Development of optimization framework for embedded software based on automatic tuning of modern GCC via optimisation phases reordering

Speakers: Otrashchenko Aleksey, undergradute student
Akimov Zakhar, undergradute student
Supervisor: Nikolay Efanov, Ph.D., Associate Professor

Engineering & Telecommunications Conference,
November 22 – 23, 2023

Content

- Introduction
- General solution overview
- Genetic algorithm approach
- Reinforement algorithm approach
- Result & approach comparison
- Conclusion

Subject area overview

- Optimization of embedded systems source codes is a relevant problem; must be done with respect to several parameters (binary size, perfomance, etc.)
- Current compiler auto-tuning frameworks are based only on LLVM
- Embedded systems usually use GCC as toolchain

Embedded systems code optimization based on optimization phases reordering was not possible

Problem statement

1. Develop a method and framework for GCC toolchain, which will allow tuning of compiler optimization passes via reordering
2. Integrate developed framework into existing solutions for compiler auto-tuning with target being size reduction without runtime loss.

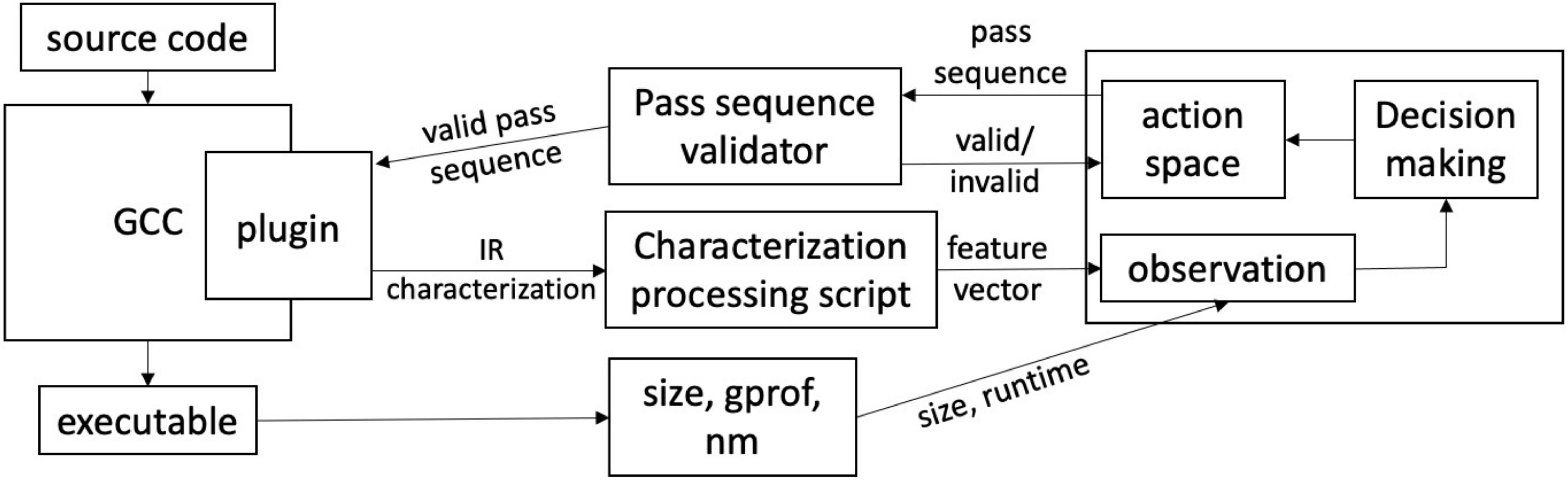
Objective function for program-granular pass sequence search:

$$size\big(prog(pass\_seq)\big)\Big|_{\Delta runtime(prog(pass\_seq),O2)<\varepsilon} \rightarrow min$$

Objective function for function-granular pass sequence search:

$$size\big(func(pass\_seq)\big)\Big|_{\Delta runtime(func(pass\_seq),O2)<\varepsilon} \rightarrow min$$

GCC optimization pass tuning framework

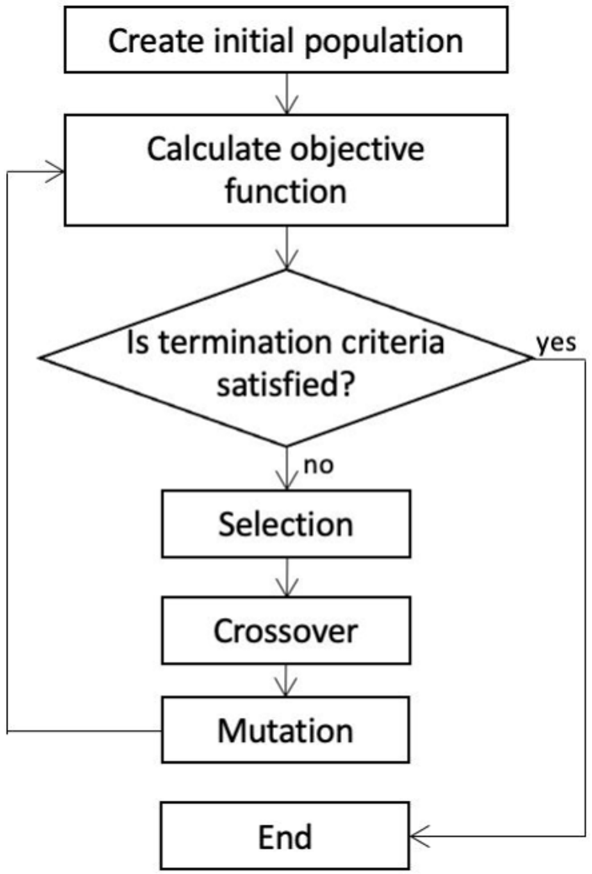


engineering &telecom

Auto-tuning approach #1
Genetic Algorithm

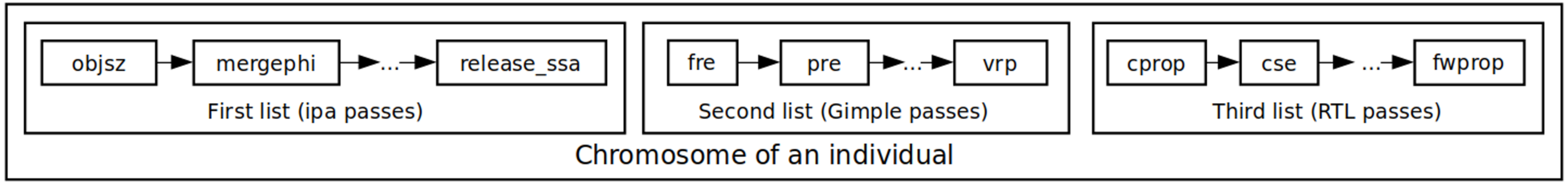
Genetic algorithm overview

- Initial Population: Usually is generated randomly, might be seeded in areas, where optimal solution might be found
- For each individual the fitness function is calculated, which defines the quality of given individual.
- If termination criteria is satisfied, the best individual of the current population is given as solution. Otherwise, the process continues
- The fitter individuals are chosen for reproduction
- For each new solution to be produced, a pair of parents is chosen. Parents' genes are combined through crossover process. Then, a mutation may happen to the resulting gene sequence
- Each individual's quality from resulting population is calculated via objective function and this goes on until termination criteria is satisfied

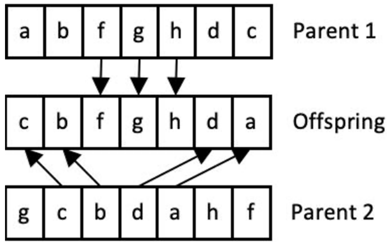


GCC GA implementation details

- Chromosome structure: The chromosome consists of 3 expertly chosen pass lists, that include IPA passes, general intra-procedural optimization passes, and RTL optimization passes.



- Crossover: The OX1 crossover method is used for each of lists in chromosome.



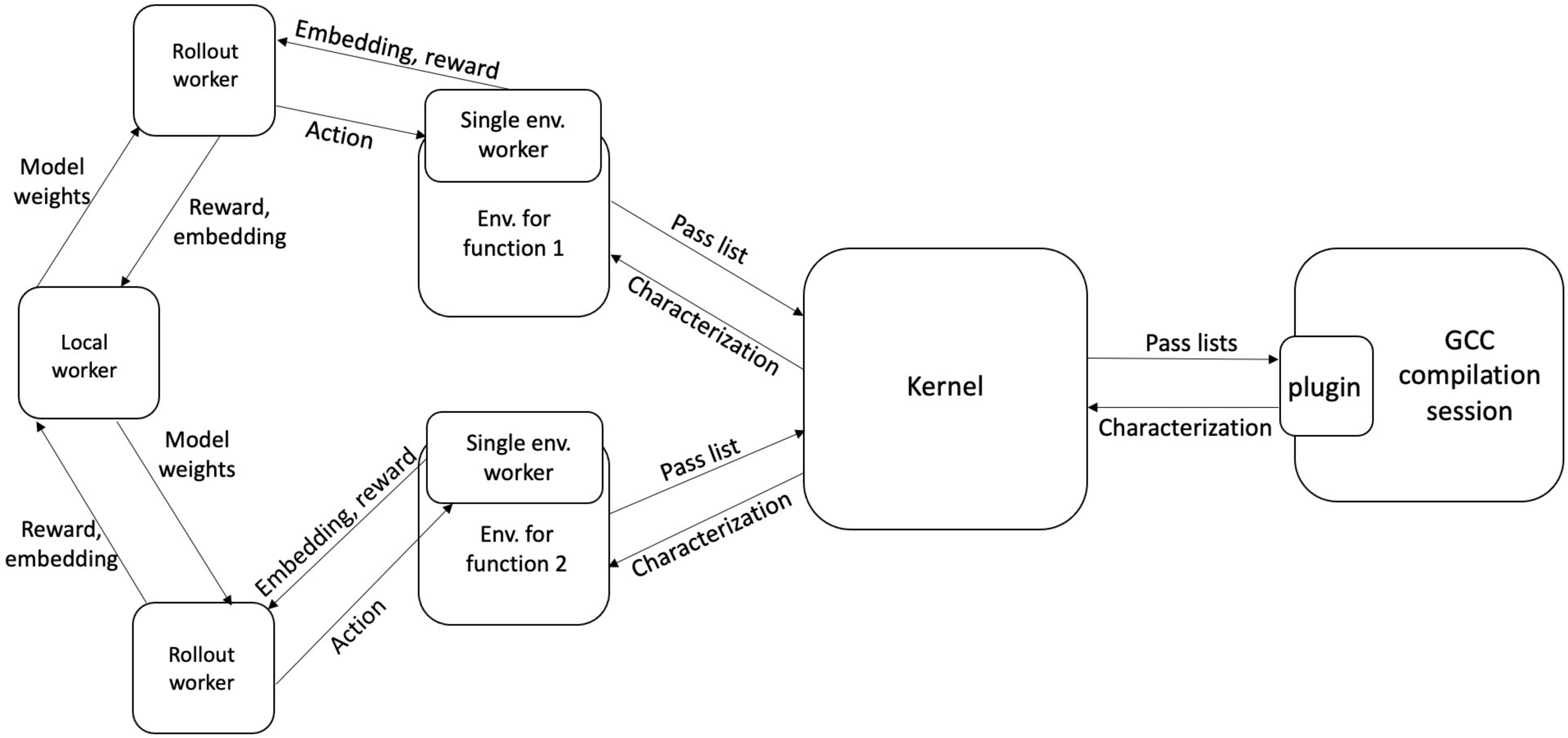
- Mutation: A removal of random existing pass / emplacement of a pass into place of previously removed pass was chosen as a mutation method.
- After both crossover and mutation the resulting pass sequence is checked for correctness
- Objective function: The objective function is calculated as follows (everything relative to GCC -O2):

$$\begin{cases} \textit{size reduction}, & \textit{size reduction} < 0 \\ \begin{cases} \textit{size reduction} - \textit{perf. loss}, & \textit{perf. loss} > 0 \\ \textit{size reduction}, & \textit{perf. loss} \leqslant 0 \end{cases} & \textit{size reduction} > 0 \end{cases}$$

- Stopping criteria: The search stops after 50 generations without change of objective function maximum

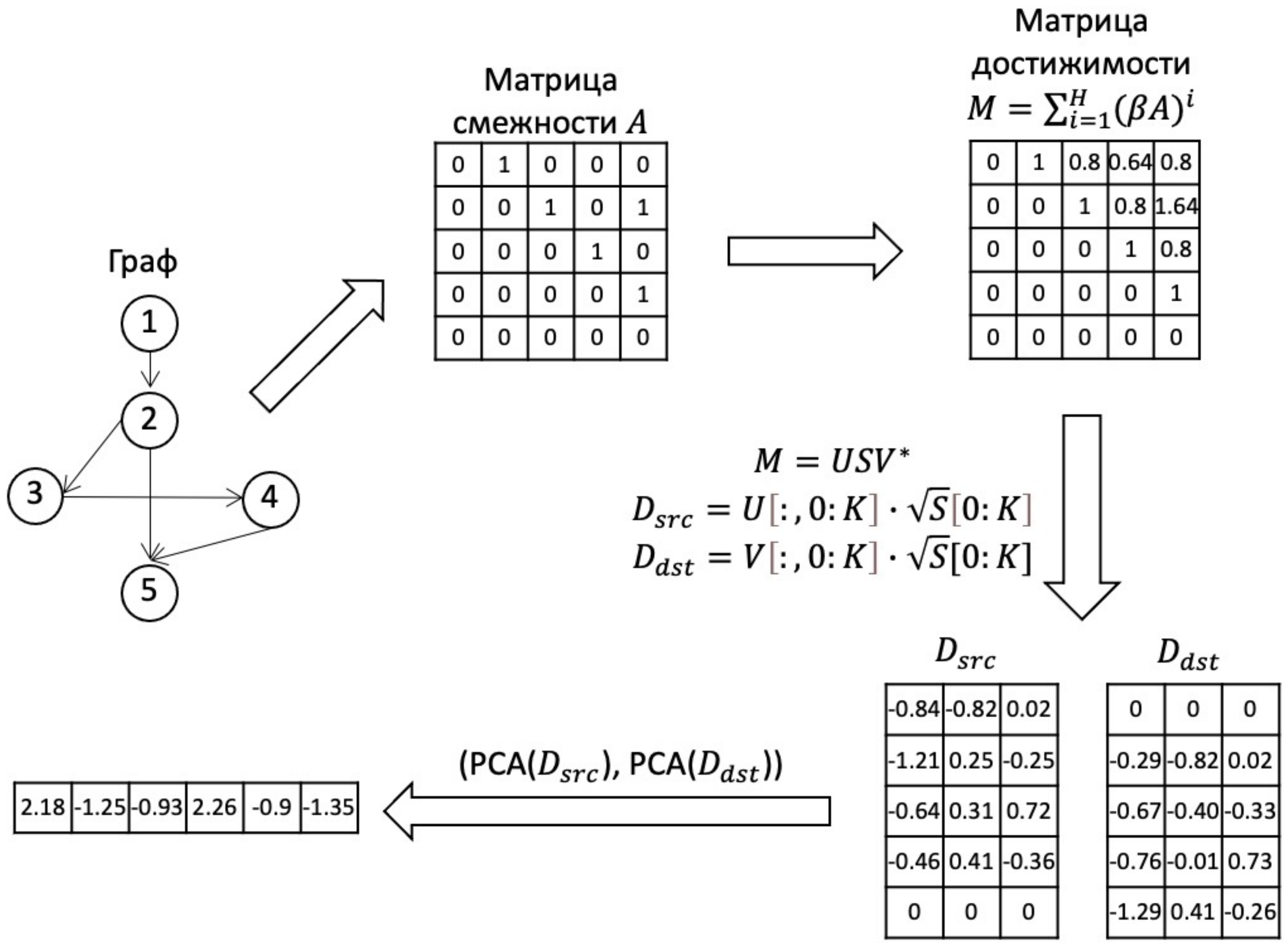
Auto-tuning approach #2
Reinforcement Learning

Incorporation to Ray/Rllib



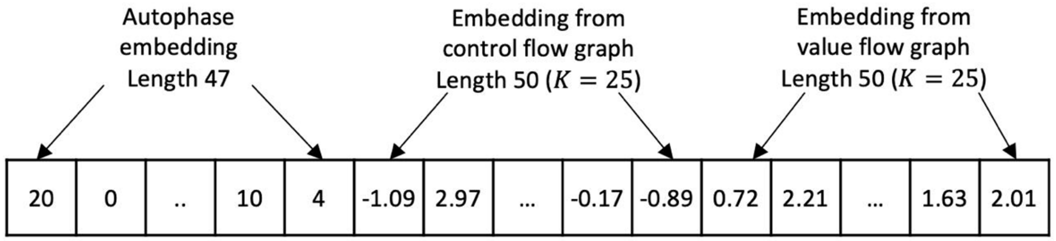
GCC IR embeddings

- GCC IR is characterised using autophase characterisation, control and value flow graphs
- Autophase characterisation consits of information about IR, available immediately during compilation
- The embedding from control flow graph and value flow graph are acquired as shown on the picture



Graph to embedding pipeline

Whole embedding:

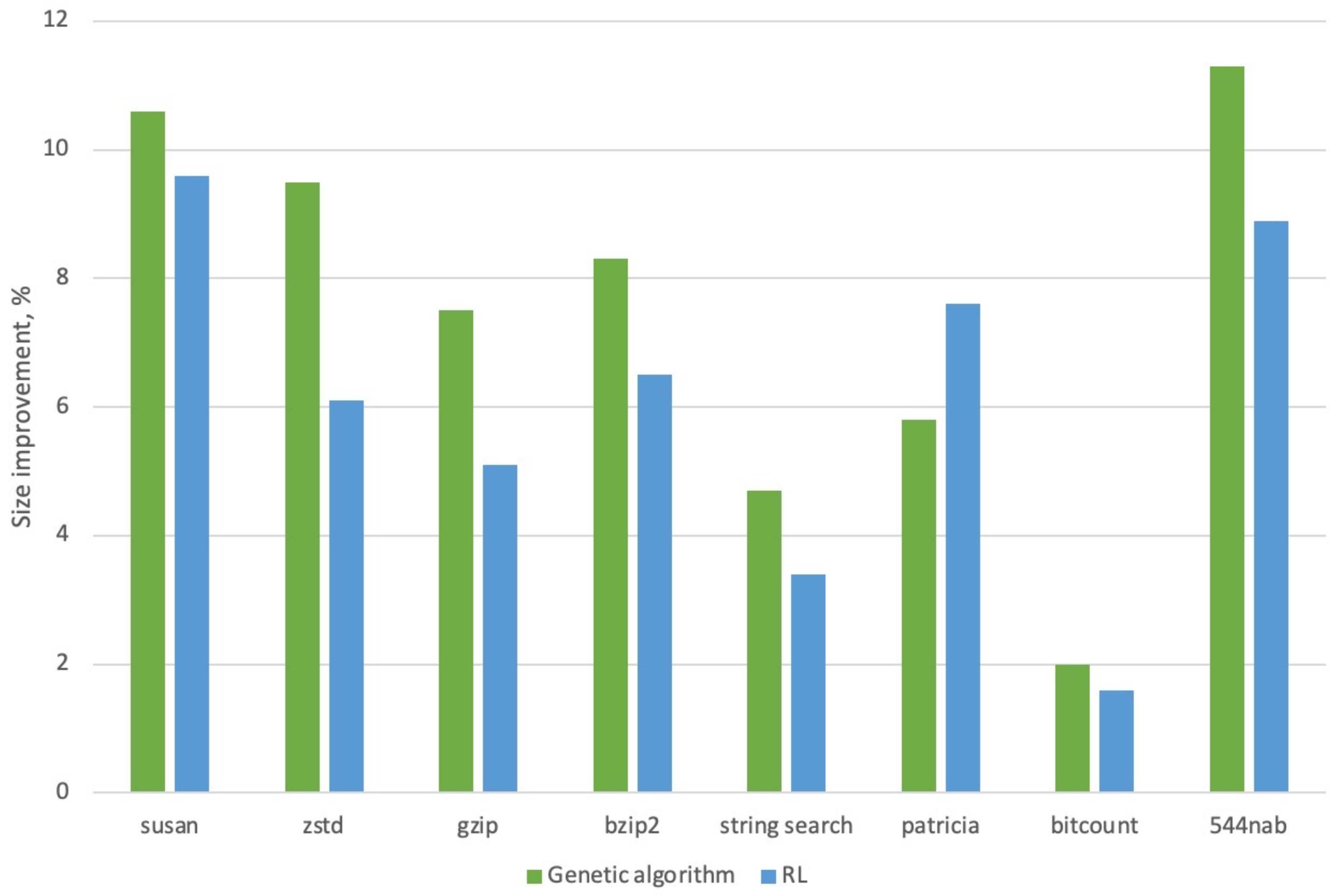


Results

Benchmark	RL size optimization, % compared to -O2	Genetic algorithm optimization, % compared to -O2
susan	9.6	10.7
zstd	6.1	9.5
gzip	5.1	7.5
bzip2	6.5	8.2
stringsearch	3.4	4.7
patricia	7.6	5.8
bitcount	1.6	2.0
544nab	8.9	12.0

engineering &telecom

Results comparison



engineering &telecom

Conclusion

- Size reduction up to ~10% was achieved with loss in runtime within error margin
- Genetic algorithm shows better results, but takes much more time to auto-tune the pass order and has no way to transfer knowledge between programs

Futher directions

- Implement genetic algorithm with function granularity
- Collect data from function-granular genetic algorithms runs for further use in supervised learning
- Apply new algorithms for reinforcement learning

engineering &telecom

Q&A and discussion section