. The use of GI significantly increases the efficiency of the search compared to a run from random solutions [9] and is able to provide a better solution than -O3 as-is.

[3]https://github.com/parthnan/SubsetSum-BacktrackAlgorithm

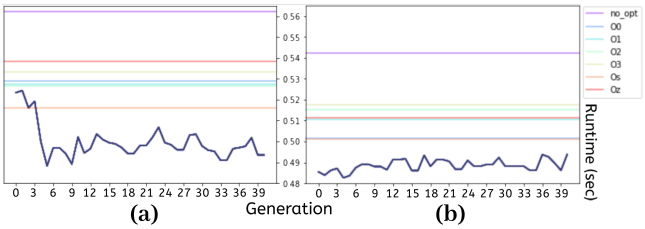

Figure 2: Runtime Improvement of Two Sample Trials

4 CONCLUSION AND FUTURE WORK

The -O3 sequence is hand-crafted with expert domain knowledge, and does not contain problem-specific optimizations. Shackleton-GI automatically produces a sequence of edits that generates a problem-specific optimization solution to a user-provided source program. We proposed a Pass-level edit representation for GI that can be extended into different object types, and showed that our approach is able to achieve substantial runtime improvements compared to a strong compiler baseline.

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] William 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.