Genetic Improvement in the Shackleton Framework for Optimizing LLVM Pass Sequences

Stella Li, Hannah Peeler, Andrew Sloss, Kenneth Reid,
Wolfgang Banzhaf

Michigan State University, Johns Hopkins University, Arm Ltd.

Genetic Improvement Workshop | GECCO

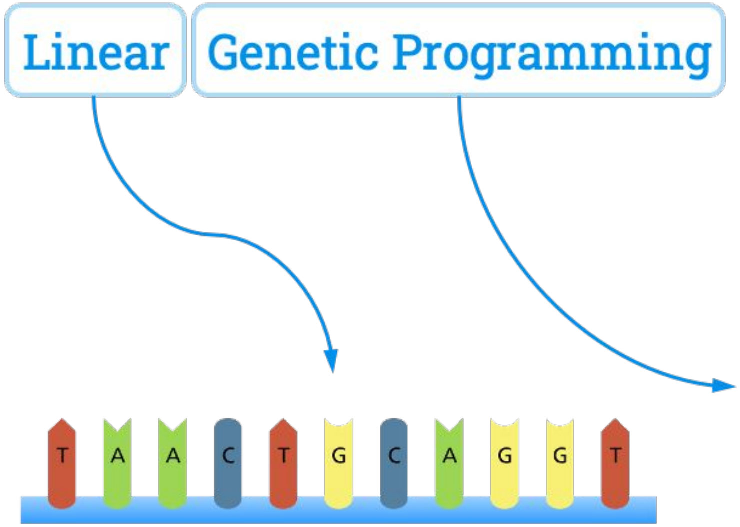
https://arxiv.org/abs/2204.13261

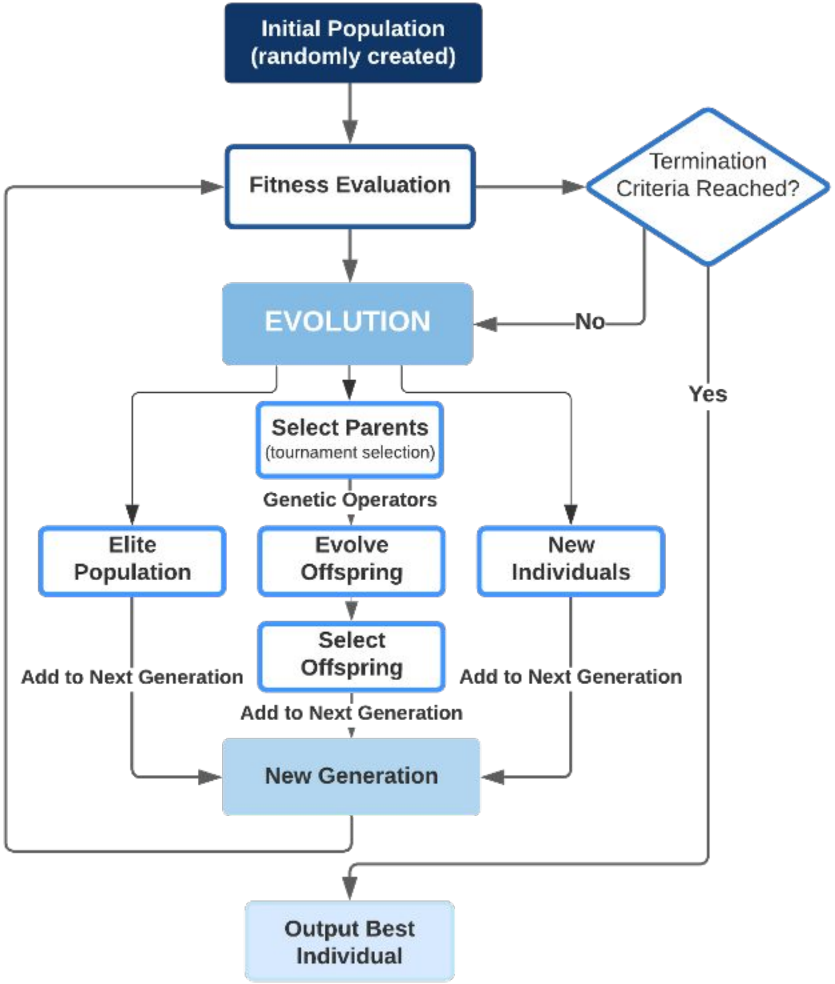
Key Points

- Using genetic improvement to find optimized LLVM Pass sequences
- Automatic way to find a problem-specific optimization sequence
  - Without expert domain knowledge
- 3.7% runtime improvement (good in compiler world)

Overview

I. The Shackleton Framework

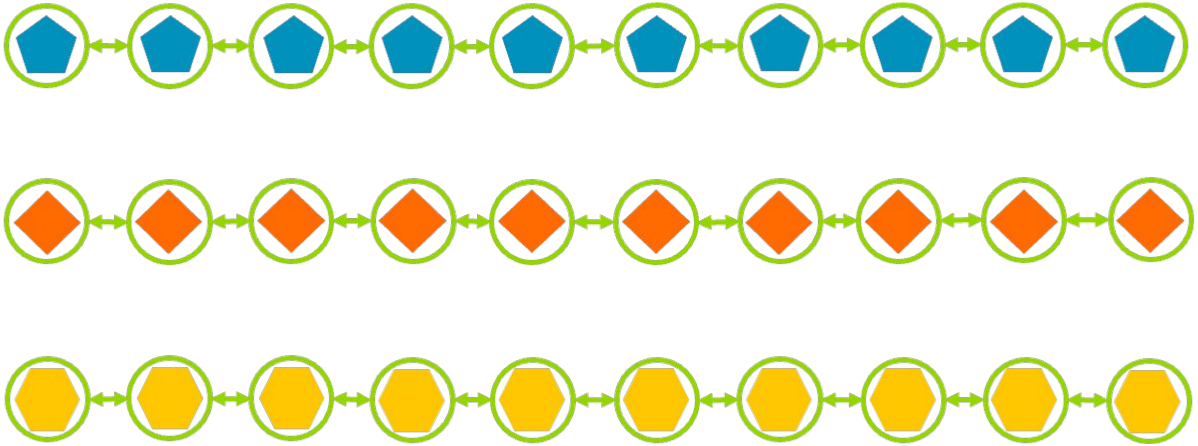




Osaka List Structure

Generalized linear representation of objects that unifies:

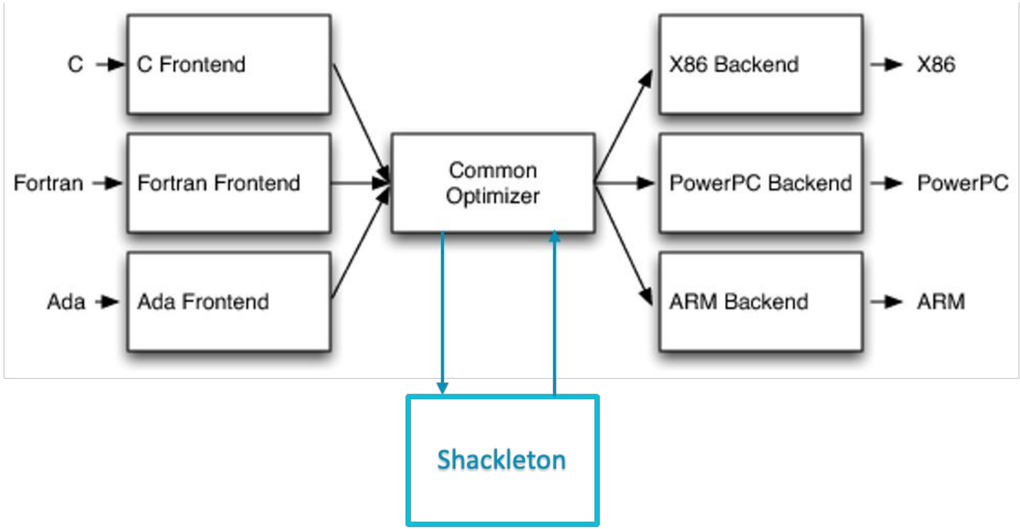
- Initialization
- Selection
- Crossover
- Mutation



- Customized fitness

II. LLVM Optimization Passes

- New target application, mostly unrelated to GA
- Variety of possible injection points for GA
  - Common Optimizer
  - Transitions from frontend to middle, middle to backend
- Open source
  - Easily accessible, popular



A bit more about LLVM - default optimization levels

-O0: compiles the fastest and generates the most debuggable code

-O1: in between -O0 and -O2

-O2: moderate level of optimization which enables most optimizations

-O3: -O2 plus optimizations that take longer to perform or that may generate larger code (in an attempt to make the program run faster)

-O4: adds link-time optimization

-Os: -O2 with extra optimizations to reduce code size

-Oz: -Os (and thus -O2) but reduces code size further

A bit more about LLVM - default optimization levels

-O0: compiles the fastest and generates the most debuggable code

-O1: in between -O0 and -O2

-O2: moderate level of optimization which enables most optimizations

-O3: -O2 plus optimizations that take longer to perform or that may generate larger code (in an attempt to make the program run faster)

-O4: adds link-time optimization

-Os: -O2 with extra optimizations to reduce code size

-Oz: -Os (and thus -O2) but reduces code size further

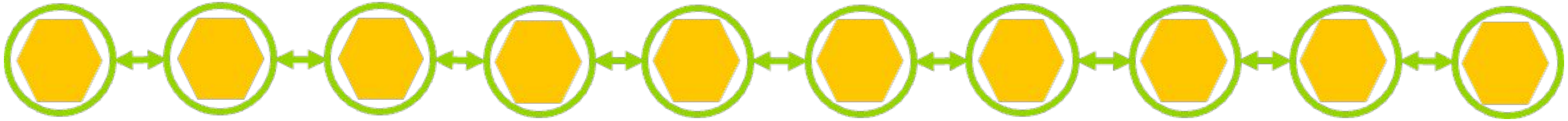
III. Genetic “Edit” Rules

1. Deletion
2. Insertion
3. Replacement

Inside an “Edit”:

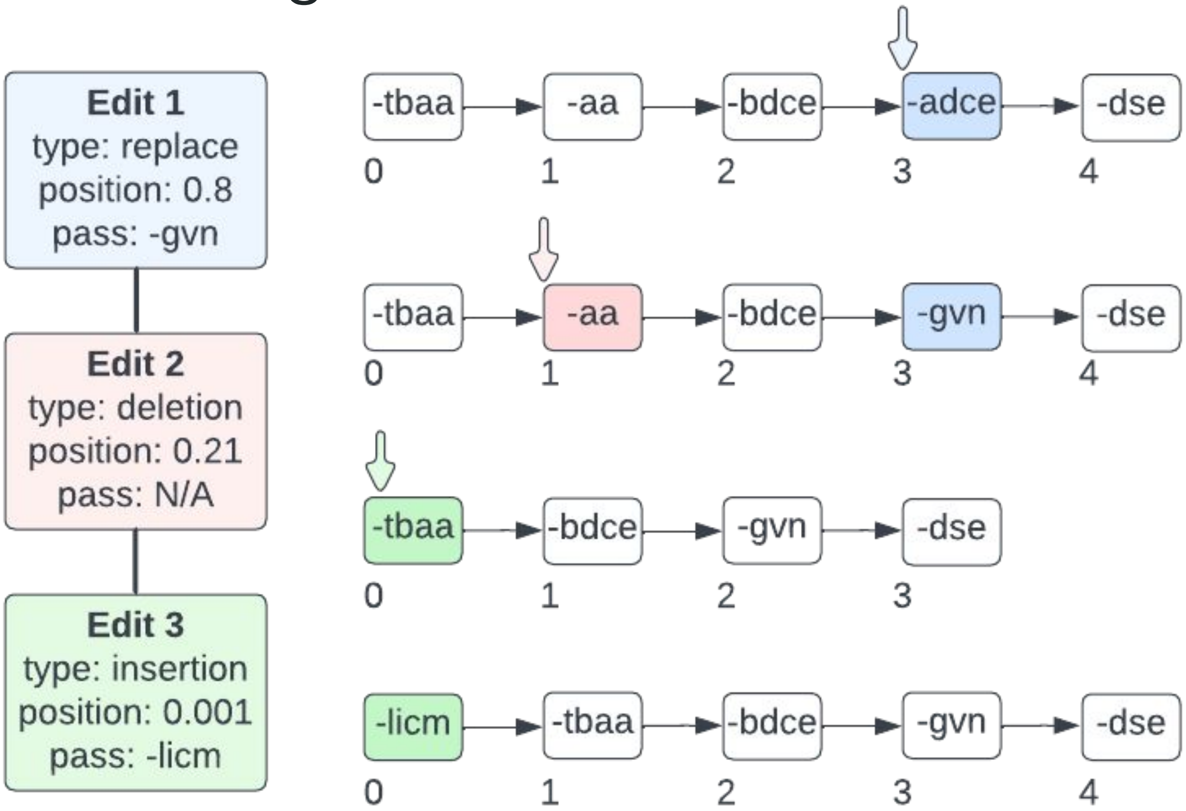
1. Edit type
2. Position
3. New Pass (null for deletion)

“Edit” Representations in Osaka List Structure



Edit ↔ Gene

Example Walk-through



Experimental Outline

a. Target program: Backtracker Algorithm for the Subset Sum Problem (SSP)

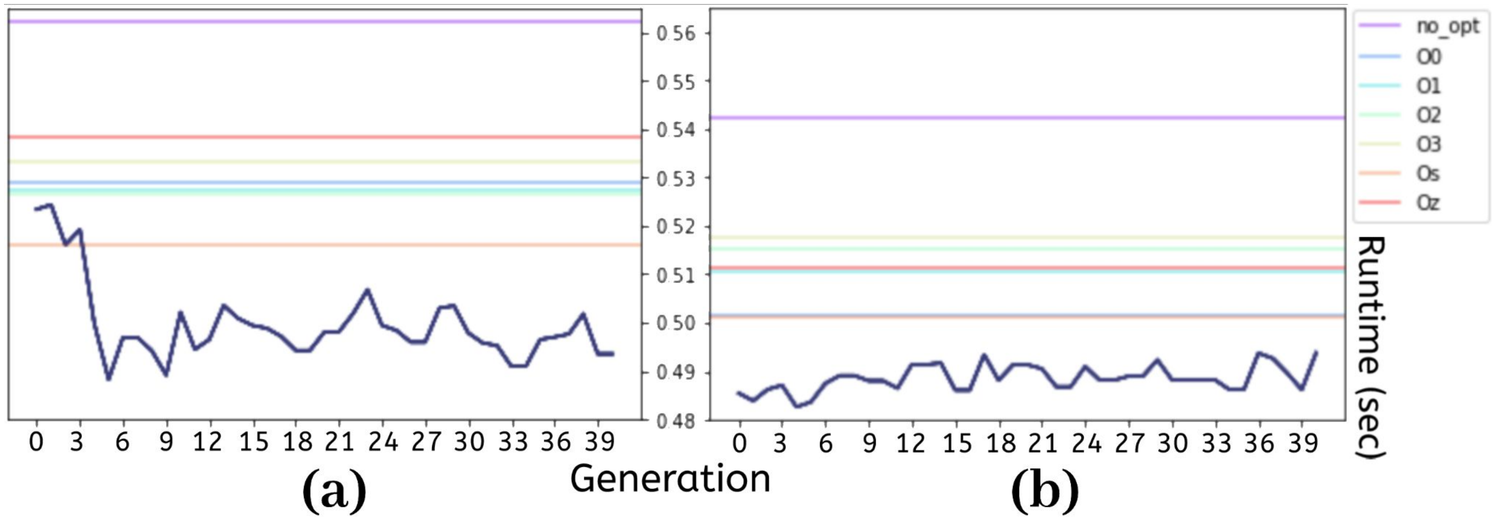
b. Hyperparameters:

num_generations	50
population_size	40
percent_crossover	60
percent_mutation	80
percent_elite	10
tournament_size	4
nest_size	6
individual_size	0 (this means random length)

c. 8 repeated runs

Results

Runtime Improvement: 3.7% (±0.8768)



Hoste, Kenneth and Eeckhout, Lieven (2008): 3.1%
Ashouri, Amir Hossein et. al. (2016): 4%, 2%
Ashouri, Amir Hossein et. al. (2017): 5%
Wang, Zheng and O'Boyle, Michael (2018): 5%

Efficiency Analysis

Entire search space: 10^167 ➜ would take decades!

Genetic Improvement: start from known solution (-O3) ➜ a few hours

Conclusions

- Don’t need domain expertise
- Problem-specific
- Efficient search process
- 3.7% runtime improvement

Future Directions

- Hyperparameter tuning specific for GI
- More test cases & standard benchmarks

thanks for listening!

Contact info:

Stella Li: sli136@jhu.edu
Hannah Peeler: hpeeler@utexas.edu
Andrew Sloss: andrew@sloss.net
Kenneth Reid: ken@kenreid.co.uk
Wolfgang Banzhaf: banzhafw@msu.edu