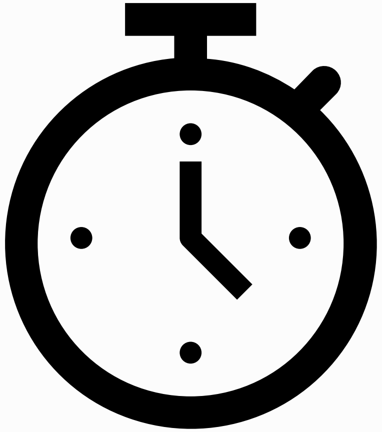
Empirical Comparison of Runtime Improvement Approaches: Genetic Improvement, Parameter Tuning and Their Combination

Thanatad Songpetchmongkol, Aymeric Blot, Justyna Petke

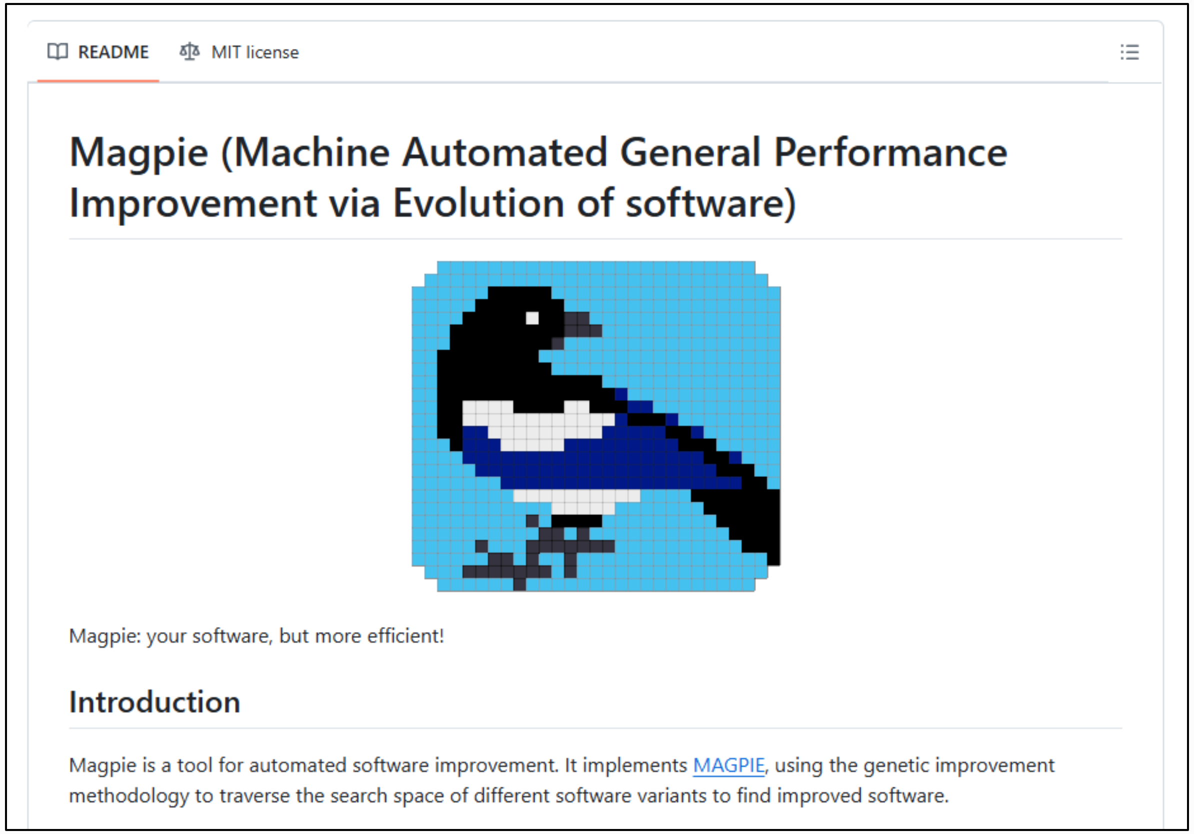
thanatad.songpetchmongkol.22@ucl.ac.uk



Software Optimisation

- Everyone uses software on a daily basis.
- To optimize existing software,
  - Configure compiler option(s)
  - Configure developer-exposed parameter(s)

    Algorithm Configuration (AC)
  - Directly manipulate source code → Genetic Improvement (GI)
- MAGPIE is a framework for improving functional and non-functional properties of software.
  - Combines 3 optimisation approaches (compiler, AC and GI) into a single tool.
  - Capable in optimising both AC and GI simultaneously.


README MIT license

Magpie (Machine Automated General Performance Improvement via Evolution of software)

Magpie: your software, but more efficient!

Introduction

Magpie is a tool for automated software improvement. It implements MAGPIE, using the genetic improvement methodology to traverse the search space of different software variants to find improved software.

https://github.com/bloa/magpie

Motivation

- MAGPIE applies First Improvement Local Seach (LS) to all 3 optimisation approaches.
- LS might not be the best search algorithm for all optimisation approaches.
- We aim to implement a well-performing search algorithm into MAGPIE.

Algorithm Selection

Selection Criteria

- Generalisable for both AC and GI domains.
- Shown to perform well on various benchmarks.
- Experimental time is realistically feasible.

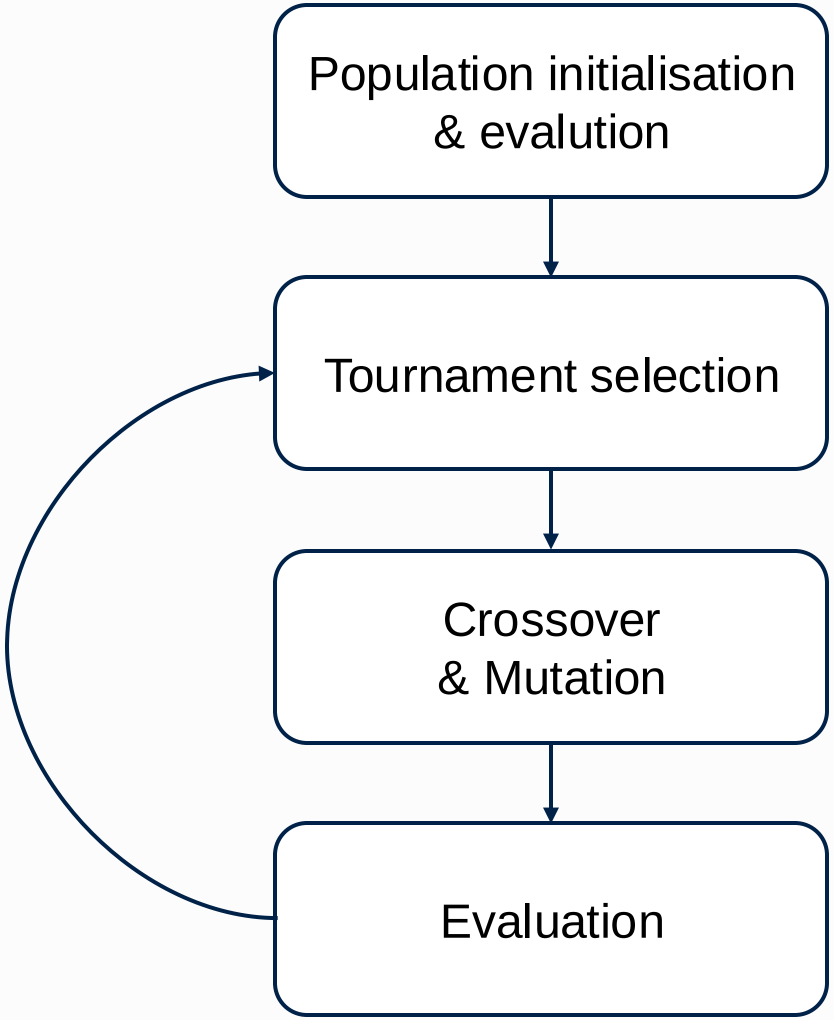
GI Domain

First Improvement Local Search (LS)

AC Domain

Genetic Algorithm (GA)

Genetic Algorithm



GA – Population Representation

	lbd-cut	var-decay	asymm	
individual 1	3	0.3	false	[StmtDelete(stmt1), StmtInsert(stmt4, stmt257)]
individual 2	10	0.45	true	[ ]
…				
individual N^{th}	5	0.9	true	[StmtReplace(stmt8, stmt36)]
	Parameter Edit (numerical / categorical)			Statement Edit

GA – Population Initialisation

- Mutate only N individuals, where N = population_size – default_size
- Value
  - Parameter Edit = default value
  - Statement Edit = empty patch
- Randomising a large number of parameter values at once results in uncompilable individuals

		lbd-cut	var-decay	asymm	
Default size	individual 1	5	0.8	false	[ ]
	individual 2	5	0.8	false	[ ]
	…				
Mutated	individual 10	9	0.33	true	[StmtDelet(stmt8)]

GA – Crossover

- First, check whether to perform crossover between two parents with 0.5 probability.
- Second, uniform crossover is performed between each gene with 0.5 probability.
  - Numerical parameter = random new numbers between existing values.
  - Categorical parameter = swap value *between parents*.
  - Statement edit = swap edit.

---

lbd-cut	asymm	
3	false	[StmtDelete(stmt1)]
10	true	[ ]

crossover

lbd-cut	asymm	
8	true	[ ]
5	false	[StmtDelete(stmt1)]

GA – Mutation

- First, check whether to perform mutation with 0.2 probability.
- Second, mutation is performed for each gene with 0.1 probability.
  - Parameter edit = random new value
  - Statement edit = remove
- Third, insert a new statement edit with 0.5 probability

---

	lbd-cut	asymm	
	5	false	[~~StmtDelete(stmt1)~~]
mutates	↓	↓	
	9	true	[StmtReplace(stmt43, stmt98)]

Benchmark

- MiniSAT_HACK_999ED_CSSC program with 25 exposed parameters.
- Target file = core/Solver.cc
- CircuitFuzz instances:
  - Training = 247 instances.
  - Testing = 277 instances.

SAT Formula

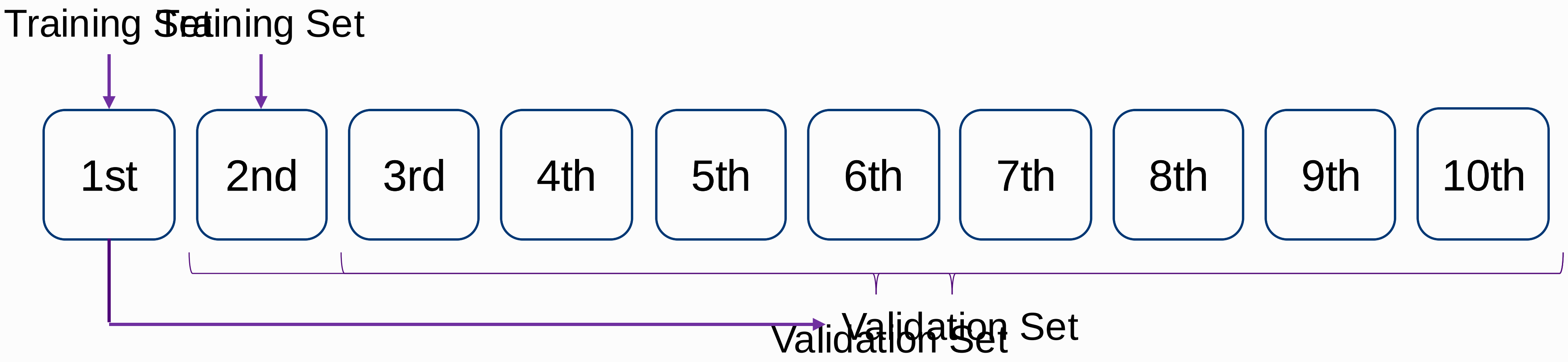
$$(x \vee y) \wedge (x \wedge \neg y) \wedge (\neg x \wedge y)$$

$$((x \wedge y) \vee z) \wedge (\neg x \vee y)$$

Experimental Protocol

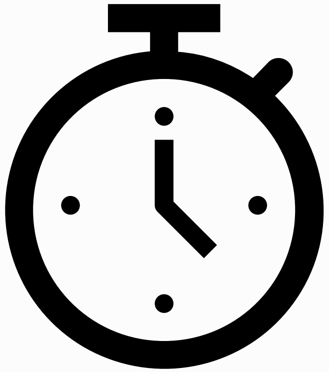
- Time budget of ~3 hours.
- K-fold cross-validation, where K = 10.
- Fitness function with Linux $perf command for # CPU instruction counts.

- 3 phases:
  - Training
  - Validation
  - Testing

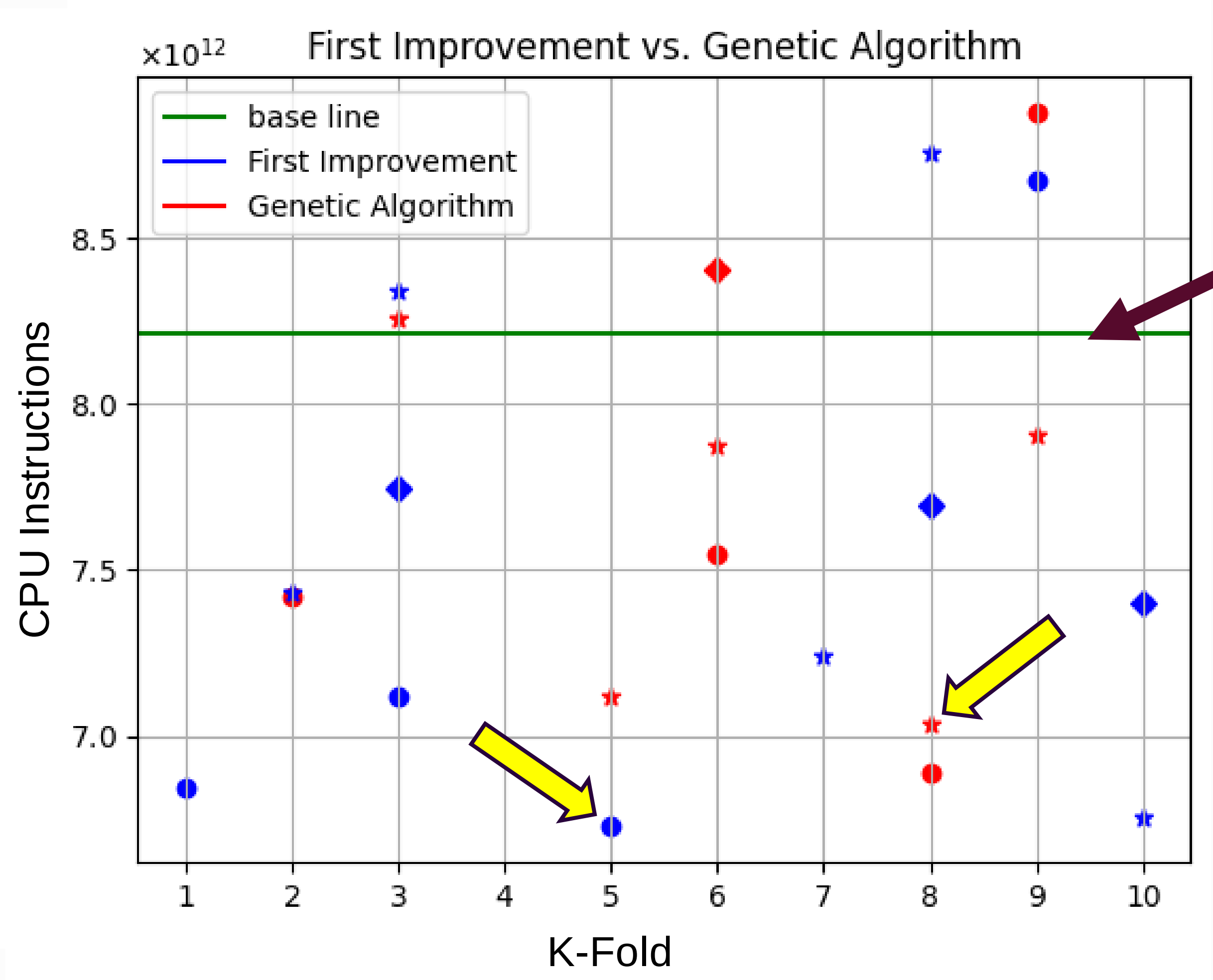


Research Questions

- RQ1: How effective are AC and GI at software performance improvement?
- RQ2: How effective is the simultaneous exploration of the joint search space of parameter values and software edits for runtime improvement?
- RQ3: Which search strategy is best for improving software performance?



RQ1: Algorithm Configuration vs. Genetic Improvement



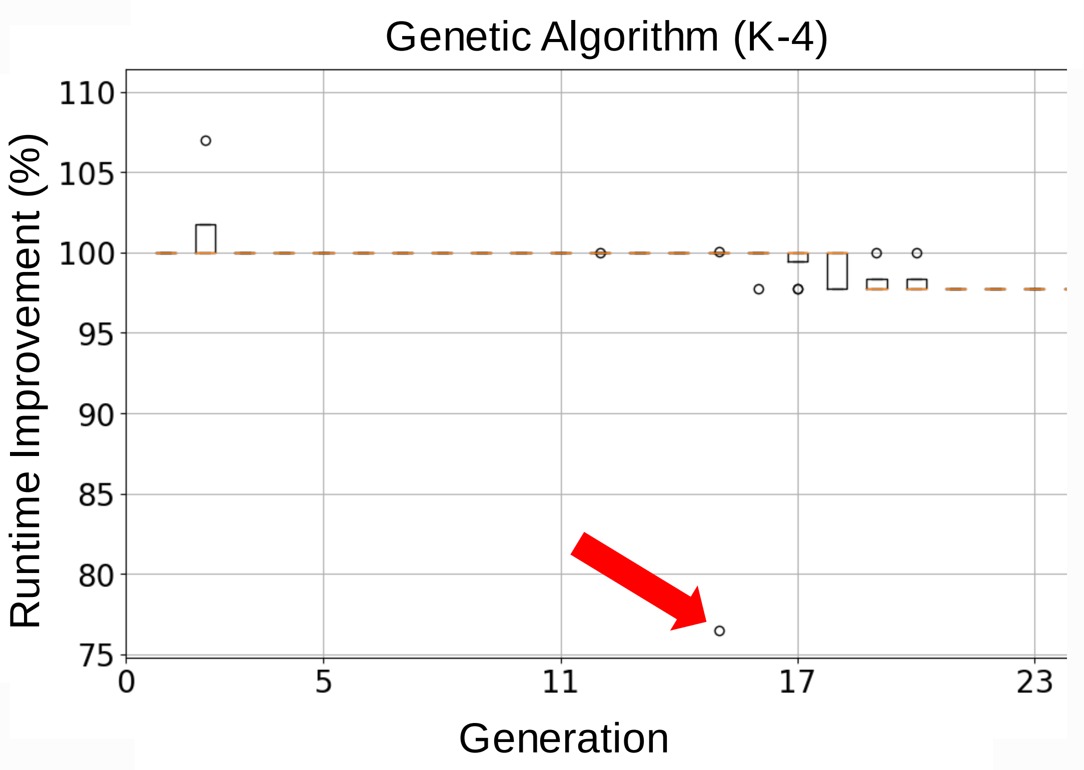
Original Program

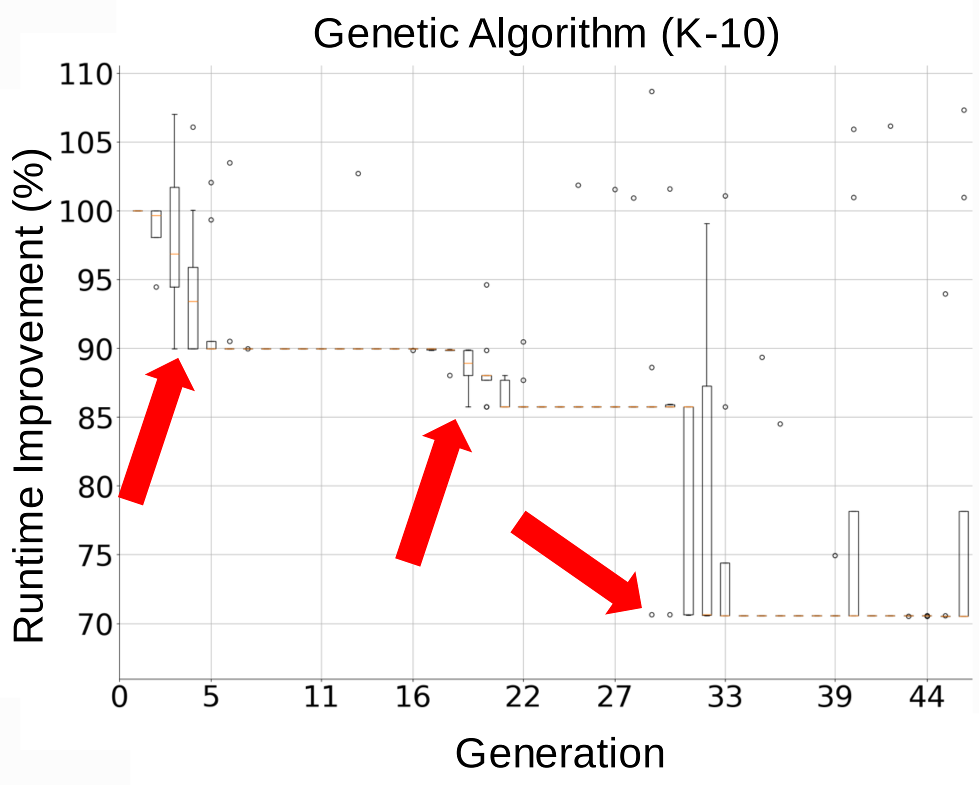
18.05% speed up with GI

14.32% speed up with AC

RQ1: Algorithm Configuration vs. Genetic Improvement

Elitism can help preserve and pass good individuals to the next generation.





RQ2: First Improvement Local Search vs. Genetic Algorithm for Joint Search Space

- LS leads to the best speed-up.
- We found that increasing population size led to better performance, for GA.
- Additional time might be necessary as the search space size increases.
- Edit's representation can influence the number of each type of edit.

Technique	Joint
LS	9.88% ★
GA (population 10)	N/A
GA (population 100)	5.59%

Technique	# Parameter Edit (avg)	# Statement Edit (avg)
LS	~4	~7
GA (population 10)	~10	~2

RQ3: Best Search Strategy in MAGPIE

- We conducted a code review on all statement edits of the best variants.
- We classified patches: correct and incorrect.
  - Manual inspection reveals no incorrect patch.
- Removal of an assert statement can lead to an error.

Rank	Technique	Speed-up
1	GI with LS	18.05%
2	AC with LS	17.75%
3	GI with GA	16.12%
4	AC with GA	14.32%
5	AC + GI with LS	9.88%
6	AC + GI with GA	N/A

Conclusion

- Simultaneous optimisation of parameters (AC) and code (GI) with state-of-the-art algorithms is possible
- The best improvement is from Genetic Improvement with Local Search (18.05%).
- Genetic Algorithm cannot find any improvement in the joint search space.
- Future work should explore addition of elitism and the increase in population size when navigating the joint search space of parameters and code.

Rank	Technique	Speed-up
1	GI with LS	18.05%
2	AC with LS	17.75%
3	GI with GA	16.12%
4	AC with GA	14.32%
5	AC + GI with LS	9.88%
6	AC + GI with GA	N/A

QR to our paper