CREST

Using Genetic Improvement & Code Transplants to Specialise a C++ Program to a Problem Class

(EuroGP'14)

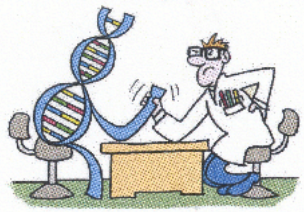
Genetic and Evolutionary Computation Conference

July 12-16, 2014
Vancouver, British Columbia

Winners of the
2014 Humies Silver Award:

Justyna Petke, Mark Harman,
William B. Langdon, Westley Weimer

Using Genetic Improvement and Code Transplants to Specialize a C++ Program to a Problem Class

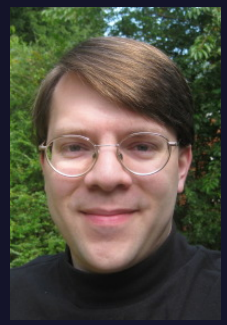


Association for Computing Machinery
Advancing Computing as a Science & Profession

sigevo

Authors



Justyna Petke Mark Harman William B. Langdon Westley Weimer

Centre for Research on Evolution, Search and Testing

University College London

University of Virginia

Genetic Improvement

Seeks to automatically improve an existing program

Criteria can be non-functional properties of the system

Uses genetic programming

Relies on a set of test cases

Question

Can we improve the efficiency of an already highly-optimised piece of software using genetic programming?

Contributions

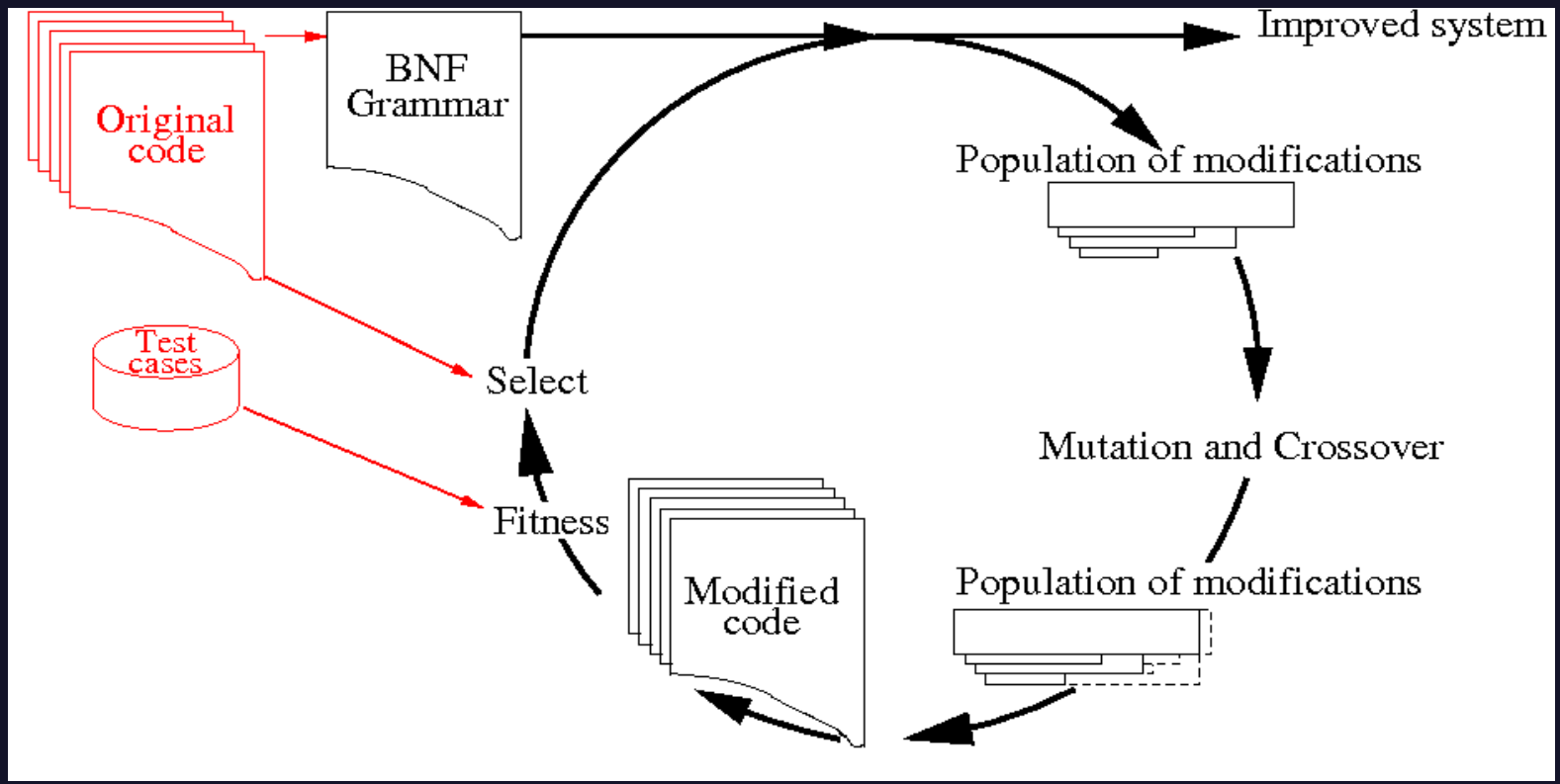
Introduction of multi-donor software transplantation

Contributions

Introduction of multi-donor software transplantation

Use of genetic improvement as means to specialise software

Genetic Improvement



Program Representation

Changes at the level of lines of source code

Each individual is composed of a list of changes

Specialised grammar used to preserve syntax

Example

```
 <Solver_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++"
```

Code Transplants

GP has access to both:

- the *host* program to be evolved
- the *donor* program(s)

Code Transplants

GP has access to both:

- the *host* program to be evolved
- the *donor* program(s)

code bank contains all lines of source code GP has access to

Mutation

Addition of one of the following operations:

DELETE

COPY

REPLACE

Example

```
<_Solver_135>

<_Solver_138>+<_Solver_140>

<for3_Solver_140><for3_Solver_836>
```

Crossover

Concatenation of two individuals

by appending two lists of mutations

```
<_Solver_135>

<_Solver_138>+<_Solver_140>

----------------------------------------

 <_Solver_135>  <_Solver_138>+<_Solver_140>
```

Fitness

Based on solution quality and

Efficiency in terms of lines of source code

Fitness

Based on solution quality and

Efficiency in terms of lines of source code

Avoids environmental bias

Fitness

Test cases are sorted into groups

One test case is sampled uniformly from each group

Fitness

Test cases are sorted into groups

One test case is sampled uniformly from each group

Avoids overfitting

Selection

Fixed number of generations

Fixed population size

Initial population contains single-mutation individuals

Selection

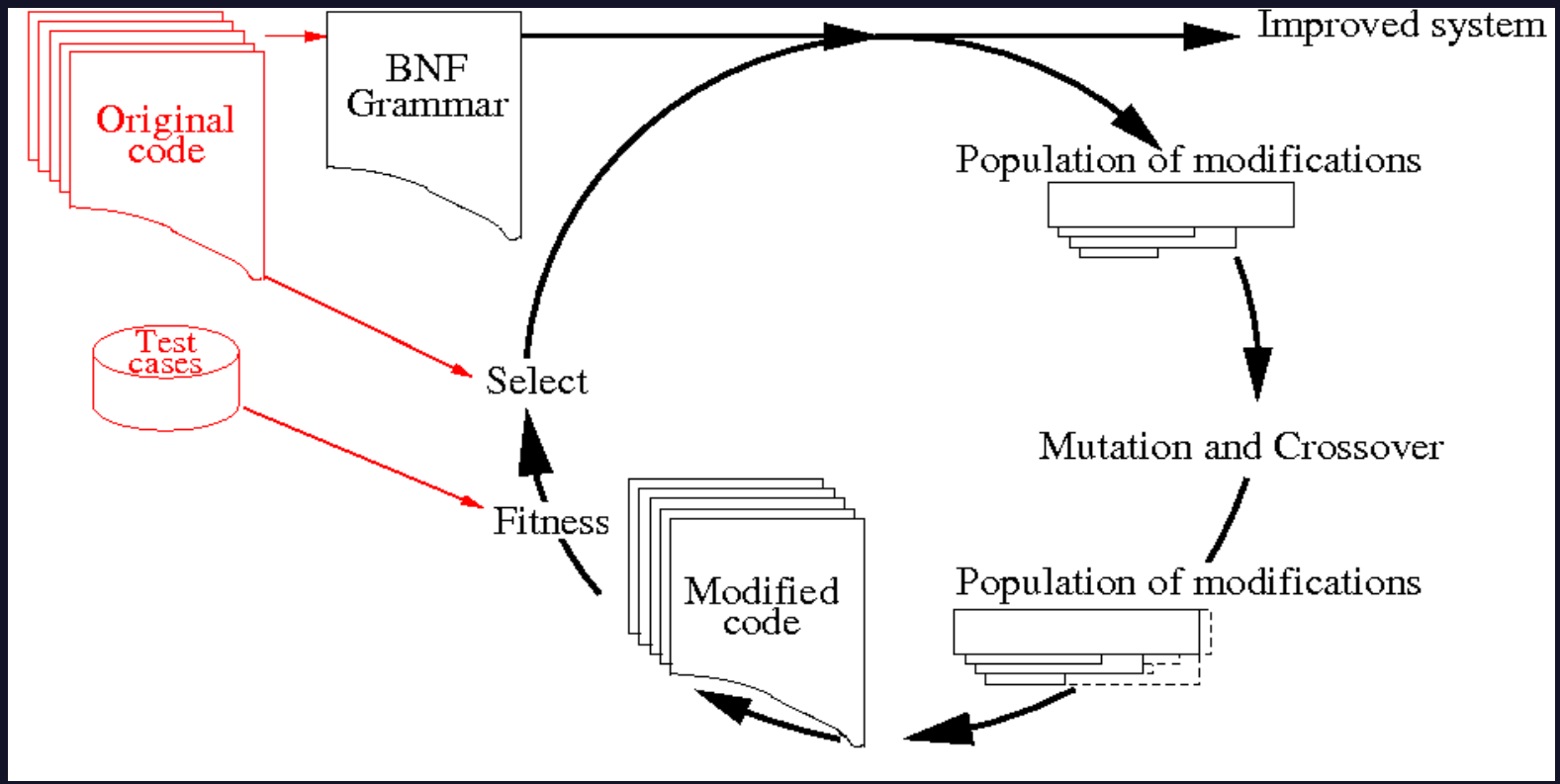
Top-half of population selected

Based on a threshold fitness value

Mutation applied with 50% probability

Crossover applied with 50% probability

Genetic Improvement



Filtering

Mutations in best individuals are often independent

Greedy approach used to combine best individuals

Question

Can we improve the efficiency of an already highly-optimised piece of software using genetic programming?

Motivation for choosing a SAT solver

Bounded Model Checking

Planning

Software Verification

Automatic Test Pattern Generation

Combinational Equivalence Checking

Combinatorial Interaction Testing

and many other applications..

Motivation for choosing MiniSAT

highly-optimised and very popular

MiniSAT-hack track in SAT solver competitions

Question

Can we evolve a version of the MiniSAT solver that is faster

than any of the human-improved versions of the solver?

Experiments: Setup

Solvers used:

MiniSAT2-070721

Experiments: Setup

Solvers used:

MiniSAT2-070721

Test cases used:

$\sim$ 2.5% improvement for general benchmarks (SSBSE'13)

Motivation for choosing a SAT solver

MiniSAT-hack track in SAT solver competitions

- good source for software transplants

Question

Can we evolve a version of the MiniSAT solver that is faster than any of the human-improved versions of the solver for a particular problem class?

Experiments: Setup

Solvers used:

MiniSAT2-070721

Test cases used:

130 from Combinatorial Interaction Testing field

Combinatorial Interaction Testing

Used for testing configurable systems

Combinatorial Interaction Testing

Used for testing configurable systems

A *t*-way interaction test suite covers all interactions between

any *t* parameters.

Combinatorial Interaction Testing

Used for testing configurable systems

A t-way interaction test suite covers all interactions between any t parameters.

Objective: find a minimal t-way interaction test suite

CIT Example

Web Browser:

Load content	Notify pop-up blocked	Cookies	Warn before add-ons install	Remember downloads
Allow Restrict Block	Yes No	Allow Restrict Block	Yes No	Yes No

CIT Example

Web Browser:

Load content	Notify pop-up blocked	Cookies	Warn before add-ons install	Remember downloads
Allow Restrict Block	Yes No	Allow Restrict Block	Yes No	Yes No

3 x 2 x 3 x 2 x 2

= 72 combinations

CIT Example

Pairwise CIT Test Suite for Web Browser:

Load content	Notify pop-up blocked	Cookies	Warn before add-ons install	Remember downloads
Allow	Yes	Allow	Yes	Yes
Allow	No	Restrict	No	No
Allow	Yes	Block	No	No
Restrict	No	Allow	No	Yes
Restrict	Yes	Restrict	No	No
Restrict	No	Block	Yes	Yes
Block	Yes	Allow	Yes	No
Block	No	Restrict	Yes	Yes
Block	Yes	Block	No	Yes

9 tests (cover all combinations between any pair of parameters)

Combinatorial Interaction Testing

CIT problem in SAT:

Is there a test suite of size N that covers all t-way interactions

between any t parameters?

Combinatorial Interaction Testing

Use of SAT-solvers limited due to poor scalability

Question

How long does it take to solve a Combinatorial Interaction

Testing instance using a SAT solver?

Question

It takes hours to days to solve real-world Combinatorial

Interaction Testing problems using a SAT solver.

Experiments: Setup

Host program:

MiniSAT2-070721 (478 lines in main algorithm)

Donor programs:

Experiments: Setup

Host program:

MiniSAT2-070721 (478 lines in main algorithm)

Donor programs:

MiniSAT-best09 (winner of ’09 MiniSAT-hack competition)

MiniSAT-bestCIT (best for CIT from ’09 competition)

- total of 104 new lines in code bank

Results

Solver	Donor	Lines	Seconds
MiniSAT (original)	—	1.00	1.00
MiniSAT-best09	—	1.46	1.76
MiniSAT-bestCIT	—	0.72	0.87
MiniSAT-best09+bestCIT	—	1.26	1.63

Question

How much runtime improvement can we achieve?

Results

Solver	Donor	Lines	Seconds
MiniSAT (original)	—	1.00	1.00
MiniSAT-best09	—	1.46	1.76
MiniSAT-bestCIT	—	0.72	0.87
MiniSAT-best09+bestCIT	—	1.26	1.63
MiniSAT-gp	best09	0.93	0.95

Results

Donor: best09

13 delete, 9 replace, 1 copy

Among changes:

3 assertions removed

1 deletion on variable used for statistics

Results

Mainly IF and FOR statements switched off

Decreased iteration count in FOR loops

Removed optimisation

```
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;
    }
```

Removed optimisation

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

Results

Solver	Donor	Lines	Seconds
MiniSAT (original)	—	1.00	1.00
MiniSAT-best09	—	1.46	1.76
MiniSAT-bestCIT	—	0.72	0.87
MiniSAT-best09+bestCIT	—	1.26	1.63
MiniSAT-gp	best09	0.93	0.95
MiniSAT-gp	bestCIT	0.72	0.87

Results

Donor: bestCIT

1 delete, 1 replace

Among changes:

1 assertion deletion

1 replace operation triggers 95% of donor code

Original Code

```
// Simplify conflict clause:
    //
    int i, j;
    if (expensive_ccmin){
        uint32_t abstract_level = 0;
        for (i = 1; i < out_learnt.size(); i++)
            abstract_level |= abstractLevel(var(out_learnt[i])); // (maintain an abstraction of levels inv


        out_learnt.copyTo(analyze_toclear);
/**/        for (i = j = 1; i < out_learnt.size(); i++)
/**/            if (reason[var(out_learnt[i])] == NULL || !litRedundant(out_learnt[i], abstract_level))
/**/                out_learnt[j++] = out_learnt[i];
```

Code after REPLACE Mutation

```
// Simplify conflict clause:
    //
    int i, j;
    if (expensive_ccmin){
        uint32_t abstract_level = 0;
        for (i = 1; i < out_learnt.size(); i++)
            abstract_level |= abstractLevel(var(out_learnt[i])); // (maintain an abstraction of levels inv


        out_learnt.copyTo(analyze_toclear);
/**/      i = out_learnt.size();
/**/      int found_some = find_removable(out_learnt, i, abstract_level);
/**/      if (found_some)
/**/          j = prune_removable(out_learnt);
/**/      else
/**/          j = i;
```

Results

Solver	Donor	Lines	Seconds
MiniSAT (original)	—	1.00	1.00
MiniSAT-best09	—	1.46	1.76
MiniSAT-bestCIT	—	0.72	0.87
MiniSAT-best09+bestCIT	—	1.26	1.63
MiniSAT-gp	best09	0.93	0.95
MiniSAT-gp	bestCIT	0.72	0.87
MiniSAT-gp	best09+bestCIT	0.94	0.96

Results

Donor: best09+bestCIT

50 delete, 20 replace, 5 copy

Among changes:

5 assertions removed

4 semantically equivalent replacements

3 operations used for statistics removed

$\sim$ half of the mutations remove dead code

Results

Solver	Donor	Lines	Seconds
MiniSAT (original)	—	1.00	1.00
MiniSAT-best09	—	1.46	1.76
MiniSAT-bestCIT	—	0.72	0.87
MiniSAT-best09+bestCIT	—	1.26	1.63
MiniSAT-gp	best09	0.93	0.95
MiniSAT-gp	bestCIT	0.72	0.87
MiniSAT-gp	best09+bestCIT	0.94	0.96
MiniSAT-gp-combined	best09+bestCIT	**0.54**	**0.83**

Results

Combining results:

37 delete, 15 replace, 4 copy

56 out of 100 mutations used

Among changes:

8 assertion removed

95% of the bestCIT donor code executed

Conclusions

Introduced multi-donor software transplantation

Conclusions

Introduced multi-donor software transplantation

Used genetic improvement as means to specialise software

Conclusions

Introduced multi-donor software transplantation

Used genetic improvement as means to specialise software

Achieved 17% runtime improvement on MiniSAT

for the Combinatorial Interaction Testing domain

by combining best individuals