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

(EuroGP’14)
Justyna Petke: Using Genetic Improvement & Code Transplants to Specialise a C++ Program to a Problem Class
Authors

Justyna Petke  Mark Harman  William B. Langdon  Westley Weimer

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

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

Justyna Petke

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

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

Justyna Petke
Using Genetic Improvement & Code Transplants to Specialise a C++ Program to a Problem Class
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\% \text{ improvement for general benchmarks (SSBSE’13)}
Motivation for choosing a SAT solver

MiniSAT-hack track in SAT solver competitions
- good source for software transplants
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

Justyna Petke

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

<table>
<thead>
<tr>
<th>Load content</th>
<th>Notify pop-up</th>
<th>Cookies</th>
<th>Warn before add-ons install</th>
<th>Remember downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>Yes</td>
<td>Allow</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Restrict</td>
<td>No</td>
<td>Restrict</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Block</td>
<td>Yes</td>
<td>Block</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

3 x 2 x 3 x 2 x 2 = 72 combinations
### CIT Example

#### Web Browser:

<table>
<thead>
<tr>
<th>Allow</th>
<th>Notify pop-up</th>
<th>Cookies</th>
<th>Warn before add-ons install</th>
<th>Remember downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>Allow</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td></td>
<td>Restrict</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td></td>
<td>Block</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Load content</th>
<th>Notify pop-up blocked</th>
<th>Cookies</th>
<th>Warn before add-ons install</th>
<th>Remember downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>Yes</td>
<td>Allow</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Restrict</td>
<td>No</td>
<td>Restrict</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Block</td>
<td></td>
<td>Block</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$$3 \times 2 \times 3 \times 2 \times 2 = 72$$ combinations
### CIT Example

**Pairwise CIT Test Suite for Web Browser:**

<table>
<thead>
<tr>
<th>Load content</th>
<th>Notify pop-up blocked</th>
<th>Cookies</th>
<th>Warn before add-ons install</th>
<th>Remember downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>Yes</td>
<td>Allow</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Allow</td>
<td>No</td>
<td>Restrict</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Allow</td>
<td>Yes</td>
<td>Block</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Restrict</td>
<td>No</td>
<td>Allow</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Restrict</td>
<td>Yes</td>
<td>Restrict</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Restrict</td>
<td>No</td>
<td>Block</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Block</td>
<td>Yes</td>
<td>Allow</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Block</td>
<td>No</td>
<td>Restrict</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Block</td>
<td>Yes</td>
<td>Block</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Solver</th>
<th>Donor</th>
<th>Lines</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>MiniSAT (original)</td>
<td>—</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>MiniSAT-best09</td>
<td>—</td>
<td>1.46</td>
<td>1.76</td>
</tr>
<tr>
<td>MiniSAT-bestCIT</td>
<td>—</td>
<td>0.72</td>
<td>0.87</td>
</tr>
<tr>
<td>MiniSAT-best09+bestCIT</td>
<td>—</td>
<td>1.26</td>
<td>1.63</td>
</tr>
</tbody>
</table>
How much runtime improvement can we achieve?
### Results

<table>
<thead>
<tr>
<th>Solver</th>
<th>Donor</th>
<th>Lines</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>MiniSAT (original)</td>
<td>—</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>MiniSAT-best09</td>
<td>—</td>
<td>1.46</td>
<td>1.76</td>
</tr>
<tr>
<td>MiniSAT-bestCIT</td>
<td>—</td>
<td>0.72</td>
<td>0.87</td>
</tr>
<tr>
<td>MiniSAT-best09+bestCIT</td>
<td>—</td>
<td>1.26</td>
<td>1.63</td>
</tr>
<tr>
<td>MiniSAT-gp</td>
<td>best09</td>
<td>0.93</td>
<td>0.95</td>
</tr>
</tbody>
</table>
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
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

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

<table>
<thead>
<tr>
<th>Solver</th>
<th>Donor</th>
<th>Lines</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>MiniSAT (original)</td>
<td>—</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>MiniSAT-best09</td>
<td>—</td>
<td>1.46</td>
<td>1.76</td>
</tr>
<tr>
<td>MiniSAT-bestCIT</td>
<td>—</td>
<td>0.72</td>
<td>0.87</td>
</tr>
<tr>
<td>MiniSAT-best09+bestCIT</td>
<td>—</td>
<td>1.26</td>
<td>1.63</td>
</tr>
<tr>
<td>MiniSAT-gp</td>
<td>best09</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>MiniSAT-gp</td>
<td>bestCIT</td>
<td>0.72</td>
<td>0.87</td>
</tr>
</tbody>
</table>
Results

Donor: bestCIT

1 delete, 1 replace

Among changes:

1 assertion deletion

1 replace operation triggers 95% of donor 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 involved in conflict)
    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];
// 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 involved)

  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

<table>
<thead>
<tr>
<th>Solver</th>
<th>Donor</th>
<th>Lines</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>MiniSAT (original)</td>
<td>—</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>MiniSAT-best09</td>
<td>—</td>
<td>1.46</td>
<td>1.76</td>
</tr>
<tr>
<td>MiniSAT-bestCIT</td>
<td>—</td>
<td>0.72</td>
<td>0.87</td>
</tr>
<tr>
<td>MiniSAT-best09+bestCIT</td>
<td>—</td>
<td>1.26</td>
<td>1.63</td>
</tr>
<tr>
<td>MiniSAT-gp</td>
<td>best09</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>MiniSAT-gp</td>
<td>bestCIT</td>
<td>0.72</td>
<td>0.87</td>
</tr>
<tr>
<td>MiniSAT-gp</td>
<td>best09+bestCIT</td>
<td>0.94</td>
<td>0.96</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Solver</th>
<th>Donor</th>
<th>Lines</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>MiniSAT (original)</td>
<td>—</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>MiniSAT-best09</td>
<td>—</td>
<td>1.46</td>
<td>1.76</td>
</tr>
<tr>
<td>MiniSAT-bestCIT</td>
<td>—</td>
<td>0.72</td>
<td>0.87</td>
</tr>
<tr>
<td>MiniSAT-best09+bestCIT</td>
<td>—</td>
<td>1.26</td>
<td>1.63</td>
</tr>
<tr>
<td>MiniSAT-gp</td>
<td>best09</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>MiniSAT-gp</td>
<td>bestCIT</td>
<td>0.72</td>
<td>0.87</td>
</tr>
<tr>
<td>MiniSAT-gp</td>
<td>best09+bestCIT</td>
<td>0.94</td>
<td>0.96</td>
</tr>
<tr>
<td>MiniSAT-gp-combined</td>
<td>best09+bestCIT</td>
<td>0.54</td>
<td>0.83</td>
</tr>
</tbody>
</table>
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.