Learning LLVM Optimization Passes for Faster Code

Reto Merz

Introduction

Classical compilers improve code in a **sequence of behavior-conserving optimization passes** like *dead code elimination*, *loop unrolling* or *vectorization*, whose order impacts the quality of the generated machine code and is either fixed or chosen based on heuristics. The goal of this project was to explore whether RL agents based on the **graph structure** of general purpose **LLVM** IR code can learn **execution-speed** enhancing sequences of optimization passes.

- **States** Programs in LLVM IR
- **Actions** 124 Optimization Passes
- **Dynamics** Opt. Pass processed by LLVM compiler
- **Rewards** Improvement in Execution Speed

Challenges

1. **Discrete action space** of 124 optimization passes
2. **Slow rollout** (~1s per step) due to compilation and benchmarking of intermediate programs
3. **Noisy rewards** associated with benchmarking execution speed
4. High **number of hyperparameters** for an unexplored problem

Setup

My code interacts with the LLVM compiler in a `compiler_gym` environment [4]. I first implemented entropy-maximization PPO with GAE. Due to the poor performance of PPO in the given sample regime, I additionally adapted SAC to discrete action spaces and implemented it. The code for the project can be found on GitHub:
github.com/retolucamerz/learning-optimization-passes

- Hardware: i5-8600K / GTX 1080
- Dataset: CBench-v1 [5]
- 25+ runs using 6+ days of compute

References

[1] Chris Cummins, Zacharias V Fisches, Tal Ben-Nun, Torsten Hoefler, Michael FP O'Boyle, and Hugh Leather. Programl: A graph-based program representation for data flow analysis and compiler optimizations. In International Conference on Machine Learning, pages 2244–2253. PMLR, 2021

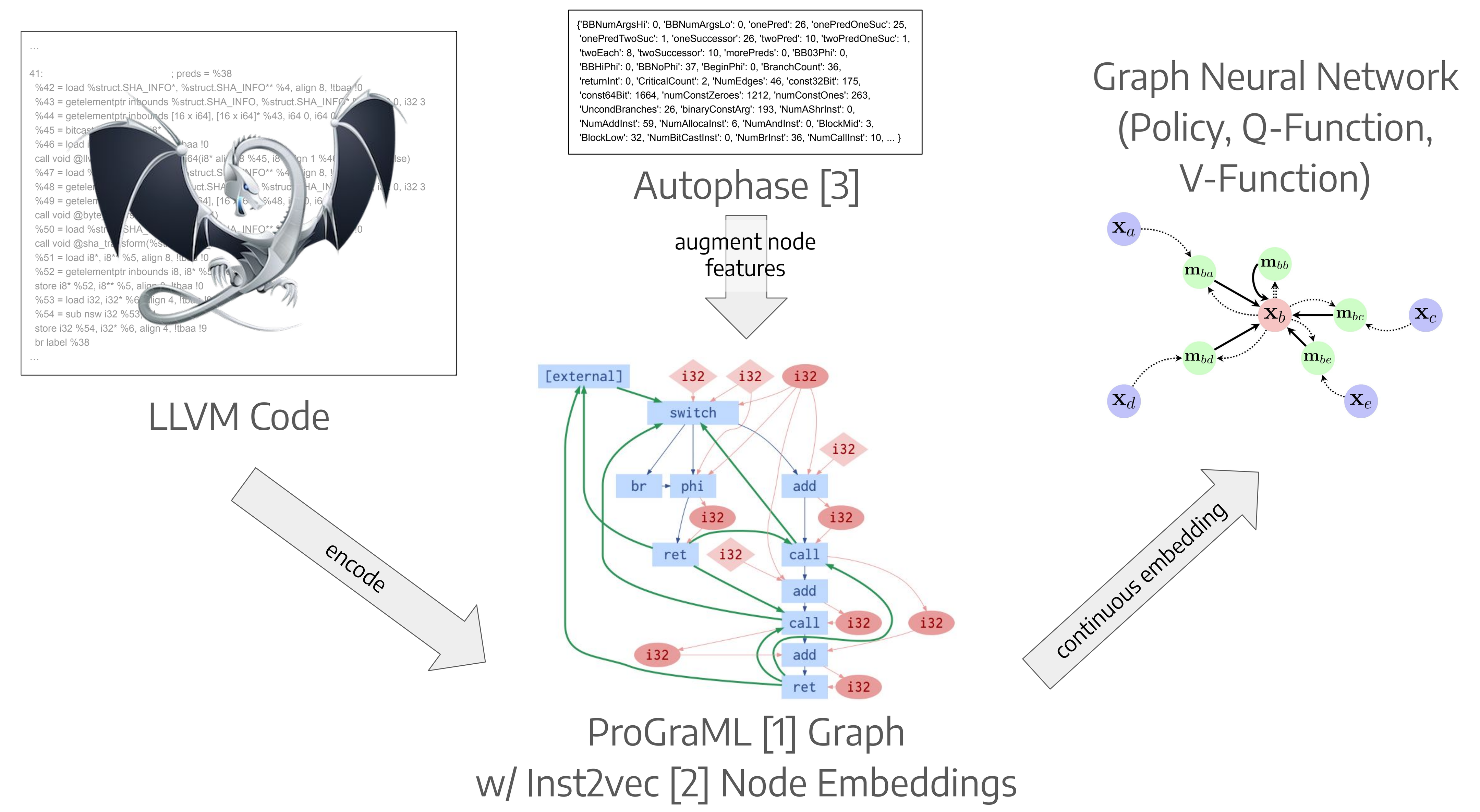
[2] Tal Ben-Nun, Alice Shoshana Jakobovits, and Torsten Hoefler. Neural code comprehension: A learnable representation of code semantics. Advances in neural information processing systems, 31, 2018

[3] Haj-Ali A, Huang QJ, Xiang J, Moses W, Asanovic K, Wawrzynek J, Stoica I. Autophase: Juggling hls phase orderings in random forests with deep reinforcement learning. Proceedings of Machine Learning and Systems. 2020 Mar 15;2:70-81

[4] Chris Cummins, Bram Wasti, Jiadong Guo, Brandon Cui, Jason Ansel, Sahir Gomez, Somya Jain, Jia Liu, Olivier Teytaud, Benoit Steiner, et al. Compilergym: Robust, performant compiler optimization environments for ai research. In 2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), pages 92–105. IEEE, 2022

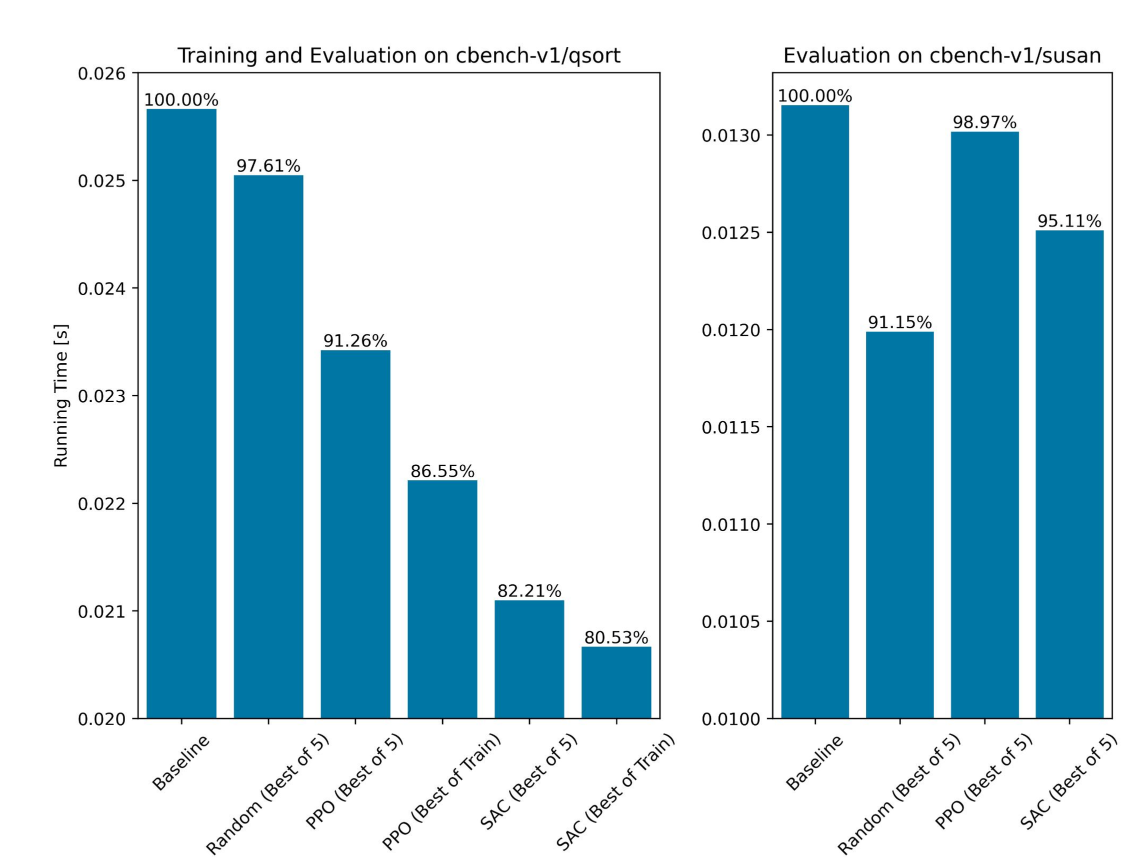
[5] Grigori Fursin. Collective tuning initiative: automating and accelerating development and optimization of computing systems. In GCC Developers' Summit, 2009

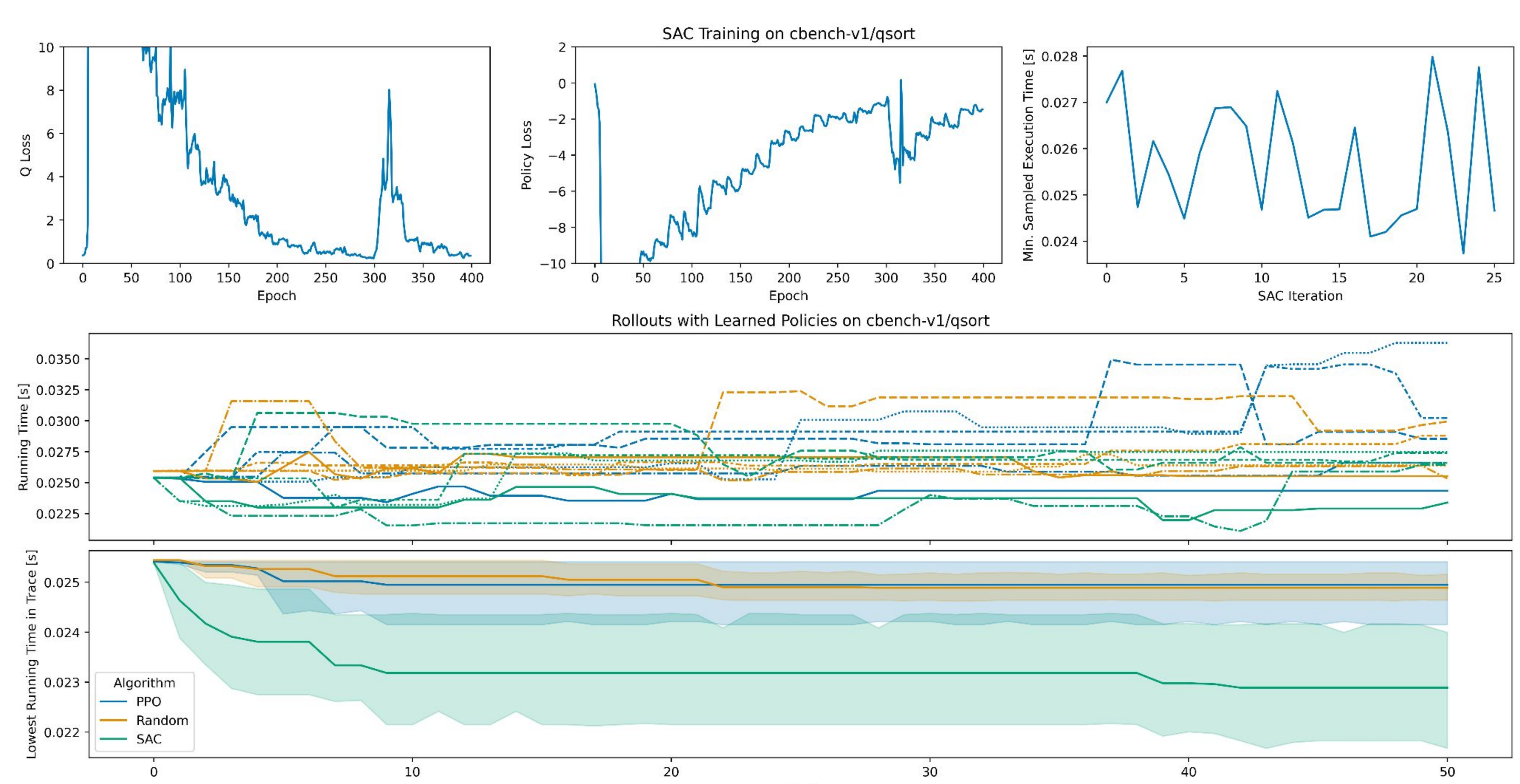
Model



Experiments and Results

PPO training was quite difficult, with many runs not improving beyond random rollouts. Experimentation with different graph convolutions, augmented state representations (Autophase), and a 2.5x speedup in rollouts due to reusing compilation and benchmarking results didn't help much.
SAC had quite a few stability issues during training, sometimes diverging completely. Scaling the rewards improved the convergence of the Q loss, and thus enabled learning working policies.





Conclusion

The limited available data suggests that it is possible to learn agents that optimize the execution time of *single* programs for some target hardware. However, I have not been able to train converging policies on multiple programs. These results indicate that there may be better (non-ML related) methods such as heuristics based search for this problem.