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

Justyna Petke[1], Mark Harman[1], William B. Langdon[1] & Westley Weimer[2]

[1]University College London
[2]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

Contributions

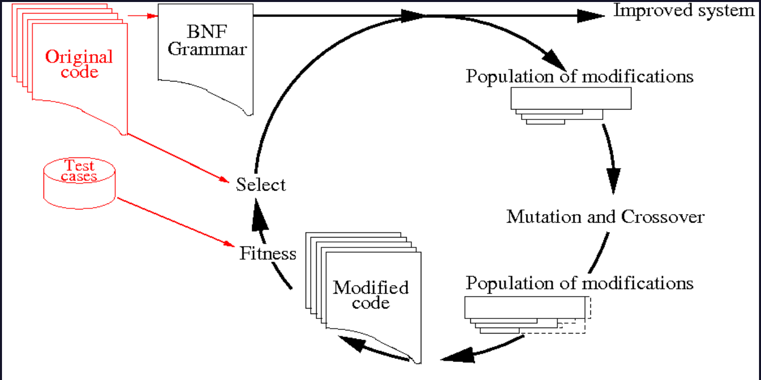
Introduction of multi-donor software transplantation

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

Avoids environmental bias

Fitness

Test cases are sorted into groups

One test case is sampled uniformly from each group

Avoids overfitting

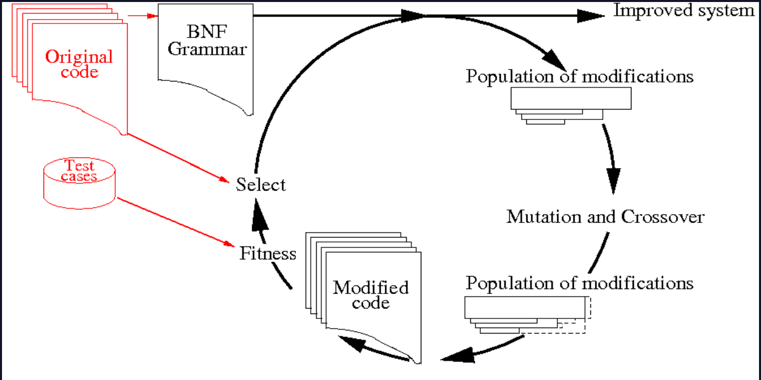
Selection

Fixed number of generations

Fixed population size

Top-half selected for next generation

Genetic Improvement



Filtering

Mutations in best individuals are often independent

Greedy approach used to combine best individuals

Motivation for choosing a SAT solver

Boolean satisfiability (SAT) example:

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

$$\neg x_2 \vee \neg x_3$$

- x_i : a Boolean variable

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 a SAT solver

MiniSAT-hack track in SAT solver competitions

Motivation for choosing a SAT solver

MiniSAT-hack track in SAT solver competitions

- good source for software transplants

Experiments: Setup

Solvers used:

MiniSAT2-070721

Experiments: Setup

Solvers used:

MiniSAT2-070721

Test cases used:

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

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

Use of SAT-solvers limited due to poor scalability

Combinatorial Interaction Testing

Used for testing configurable systems

Use of SAT-solvers limited due to poor scalability

How long does it take to solve a real-world problem?

Combinatorial Interaction Testing

Used for testing configurable systems

Use of SAT-solvers limited due to poor scalability

It takes hours to days to solve a simple real-world problem

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

Question

Can we evolve a version of the MiniSAT solver that is faster than *any* of the human-improved versions of the solver?

Results

Solver	Donor	Lines	Time
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

Results

Solver	Donor	Lines	Time
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

Results

Solver	Donor	Lines	Time
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

Results

Solver	Donor	Lines	Time
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

$\sim$ half of the mutations remove dead code

Results

Solver	Donor	Lines	Time
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