Applying Genetic Improvement to MiniSAT

Justyna Petke, William B. Langdon and Mark Harman

Centre for Research on Evolution, Search and Testing
University College London

Genetic Improvement Programming

Genetic Improvement Programming

Automatically improving a system's behaviour

Non-functional properties can be the criteria for improvement

Relies on a set of test cases

Genetic Improvement Programming

Automatically improving a system's behaviour

Non-functional properties can be the criteria for improvement

Relies on a set of test cases

Genetic Programming tries many possible options

Software designer chooses best

Motivation

Bowtie2:

Used for processing DNA sequences

50 000 lines of C++

DNA sequences from the “1000 Genome Project”

Motivation

Bowtie2:

Used for processing DNA sequences

50 000 lines of C++

DNA sequences from the “1000 Genome Project”

Only 7 lines of code changed in 3 C++ files

70+ faster on average

Very small improvement in Bowtie2 results

Bowtie2

CREST

"Optimising Existing Software with Genetic Programming"

William B. Langdon and Mark Harman

IEEE Transactions on Evolutionary Computation

to appear

Motivation

Motivation

Try another example:

Motivation

Try another example:

Easy to analyse

Motivation

Try another example:

Easy to analyse

Popular

Software chosen

SAT solver:

MiniSAT

SAT solver Applications

Bounded Model Checking

Planning

Software Verification

Automatic Test Pattern Generation

Combinational Equivalence Checking

Combinatorial Interaction Testing

and many others..

Representation: Move operations

Representation: Move operations

Grammar rule:

a line of code or a part of loop/condition (for, if, while, else)

Representation: Move operations

Grammar rule:

a line of code or a part of loop/condition (for, if, while, else)

Change code by re-using existing human written code:

Representation: Move operations

Grammar rule:

a line of code or a part of loop/condition (for, if, while, else)

Change code by re-using existing human written code:

Copy a line

Representation: Move operations

Grammar rule:

a line of code or a part of loop/condition (for, if, while, else)

Change code by re-using existing human written code:

Copy a line

Replace a line with another line from the program

Representation: Move operations

Grammar rule:

a line of code or a part of loop/condition (for, if, while, else)

Change code by re-using existing human written code:

Copy a line

Replace a line with another line from the program

Delete a line

Representation: Move operations

Grammar rule:

a line of code or a part of loop/condition (for, if, while, else)

Change code by re-using existing human written code:

Copy a line

Replace a line with another line from the program

Delete a line

Evolve a list of changes

BNF grammar

```
 <Solver_135>     ::=     "{Log_count64++;/*135*/} if" <IF_Solver_135> "  return false;\n"
<IF_Solver_135>  ::=     "(!ok)"
<Solver_138>     ::=     "" <_Solver_138> "{Log_count64++;/*138*/}\n"
<_Solver_138>    ::=     "sort(ps);"
<Solver_139>     ::=     "Lit p; int i, j;\n"
<Solver_140>     ::=     "for(" <for1_Solver_140> ";" <for2_Solver_140> ";" <for3_Solver_140> ") {\n"
<for1_Solver_140>        ::=     "i = j = , p = lit_Undef"
<for2_Solver_140>        ::=     "i < ps.size()"
<for3_Solver_140>        ::=     "i++"
```

Representation: Combining moves

Representation: Combining moves

Mutation: append another random change to the list

Representation: Combining moves

Mutation: append another random change to the list

Crossover: append lists from two parents

Representation: Combining moves

Mutation: append another random change to the list

Crossover: append lists from two parents

Only creating a new individual shortens the list

Representation: Combining moves

Mutation: append another random change to the list

Crossover: append lists from two parents

Only creating a new individual shortens the list

Selection: top half of the population was chosen

Fitness function

Fitness function

Run program and count lines used

Fitness function

Run program and count lines used

2 measures:

Fitness function

Run program and count lines used

2 measures:

Quality of answers produced (right/wrong, automatic oracle)

Fitness function

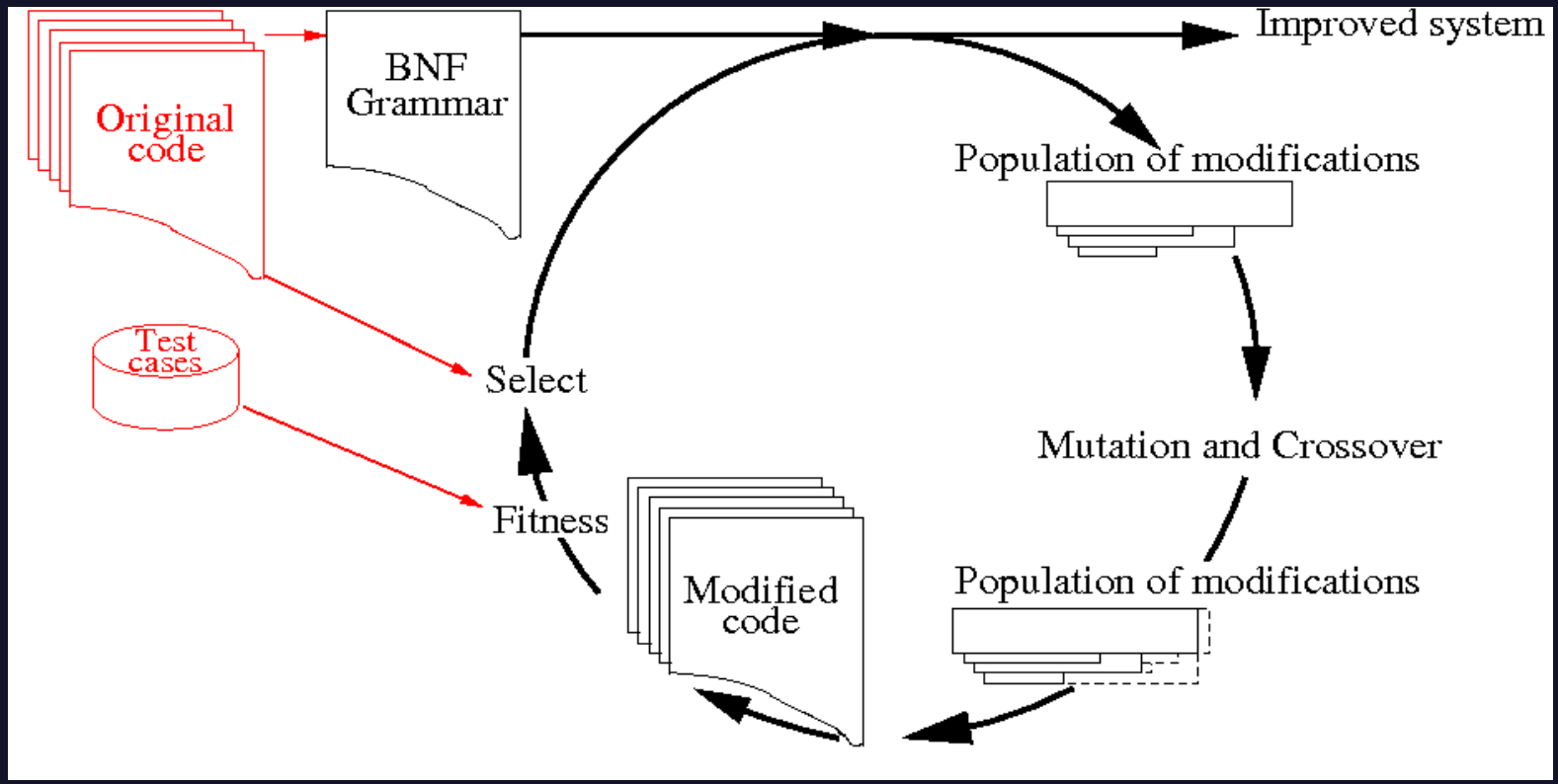
Run program and count lines used

2 measures:

Quality of answers produced (right/wrong, automatic oracle)

Resources used (number of lines used)

GP Improvement



MiniSAT

SAT solver

16 header files

6 C++ files

582 lines of C++ code in Solver.cc file

480 lines of C++ code in SimpSolver.cc file

BNF produces 648 lines that GP can manipulate

MiniSAT

SAT solver

16 header files

6 C++ files

582 lines of C++ code in Solver.cc file (all experiments)

480 lines of C++ code in SimpSolver.cc file (2^{nd} experim.)

BNF produces 648 lines that GP can manipulate

GP evolution parameters

GP evolution parameters

Training data set size: 71

GP evolution parameters

Training data set size: 71

Population size: 20

GP evolution parameters

Training data set size: 71

Population size: 20

Generations: 100

GP evolution parameters

Training data set size: 71

Population size: 20

Generations: 100

5 test examples, reselected every generation

GP evolution parameters

Training data set size: 107 (industrial)

Population size: 100

Generations: 20

5 test examples, reselected every generation

Results

Results

CREST

$\sim$ 14 hours

Results

CREST

- $\sim$ 14 hours
- $\sim$ 70% compiled

Results

CREST

$\sim$ 14 hours

$\sim$ 70% compiled

Very small improvements

Results

CREST

$\sim$ 14 hours

$\sim$ 70% compiled

Very small improvements

No clear winner so far..

Results

CREST

$\sim$ 14 hours

$\sim$ 70% compiled

Very small improvements

No clear winner so far..

Mainly stats and optimisations removed

Results

CREST

$\sim$ 14 hours

$\sim$ 70% compiled

Very small improvements ($\sim$2.5%)

No clear winner so far..

Mainly stats and optimisations removed

SAT example

$$x_1 \vee x_2 \vee \neg x_4$$
$$\neg x_2 \vee \neg x_3$$

- x_i : a Boolean variable
- x_i, $\neg x_i$: a literal
- $\neg x_2 \vee \neg x_3$: a clause

Example

```
bool Solver::satisfied(const Clause& c) const {
   for (int i = 0; i < c.size(); i++){
       if (value(c[i]) == l_True){
           return true;
       }
   }
   return false;
   }
```

Example

```
bool Solver::satisfied(const Clause& c) const {
   for (int i = 0; 0; i++){
       if (va lue(c[i]) == l_True){
           return true;
       }
   }
   return false;
   }
```

Summary

Summary

Genetic Improvement Programming:

Summary

Genetic Improvement Programming:

Automatically improves system behaviour

Summary

Genetic Improvement Programming:

Automatically improves system behaviour

According to some desired critaria using GP

Summary

Genetic Improvement Programming:

Automatically improves system behaviour

According to some desired critaria using GP

Bowtie2 : 70+ runtime improvement

Summary

Genetic Improvement Programming:

Automatically improves system behaviour

According to some desired critaria using GP

Bowtie2 : 70+ runtime improvement

MiniSAT : very small improvements so far..

Research directions

Research directions

Specialise test sets for GP

Research directions

Specialise test sets for GP

Change population and generation size

Research directions

Specialise test sets for GP

Change population and generation size

Discover historical changes using an older version of the solver (partially done)

Research directions

Specialise test sets for GP

Change population and generation size

Discover historical changes using an older version of the solver (partially done)

Target heavily used parts of MiniSAT (partially done)

Research directions

CREST

- Specialise test sets for GP
- Change population and generation size
- Discover historical changes using an older version of the solver (partially done)
- Target heavily used parts of MiniSAT (partially done)
- Allow to inject lines of code from other SAT solvers

Research directions

Specialise test sets for GP

Change population and generation size

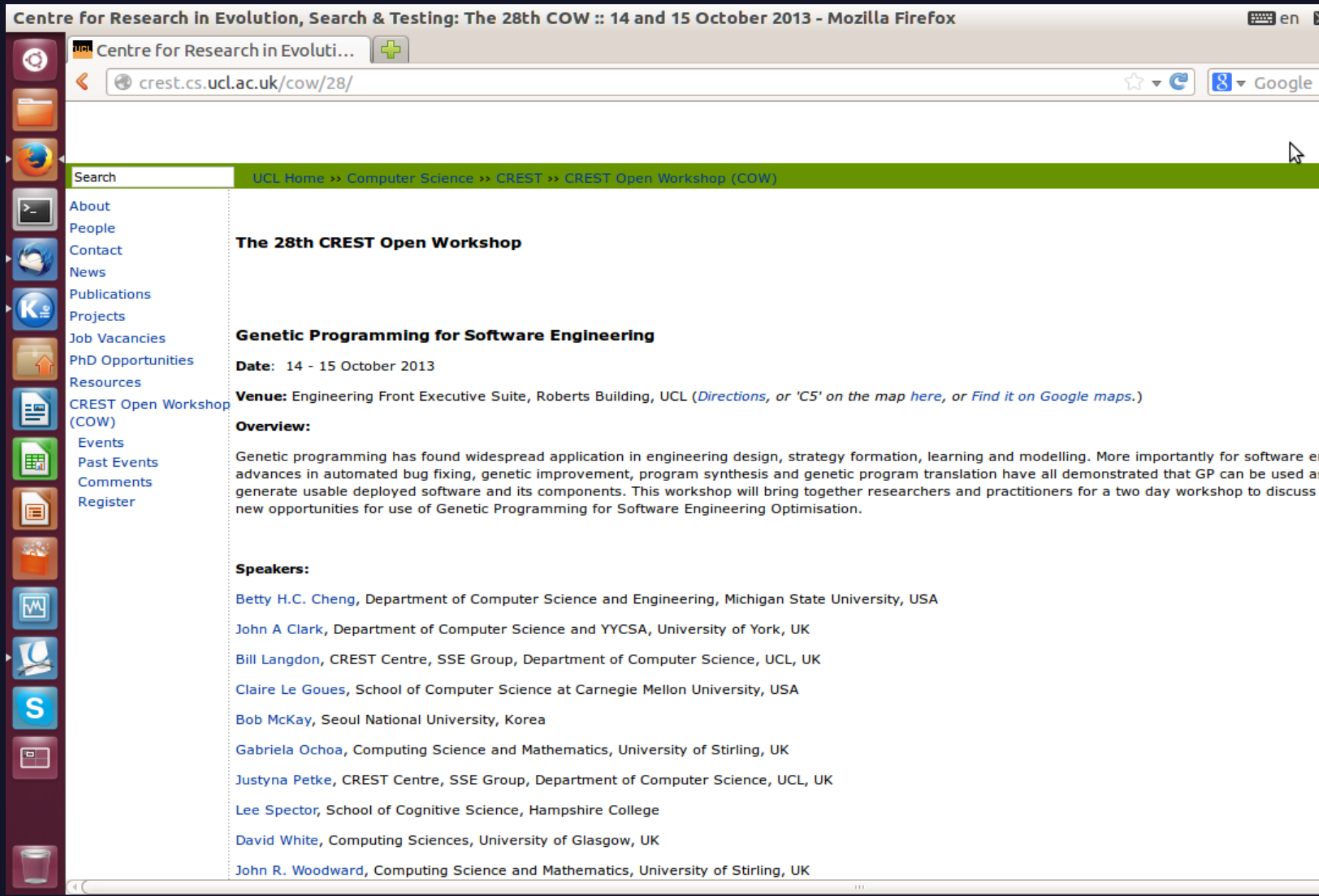
Discover historical changes using an older version of the solver (partially done)

Target heavily used parts of MiniSAT (partially done)

Allow to inject lines of code from other SAT solvers

Try to re-generate some functionality of MiniSAT

COW


Centre for Research in Evolution, Search & Testing: The 28th COW :: 14 and 15 October 2013 - Mozilla Firefox
Centre for Research in Evoluti...
crest.cs.ucl.ac.uk/cow/28/
Google
Search
UCL Home ›› Computer Science ›› CREST ›› CREST Open Workshop (COW)
About
People
Contact
News
Publications
Projects
Job Vacancies
PhD Opportunities
Resources
CREST Open Workshop (COW)
Events
Past Events
Comments
Register
The 28th CREST Open Workshop
Genetic Programming for Software Engineering
Date: 14 - 15 October 2013
Venue: Engineering Front Executive Suite, Roberts Building, UCL (Directions, or 'C5' on the map here, or Find it on Google maps.)
Overview:
Genetic programming has found widespread application in engineering design, strategy formation, learning and modelling. More importantly for software e
advances in automated bug fixing, genetic improvement, program synthesis and genetic program translation have all demonstrated that GP can be used a
generate usable deployed software and its components. This workshop will bring together researchers and practitioners for a two day workshop to discuss
new opportunities for use of Genetic Programming for Software Engineering Optimisation.
Speakers:
Betty H.C. Cheng, Department of Computer Science and Engineering, Michigan State University, USA
John A Clark, Department of Computer Science and YYCSA, University of York, UK
Bill Langdon, CREST Centre, SSE Group, Department of Computer Science, UCL, UK
Claire Le Goues, School of Computer Science at Carnegie Mellon University, USA
Bob McKay, Seoul National University, Korea
Gabriela Ochoa, Computing Science and Mathematics, University of Stirling, UK
Justyna Petke, CREST Centre, SSE Group, Department of Computer Science, UCL, UK
Lee Spector, School of Cognitive Science, Hampshire College
David White, Computing Sciences, University of Glasgow, UK
John R. Woodward, Computing Science and Mathematics, University of Stirling, UK