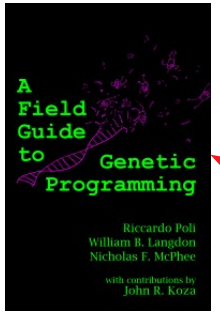
Genetic Improvement of LLVM Intermediate Representation

EuroGP 2023

W. B. Langdon

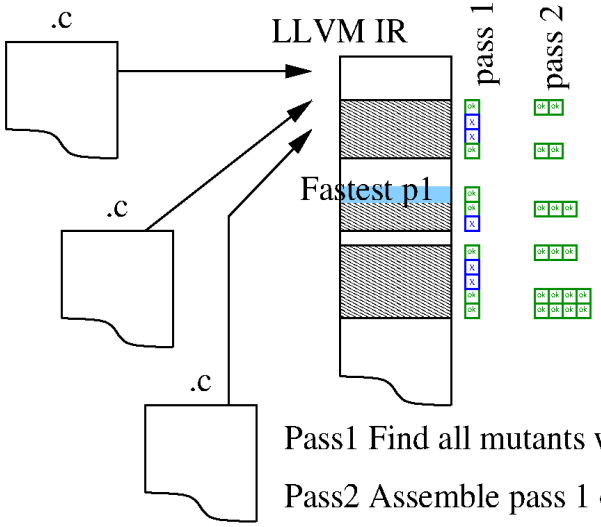
12 April 2023

Humies $10000 prizes
Submit by **Friday 2 June**



Free PDF
Free E-Book

Compile and link multiple C files to single LLVM IR



Pass1 Find all mutants which pass all tests

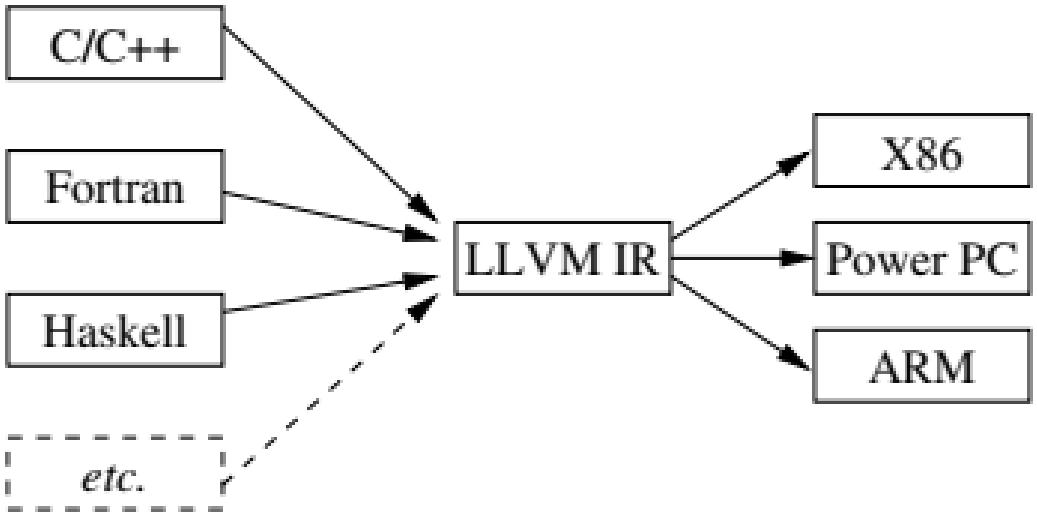
Pass2 Assemble pass 1 ok mutants to give best speed up

GI 2023
GI workshop
Saturday 20 May

Genetic Improvement of LLVM Intermediate Representation

- What is LLVM intermediate representation
- What is Genetic Improvement
- What we evolved
- Speedup compared to compiler optimiser
- Software is robust and can be evolved.

What is LLVM intermediate representation



LLVM supports 30+ programming languages and multiple types of computer,
by separating compiler front-end (source analysis) and back-end (binary code generation).
LLVM-IR links them.

What is Genetic Improvement

- GI applies search, usually genetic programming, to existing software
  - automatically fix bugs, speed up code, reduce energy consumption, bandwidth
-
  - Often C/C++ or Java source code
  - Java byte code, assembler, even machine code
  - Show evolution of LLVM intermediate code
- Given suitable representation and fitness measure any software can be evolved

Evolving LLVM

- LLVM intermediate representation created by Clang compiler `-S -emit-llvm`
- Designed to be optimised, then converted to machine code
- LLVM orginally C/C++, now ≈30 languages
- GI generate legal LLVM IR (delete only so far)
- Alternatives (may break LLVM-IR)
  - LLVM pass, eg llvm-mutate Eric Schulte
  - Grammatical Evolution, GPU Jhe-Yu Liou
  - Mutation testing

LLVM-IR

```
size_t OLC_CodeLength(const char* code, size_t size) {
  CodeInfo info;
  analyse(code, size, &info);
  return code_length(&info);
}

; Function Attrs: noinline nounwind optnone uwtable
define dso_local i64 @OLC_CodeLength(i8* noundef %0, i64 noundef %1) #0 {
  %3 = alloca i8*, align 8
  %4 = alloca i64, align 8
  %5 = alloca %struct.CodeInfo, align 8
  store i8* %0, i8** %3, align 8
  store i64 %1, i64* %4, align 8
  %6 = load i8*, i8** %3, align 8
  %7 = load i64, i64* %4, align 8
  %8 = call i32 @analyse(i8* noundef %6, i64 noundef %7, %struct.CodeInfo* noundef %5)
  %9 = call i64 @code_length(%struct.CodeInfo* noundef %5)
  ret i64 %9
}
```

Local variables

- Strongly typed (eg i32, i8*, double)
- Single-Static Assignment
- Numbered registers and labels (must be in order)
- `define }` delimit scope. Local registers start again at 0 in next function

Mutable LLVM-IR

- `store` remove whole IR line
- `call` remove,
  but if function also replace answer with 0
- Conditional branch. Force branch to left or to right
- Assignment. Replace register with 0
- Two passes. 1st locate scope boundaries.
  2nd replace all instances of deleted registers
- Representation is a list of changes

Representation list of deletions

List of mutations to lines. Separate with ;

Use : to indicate branch

% => local register number

`148:2;1895%6;1905:2;1895%40;1895%178;`

Force branch line 148 to 2nd label

In scope starting line 1895, delete %6

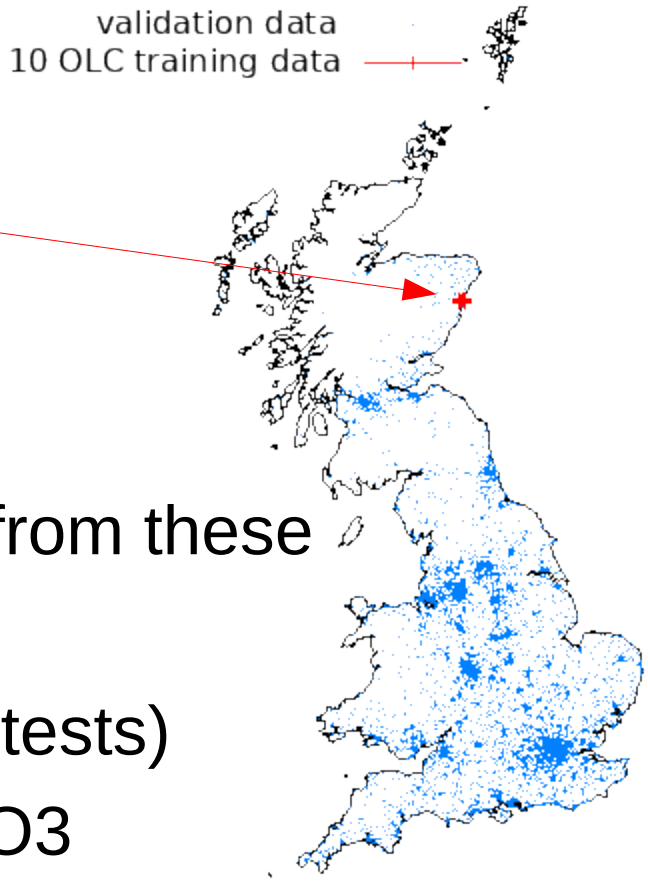
Force branch line 1905 to 2nd label

In scope starting line 1895, delete %40

In scope starting line 1895, delete %178

Google global OLC Fitness Test cases, Linux perf

- Convert latitude longitude pair into internal code (16 bytes)
- ten test cases
- Count CPU instructions using perf
- Mutate all LLVM one at a time
- Select only those that do not fail
- Best first search to select/assemble from these
- Demo on 10,000 locations
- Similar Uber H3 (13x more code, 40 tests)
- GI LLVM-IR without and with clang -O3

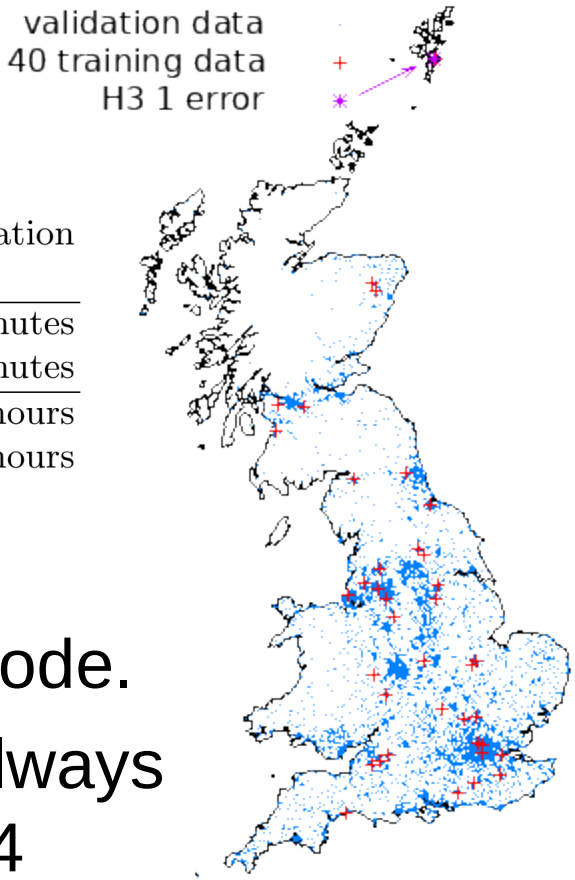


Uber global H3, Results

- H3 as OLC, but better test cases.
- H3 internal code (15 bytes)
- forty test cases

C	files	LOC	(used)	LLVM IR total	LLVM IR mutable	no output-change	Mutant size	Mutant speed up	Mutant holdout	GI duration
OLC	4	586	(127)	2546	294	141	2	698	682	5 minutes
-O3	4	586	(127)	2248	219	82	5	683	681	7 minutes
H3	43	5708	(1615)	19415	2113	955	51	2897	2631[a]	2.5 hours
-O3	43	5708	(1615)	15680	1762	1108	46	3272	2985	3.25 hours

[a] One holdout test failed

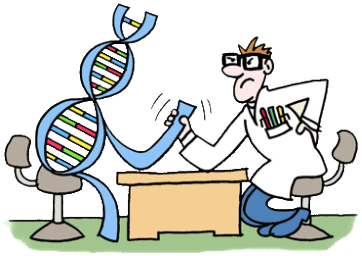


- OLC and H3 often remove redundant code.
- 10508%74 saves 872 instructions by always calling doCoords. (Zeroing register %74 removes condition before doCoords)

Conclusions

- Genetic Improvement has been applied to industrial code
- GI code has been adopted and is in use world wide
  - Eg used when designing covid tests
- GI code is under traditional human maintenance
- LLVM intermediate representation can be evolved
- other
  - Target bottle necks: profile.
  - Deal with noise: run multiple times, 1st quartile, perf
    - performance may not be stable: use differences
  - GI solutions tend to generalise. Maybe use co-evolution, perhaps source code analysis so fitness covers edge cases
- Code is not fragile

GI 2023



Genetic Improvement workshop

Hybrid Saturday 20 May

Human-Competitive results
total $10,000 prizes

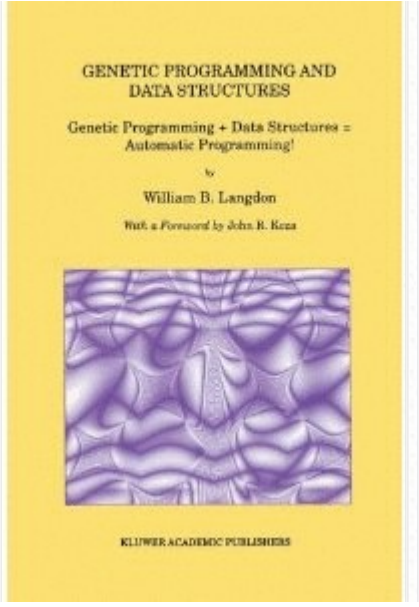
email your entry to
goodman@msu.edu
by friday 2nd June 2023

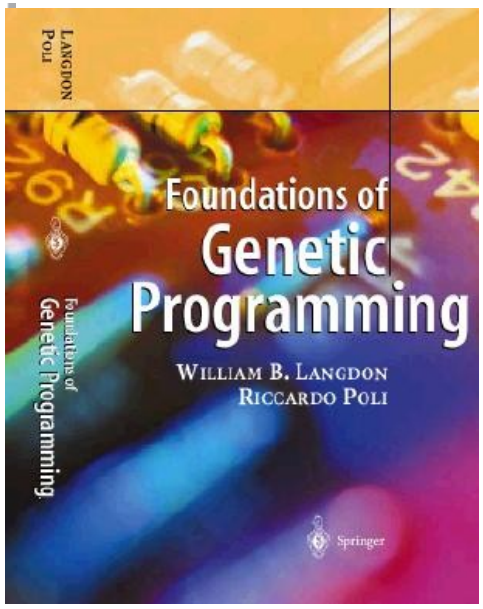
Automated Software Engineering
Special Issue on Genetic Improvement

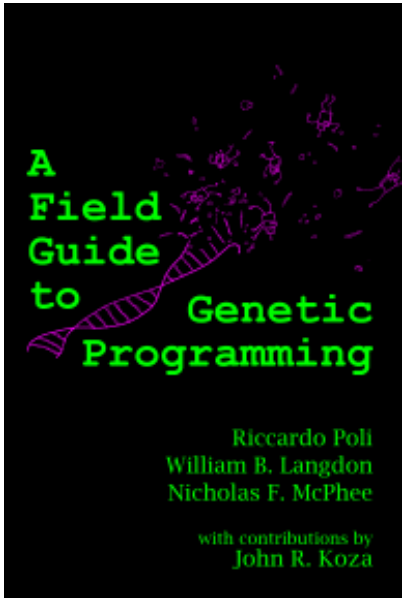
editors: Justyna Petke & Markus Wagner

Genetic Programming

W. B. Langdon





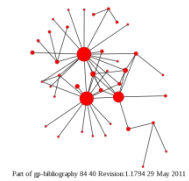


The Genetic Programming Bibliography

16261 references, 16000 authors

Make sure it has all of your papers!

E.g. email W.Langdon@cs.ucl.ac.uk or use | Add to It | web link

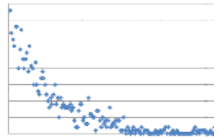


Co-authorship community.
Downloads

Downloads by day

A personalised list of every author's GP publications.

blog



Your papers

Googling GP bibliography, eg:
Evolutionary Medicine site:gpbib.cs.ucl.ac.uk