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
&telecom

# Content

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

engineering
&telecom

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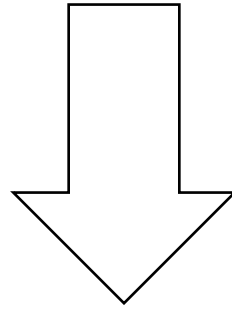
Embedded systems code optimization based on optimization phases reordering was not possible

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.
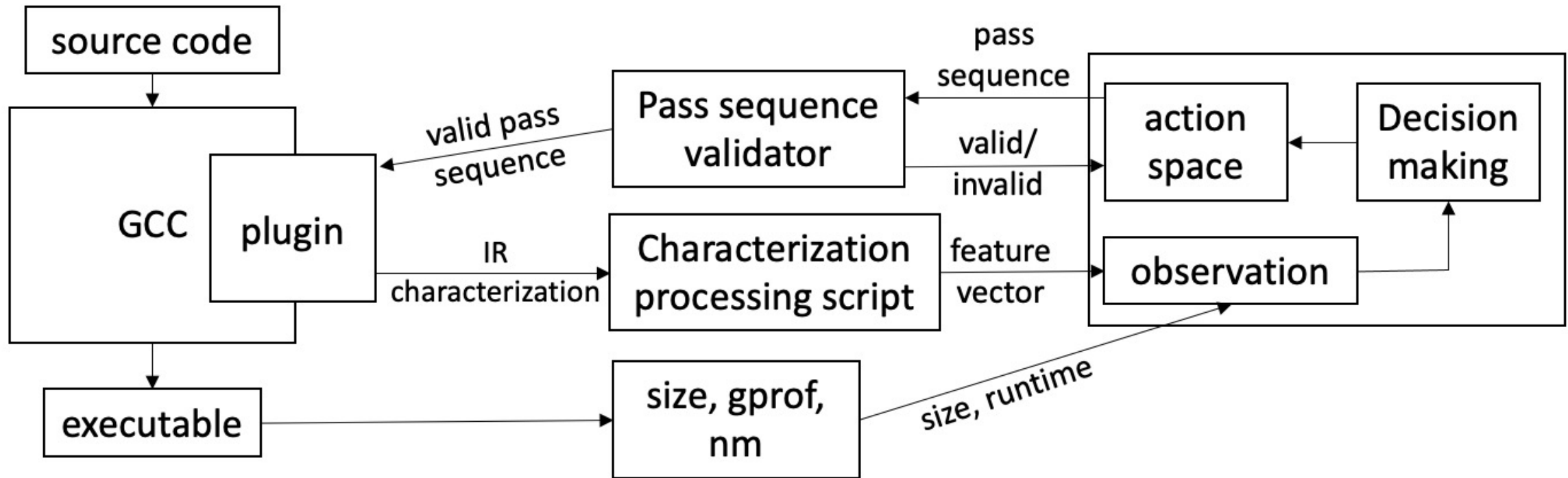
Objective function for program-granular pass sequence search:

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

Objective function for function-granular pass sequence search:

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

engineering
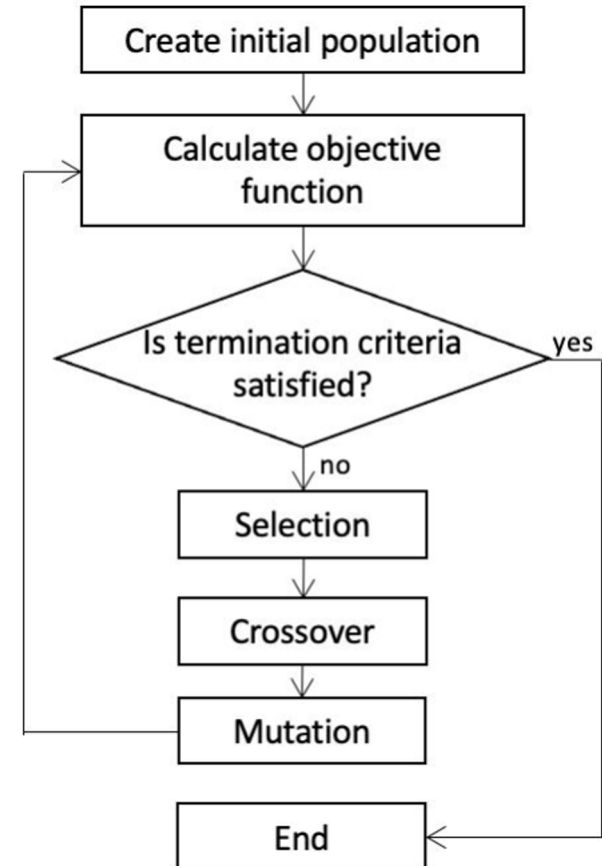&telecom

GCC optimization pass tuning framework

# 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

Create initial population

Calculate objective function

Is termination criteria satisfied? — yes

no

Selection

Crossover

Mutation

End

engineering
&telecom

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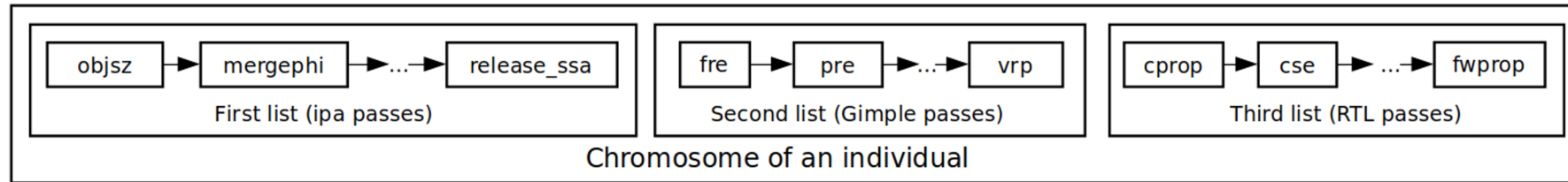
- Crossover: The OX1 crossover method is used for each of lists in chromosome.

- 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
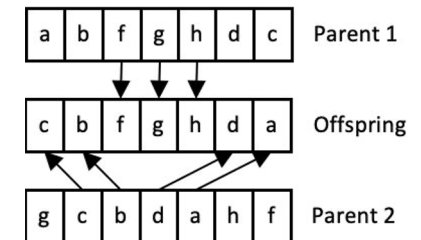
- Objective function: The objective function is calculated as follows (everything relative to GCC -O2):

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

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

Auto-tuning approach #2
Reinforcement Learning

Incorporation to Ray/RIIib
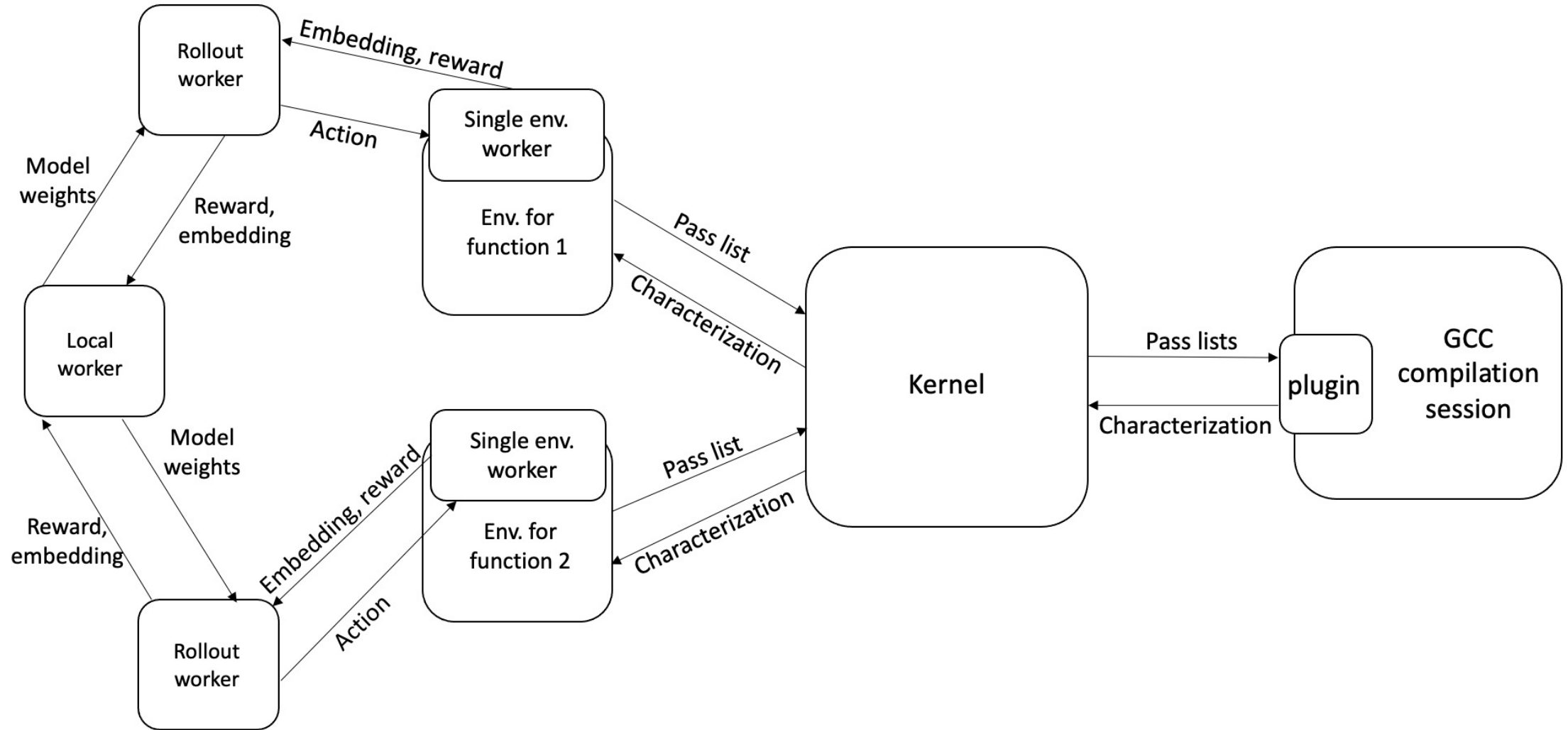
GCC IR embeddings

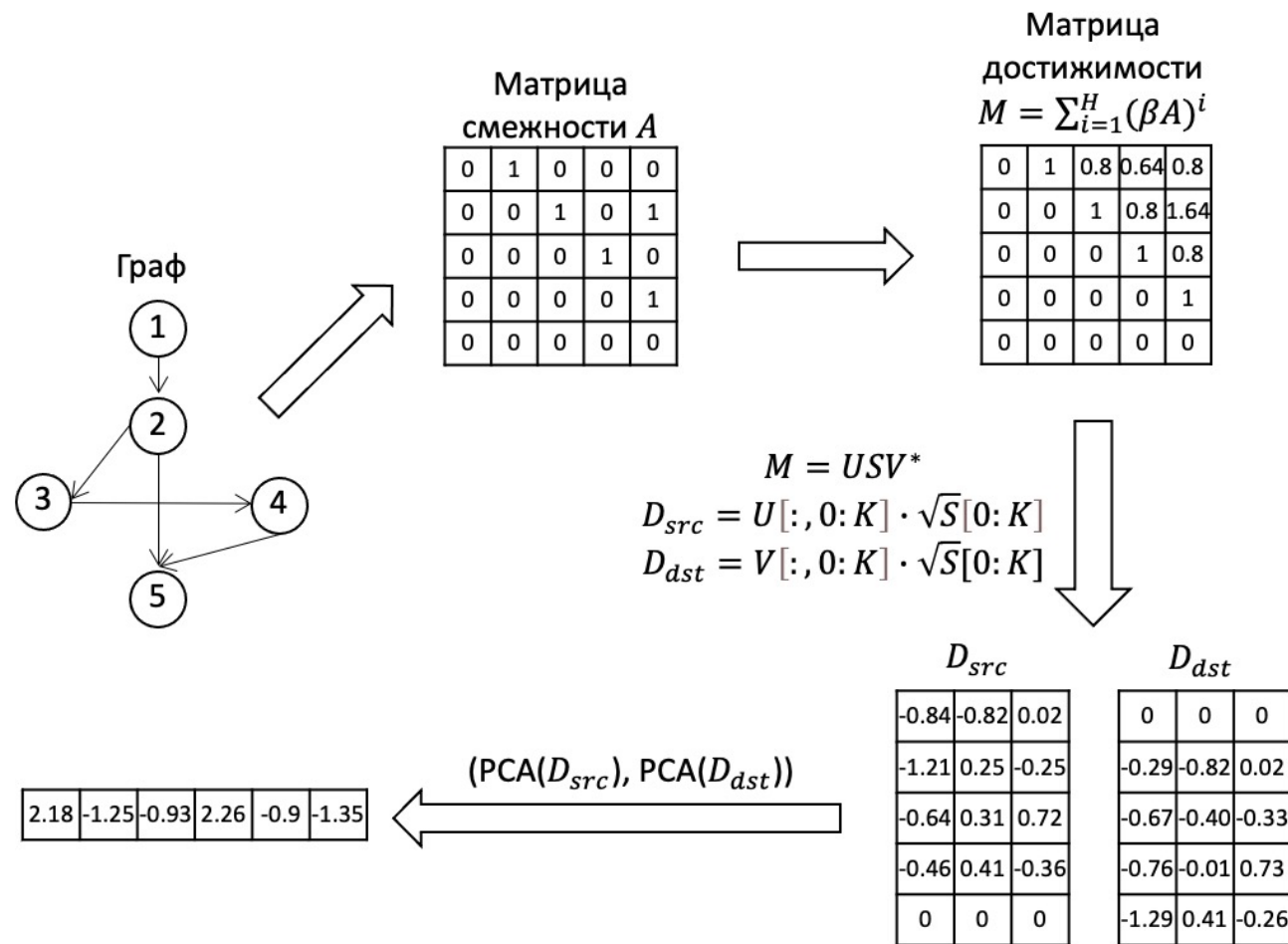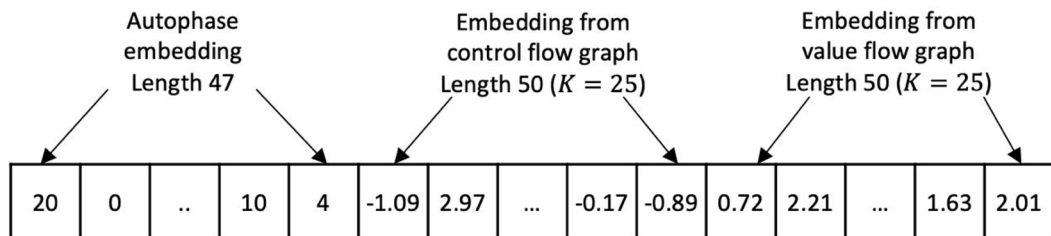- GCC IR is characterised using autophase characterisation, control and value flow graphs

- 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

Граф

Матрица смежности A

0	1	0	0	0
0	0	1	0	1
0	0	0	1	0
0	0	0	0	1
0	0	0	0	0

Матрица достижимости
$$M = \sum_{i=1}^{H}(\beta A)^i$$

0	1	0.8	0.64	0.8
0	0	1	0.8	1.64
0	0	0	1	0.8
0	0	0	0	1
0	0	0	0	0

$$M = USV^*$$
$$D_{src} = U[:, 0:K] \cdot \sqrt{S}[0:K]$$
$$D_{dst} = V[:, 0:K] \cdot \sqrt{S}[0:K]$$

D_{src}

-0.84	-0.82	0.02
-1.21	0.25	-0.25
-0.64	0.31	0.72
-0.46	0.41	-0.36
0	0	0

D_{dst}

0	0	0
-0.29	-0.82	0.02
-0.67	-0.40	-0.33
-0.76	-0.01	0.73
-1.29	0.41	-0.26

$(PCA(D_{src}), PCA(D_{dst}))$

| 2.18 | -1.25 | -0.93 | 2.26 | -0.9 | -1.35 |

Graph to embedding pipeline

Whole embedding:

| Autophase embedding Length 47 | | | | | Embedding from control flow graph Length 50 ($K = 25$) | | | | | Embedding from value flow graph Length 50 ($K = 25$) | | | | |

| 20 | 0 | .. | 10 | 4 | -1.09 | 2.97 | ... | -0.17 | -0.89 | 0.72 | 2.21 | ... | 1.63 | 2.01 |

engineering &telecom

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
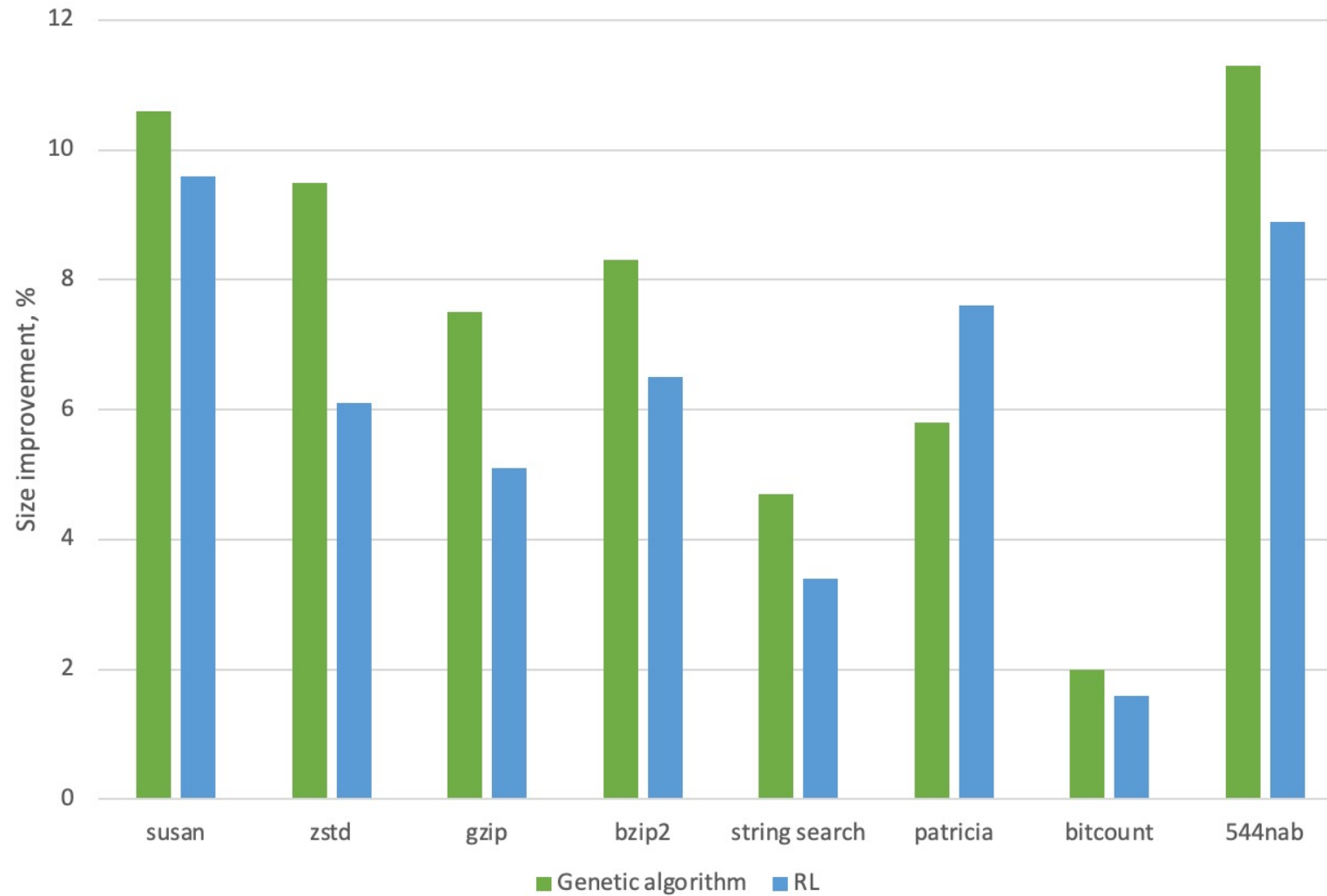
Conclusion

- 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

Q&A and discussion section