CREST

Genetic Improvement

Dagstuhl Seminar 15442

Approaches and Applications of Inductive Programming

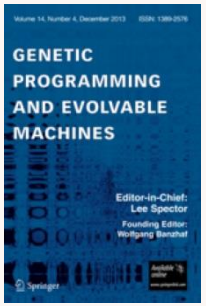
25-30 October 2015

W. B. Langdon

Department of Computer Science

UCL

Genetic Improvement special issue of Genetic Programming and Evolvable Machines, deadline 19 December 2015



Genetic Improvement

- Genetic Programming to improve human written programs
- Insights
- Examples
  - Automatic bug repair
    - 100s of real bugs, millions of lines of C/C++
  - Evolving 50000 lines of C++
    - Quality, speed, battery life tradeoffs
    - GPU code, up to 10000 faster

Genetic Improvement: Insights

- Work on industrial strength languages
- Focus search
- Evolve patches, change to C program source
- Evolve source code v. machine code
  - Ensure many patches/mutants compile
  - Software resilient to mutation
- Choose receptive domain
- Separate fitness from validation
  - Evolution exploits fitness
  - Present results on a slide, e.g. source code

Ensure many patches/mutants compile

- Create many patches/mutants
- Two common approaches
  - BNF grammar
  - abstract syntax tree

  Both ensure syntax {}; is correct. Main reason for not compiling is variable out of scope.
- Often faster to compile population of mutants than one at a time

Evolution exploits fitness

Computer does what you told it (not what you wanted)

- Do not assume no bugs because it looks ok
- Ensure guidance is in right direction
- Avoid over fitting
  - e.g. randomisation, such as DSS
- A 1 in 1000 chance will come up, do not let it trash your system or abort your GP.
  - A mutant which crashes should get low fitness not hang your evolutionary system
  - CPU and/or time limits (also 1994)
  - Sand boxing (perhaps virtual machines)

Present results on a slide

<for2_bt2_io_622><for2_bt2_io_278>

Mutation: replace 2nd part of for loop on line 622 with 2nd part of for loop on line 278

<bt2_io_278> ::= "for(uint32_t i = 0; i < this->_nPat; i++) {\n"
<bt2_io_622> ::= "for(uint32_t i = 0; i < offsLenSampled; i++) {\n"

Line 278 for(uint32_t i = 0; i < this->_nPat; i++) {
Line 622 for(uint32_t i = 0; i < offsLenSampled; i++) {

Original code

Line 622 for(uint32_t i = 0; i < this->_nPat; i++) {

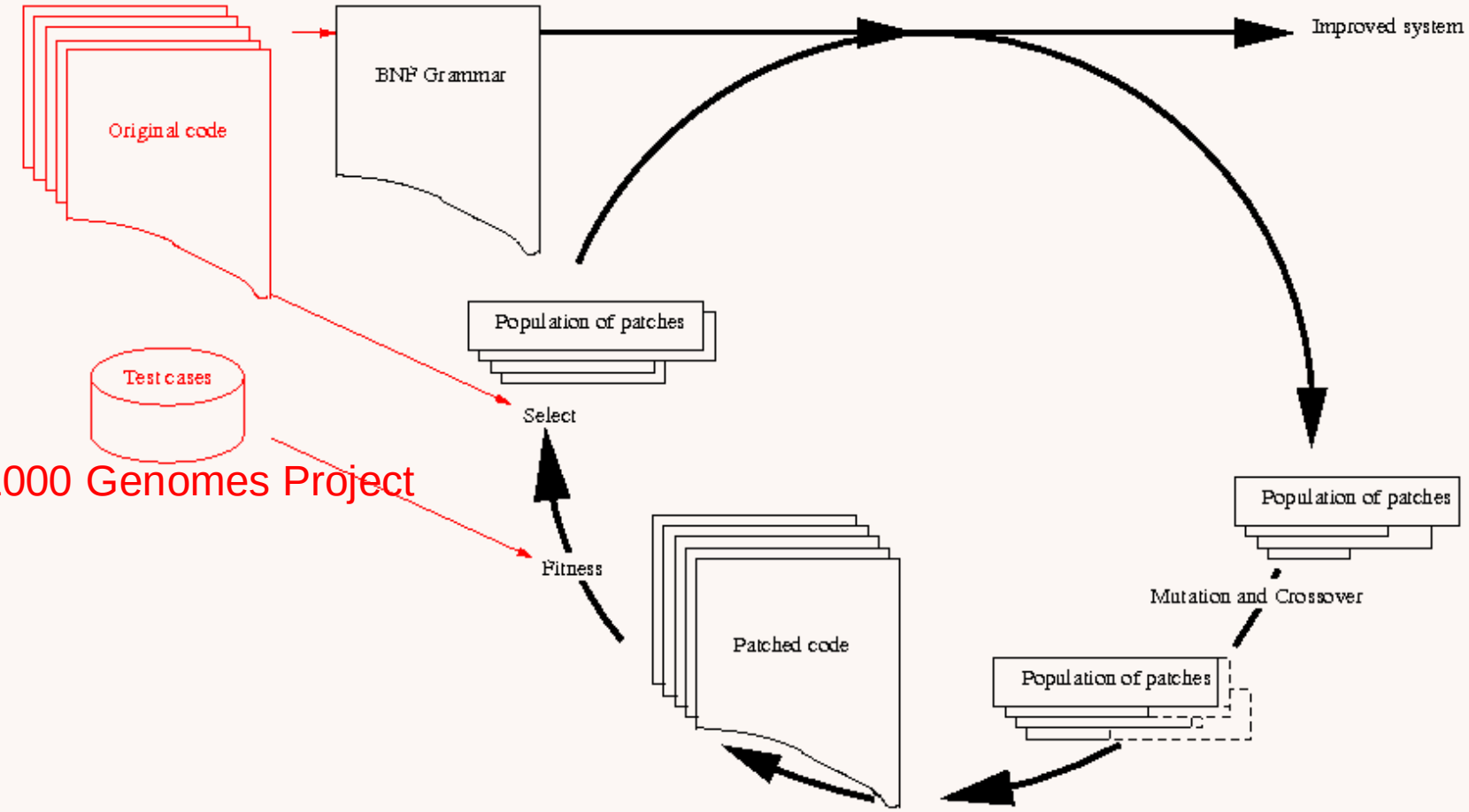
Code after mutation

Typically offsLenSampled=179,215,892 _nPat=84

Before mutation for loop lines 622-626 iterated 179,215,892 times, after only 84.
Obviously faster.
Exactly same result since lines 622-626 do nothing useful.

7 mutations make Bowtie2 more than 70 times faster

GP Evolving Patches

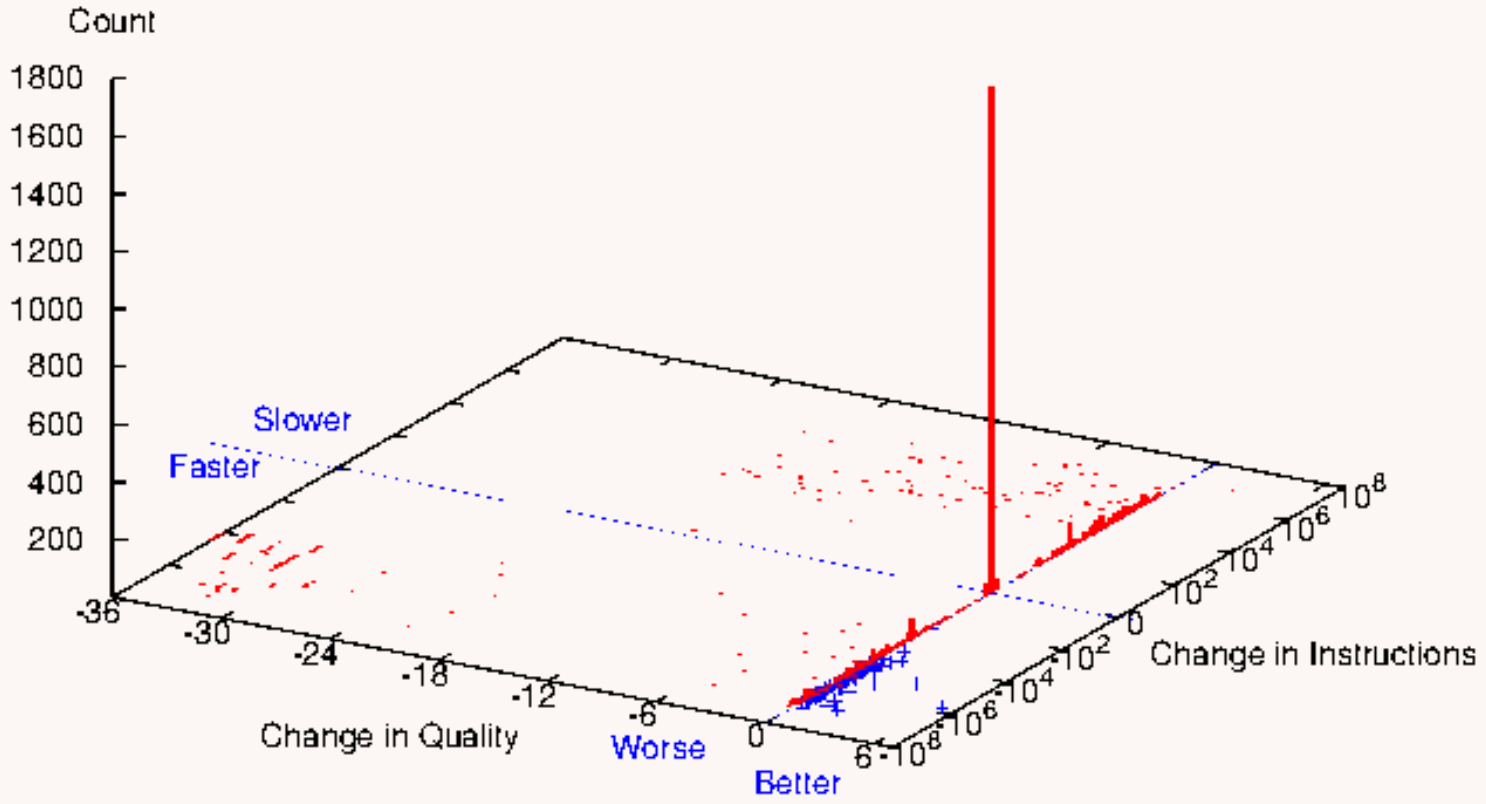


The 1000 Genomes Project

Software is not fragile

Software resilient to mutation

Trading performance against speed



10000 random mutation runs GISMO bowtie2, WBL 3 May 2012

Insights for Genetic Improvement

- Work on industrial strength languages, C/C++ Java
- Focus search, eg mutate only code which is used
- Evolve patches, small changes not whole code
- Evolve source code v. machine code
- Ensure many mutants compile
- Choose receptive domain, eg Bioinformatics
- Separate fitness from validation, validate after search
- Evolution exploits fitness, may have to update objective
- Present results on a slide
- Software is not fragile
  - break it, bend it, Evolve it

Be ambitious: do something impossible Free code

END

http://www.cs.ucl.ac.uk/staff/W.Langdon/ http://www.epsrc.ac.uk/ EPSRC

Genetic Improvement

W. B. Langdon

CREST

Department of Computer Science

Copies in Dagstuhl library