Genetic Improvement

Justyna Petke

Centre for Research in Evolution, Search and Testing
University College London
Thank you

Mark Harman  Yue Jia  Alexandru Marginean
What does the word “Computer” mean?

**Oxford Dictionary**

“a person who makes calculations, especially with a calculating machine.”

**Wikipedia**

“The term "computer", in use from the mid 17th century, meant "one who computes": a person performing mathematical calculations.”
What does the word “Computer” mean?

Oxford Dictionary

“a person who makes calculations, especially with a calculating machine.”

Wikipedia

“The term "computer", in use from the mid 17th century, meant "one who computes": a person performing mathematical calculations.”
in the beginning ...
The First Computer?
Different People have different opinions
Trial model of a part of the Analytical Engine, built by Babbage.

1873
"Pickering's Harem," so-called, for the group of women computers at the Harvard College Observatory

1893
Who are the programmers
Who are the programmers?
Who are the programmers
Who are the programers

it’s always been people
Who are the programmers

it’s always been people
Who are the programmers

lots of people
why people?

human computers seem quaint today

will human programmers seem quaint tomorrow?
Programming is changing
Functional Requirements

- functionality of the Program

Non-Functional Requirements

- Execution Time
- Memory
- Bandwidth
- Battery
- Size
Software Design Process
Software Design Process
Software Design Process
Software Design Process
Software Design Process
Multiplicity

- Multiple Platforms
- Conflicting Objectives
- Multiple Devices
Which requirements must be human coded?

Functional Requirements

Non-Functional Requirements

humans have to define these
Which requirements are essential to human?

**Functional Requirements**
humans have to define these

**Non-Functional Requirements**
we can optimise these
The GISMÖE challenge: Constructing the Pareto Program Surface Using Genetic Programming to Find Better Programs

Mark Harman¹, William B. Langdon¹, Yue Jia², David R. White³, Andrea Arcuri³, John A. Clark⁴
¹CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
²School of Computing Science, University of Glasgow, Glasgow, G12 8QQ, Scotland, UK.
³Simula Research Laboratory, P. O. Box 134, 1325 Lysaker, Norway.
⁴Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, UK.

ABSTRACT
Optimising programs for non-functional properties such as speed, size, throughput, power consumption and bandwidth can be demanding; pity the poor programmer who is asked to cater for them all at once! We set out an alternate vision for a new kind of software development environment inspired by recent results from Search Based Software Engineering (SBSE). Given an input program that satisfies the functional requirements, the proposed programming environment will automatically generate a set of candidate program implementations, all of which share functionality, but each of which differ in their non-functional tradeoffs. The software designer navigates this diverse Pareto surface of candidate implementations, gaining insight into the tradeoffs and selecting solutions for different platforms and environments, thereby stretching beyond the reach of current compiler technology. We present preliminary results on the

Keywords
SBSE, Search Based Optimization, Compilation, Non-functional Properties, Genetic Programming, Pareto Surface.

1. INTRODUCTION
Humans find it hard to develop systems that balance many competing and conflicting non-functional objectives. Even meeting a single objective, such as execution time, requires automated support in the form of compiler optimisation. However, though most compilers can optimise compiled code for both speed and size, the programmer may find themselves making arbitrary choices when such objective are in conflict with one another.

Furthermore, speed and size are but two of many objectives that the next generation of software systems will have to address, with demand for more, such as bandwidth.

Genetic Improvement

Justyna Petke
Pareto Front
Pareto Front

each circle is a program found by a machine
different non functional properties have different pareto program fronts
Failed Test Cases
Why can’t functional properties be optimisation objectives?
Optimisation
Optimisation

2.5 times faster but failed 1 test case?
Optimisation

double the battery life
but failed 2 test cases?
Conformant GI

functional correctness is king

conforms to traditional views

vs.

Heretical GI
Conformant GI vs. Heretical GI

- *functional correctness is king*
  - conforms to traditional views

- *correctness means having sufficient resources for computation*
  - conforms to a compelling new orthodoxy
Conformant GI vs. Heretical GI

- functional correctness is king
- conforms to traditional views

- correctness means having sufficient resources for computation
- conforms to a compelling new orthodoxy

there’s nothing correct about a flat battery
can it work?
Software Uniqueness

500,000,000 LoC

one has to write approximately 6 statements before one is writing unique code
Software Uniqueness


"The space of candidate programs is far smaller than we might suppose."

one has to write approximately 6 statements before one is writing unique code.
Software Robustness

after one line changes up to 89% of programs that compile run without error
Software Robustness

W. B. Langdon and J. Petke
Software is Not Fragile. (CS-DC 2015)

“Software engineering artefacts are more robust than is often assumed.”
Genetic Improvement for Software Specialisation
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
Question

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

Introduction of multi-donor software transplantation

Use of genetic improvement as means to specialise software
Genetic Improvement

Original code → BNF Grammar → Select → Fitness → Modified code → Population of modifications → Mutation and Crossover → Improved system
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)
Mutation

Addition of one of the following operations:

dele

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
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 and Crossover applied
Genetic Improvement

Original code

BNF Grammar

Select

Fitness

Modified code

Population of modifications

Mutation and Crossover

Population of modifications

Improved system
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

Boolean satisfiability (SAT) example:

\[ x_1 \lor x_2 \lor \neg x_4 \]
\[ \neg x_2 \lor \neg x_3 \]

- \( x_i \): a Boolean variable
- \( x_i, \neg x_i \): a literal
- \( \neg x_2 \lor \neg x_3 \): a clause
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
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

Test cases used:

∼ 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:

from Combinatorial Interaction Testing field
Combinatorial Interaction Testing

Use of SAT-solvers limited due to poor scalability

SAT benchmarks containing millions of clauses

It takes hours to days to generate a CIT test suite using SAT
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
## 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>
Question

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>

Genetic Improvement

Justyna Petke
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

<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 best09</td>
<td>best09</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>MiniSAT-gp bestCIT</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
# 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 best09</td>
<td></td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>MiniSAT-gp bestCIT</td>
<td></td>
<td>0.72</td>
<td>0.87</td>
</tr>
<tr>
<td>MiniSAT-gp best09+bestCIT</td>
<td></td>
<td>0.94</td>
<td>0.96</td>
</tr>
</tbody>
</table>

Genetic Improvement
Justyna Petke
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

~ 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>

**Genetic Improvement**  
Justyna Petke
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

Used genetic improvement as means to specialise software

Achieved 17% runtime improvement on MiniSAT

for the Combinatorial Interaction Testing domain

by combining best individuals
Inter version transplantation

Justyna Petke, Mark Harman, William B. Langdon and Westley Weimer
Using Genetic Improvement & Code Transplants to Specialise a C++ program
to a Problem Class (EuroGP’14)
Genetic Improvement of Programs

W. B. Langdon and M. Harman
Optimising Existing Software with Genetic Programming. TEC 2015

- Sensitivity Analysis
- Non-functional property Test
- 70 times faster
- HC clean up: 7
- slight semantic improvement
- 30+ interventions
Genetic Improvement of Programs

W. B. Langdon and M. Harman
Genetically Improved CUDA C++ Software, EuroGP 2014

7 times faster updated for new hardware automated updating
Memory speed trade offs

Fan Wu, Westley Weimer, Mark Harman, Yue Jia and Jens Krinke
Deep Parameter Optimisation
Conference on Genetic and Evolutionary Computation (GECCO 2015)

Improve execution time by 12% or achieve a 21% memory consumption reduction
Reducing energy consumption

Energy consumption can be reduced by as much as 25%

Bobby R. Bruce Justyna Petke Mark Harman
Reducing Energy Consumption Using Genetic Improvement
Conference on Genetic and Evolutionary Computation (GECCO 2015)
Mark Harman, Yue Jia and Bill Langdon, Babel Pidgin: SBSE can grow and graft entirely new functionality into a real world system. Symposium on Search-Based Software Engineering SSBSE 2014. (Challenge track)
Real world cross system transplantation

Successfully autotransplanted new functionality and passed all regression tests for 12 out of 15 real world systems

Earl T. Barr, Mark Harman, Yue Jia, Alexandru Marginean, and Justyna Petke
Automated Software Transplantation (ISSTA 2015)
Automated Software Transplantation

E.T. Barr, M. Harman, Y. Jia, A. Marginean & J. Petke

ACM Distinguished Paper Award at ISSTA 2015

Coverage in

Code 'transplant' could revolutionise programming

2647 shares of article in Wired.co.uk

Coverage in

Click
Why Antarctic transplantation?

- ~100 players
- Why not handle H.264?
- Start from scratch
- Check open source repositories
Human Organ Transplantation
Automated Software Transplantation

Manual Work:
- Organ Entry
- Organ’s Test Suite
- Implantation Point

Donor
- ENTRY
- Organ

Host

Organ Test Suite

Genetic Improvement
μTrans

Stage 1: Static Analysis
Stage 2: Genetic Programming
Stage 3: Organ Implantation

Host
Implantation Point

Donor
Organ Entry

Organ Test Suite
Host Beneficiary

CREST
Genetic Improvement
Justyna Petke
Stage 1 — Static Analysis

Donor: int X -> Host: int A, B, C
Stage 2 — GP

Matching Table

Donor Variable ID    Host Variable ID (set)

$V_1^D$        $V_1^H, V_2^H$

$V_2^D$        $V_3^H, V_4^H, V_5^H$

... 

Individual

<table>
<thead>
<tr>
<th>Var Matching</th>
<th>Individual</th>
</tr>
</thead>
<tbody>
<tr>
<td>M$_1$:</td>
<td>$V_1^D$</td>
</tr>
<tr>
<td></td>
<td>$V_1^H$</td>
</tr>
<tr>
<td>M$_2$:</td>
<td>$V_2^D$</td>
</tr>
<tr>
<td></td>
<td>$V_4^H$</td>
</tr>
</tbody>
</table>

Does it produce the correct output?

Strong Proxies: Does it produce the correct output?

fitness($i$) = \[
\begin{cases} 
\frac{1}{3} (1 + \frac{|TX_i|}{|T|} + \frac{|TP_i|}{|T|}) & i \in I_C \\
0 & i \notin I_C
\end{cases}
\]
Research Questions

- Acceptance Tests
- Host
- Donor

Is autotransplantation useful?
Research Questions

- Do we break the initial functionality?
- How about the computational effort?
- Have we really added new functionality?
- Is autotransplantation useful?

Empirical Study

- 15 Transplantations
- 300 Runs
- 5 Donors
- 3 Hosts

Case Study:

- H.264 Encoding Transplantation

How about the computational effort?

Is autotransplantation useful?
Validation

Regression Tests

Augmented Regression Tests

Donor Acceptance Tests

Acceptance Tests

Host Beneficiary

Manual Validation

CREST
Genetic Improvement

Justyna Petke
## Subjects

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Type</th>
<th>Size KLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idct</td>
<td>Donor</td>
<td>2.3</td>
</tr>
<tr>
<td>Mytar</td>
<td>Donor</td>
<td>0.4</td>
</tr>
<tr>
<td>Cflow</td>
<td>Donor</td>
<td>25</td>
</tr>
<tr>
<td>Webserver</td>
<td>Donor</td>
<td>1.7</td>
</tr>
<tr>
<td>TuxCrypt</td>
<td>Donor</td>
<td>2.7</td>
</tr>
<tr>
<td>Pidgin</td>
<td>Host</td>
<td>363</td>
</tr>
<tr>
<td>Cflow</td>
<td>Host</td>
<td>25</td>
</tr>
<tr>
<td>SoX</td>
<td>Host</td>
<td>43</td>
</tr>
</tbody>
</table>

- **Case Study**
  - x264: Donor 63
  - VLC: Host 422

- **Minimal size**: 0.4k
- **Max size**: 422k
- **Average Donor**: 16k
- **Average Host**: 213k
Experimental Methodology and Setup

Host

Implantation Point

Organ Test Suite

Donor

OE

64 bit Ubuntu 14.10
16 GB RAM
8 threads

Justyna Petke

Genetic Improvement
## Empirical Study

<table>
<thead>
<tr>
<th>Donor</th>
<th>Host</th>
<th>All Passed</th>
<th>Regression</th>
<th>Regression++</th>
<th>Acceptance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idct</td>
<td>Pidgin</td>
<td>16</td>
<td>20</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>Mytar</td>
<td>Pidgin</td>
<td>16</td>
<td>20</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>Web</td>
<td>Pidgin</td>
<td>16</td>
<td>20</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Cflow</td>
<td>Pidgin</td>
<td>16</td>
<td>20</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>Tux</td>
<td>Pidgin</td>
<td>16</td>
<td>20</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>Idct</td>
<td>Cflow</td>
<td>16</td>
<td>20</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>Mytar</td>
<td>Cflow</td>
<td>16</td>
<td>20</td>
<td>17</td>
<td>20</td>
</tr>
<tr>
<td>Web</td>
<td>Cflow</td>
<td>16</td>
<td>20</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>Cflow</td>
<td>Cflow</td>
<td>16</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Tux</td>
<td>Cflow</td>
<td>16</td>
<td>20</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>Idct</td>
<td>SoX</td>
<td>16</td>
<td>20</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>Mytar</td>
<td>SoX</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>20</td>
</tr>
<tr>
<td>Web</td>
<td>SoX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cflow</td>
<td>SoX</td>
<td>14</td>
<td>16</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>Tux</td>
<td>SoX</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td>188/300</td>
<td>233/300</td>
<td>196/300</td>
<td>256/300</td>
</tr>
</tbody>
</table>

In 12 out of 15 experiments we successfully autotransplanted new functionality.
# Empirical Study

<table>
<thead>
<tr>
<th>Donor</th>
<th>Host</th>
<th>Average</th>
<th>Std. Dev.</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idct</td>
<td>Pidgin</td>
<td>5</td>
<td>7</td>
<td>97</td>
</tr>
<tr>
<td>Mytar</td>
<td>Pidgin</td>
<td>3</td>
<td>1</td>
<td>65</td>
</tr>
<tr>
<td>Web</td>
<td>Pidgin</td>
<td>8</td>
<td>5</td>
<td>160</td>
</tr>
<tr>
<td>Cflow</td>
<td>Pidgin</td>
<td>58</td>
<td>16</td>
<td>1151</td>
</tr>
<tr>
<td>Tux</td>
<td>Pidgin</td>
<td>29</td>
<td>10</td>
<td>574</td>
</tr>
<tr>
<td>Idct</td>
<td>Cflow</td>
<td>3</td>
<td>5</td>
<td>59</td>
</tr>
<tr>
<td>Mytar</td>
<td>Cflow</td>
<td>3</td>
<td>1</td>
<td>53</td>
</tr>
<tr>
<td>Web</td>
<td>Cflow</td>
<td>5</td>
<td>2</td>
<td>102</td>
</tr>
<tr>
<td>Cflow</td>
<td>Cflow</td>
<td>44</td>
<td>9</td>
<td>872</td>
</tr>
<tr>
<td>Tux</td>
<td>Cflow</td>
<td>31</td>
<td>11</td>
<td>623</td>
</tr>
<tr>
<td>Idct</td>
<td>SoX</td>
<td>12</td>
<td>17</td>
<td>233</td>
</tr>
<tr>
<td>Mytar</td>
<td>SoX</td>
<td>3</td>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td>Web</td>
<td>SoX</td>
<td>7</td>
<td>3</td>
<td>132</td>
</tr>
<tr>
<td>Cflow</td>
<td>SoX</td>
<td>89</td>
<td>53</td>
<td>74</td>
</tr>
<tr>
<td>Tux</td>
<td>SoX</td>
<td>34</td>
<td>13</td>
<td>94</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>Total</strong></td>
<td><strong>334 (min)</strong></td>
<td><strong>10 (Average)</strong></td>
<td><strong>72 (hours)</strong></td>
</tr>
</tbody>
</table>
Case Study

within 26 hours performed a task that took developers an avg of 20 days of elapsed time
Automated Software Transplantation

Manual Work:
- Organ Entry
- Organ's Test Suite
- Implantation Point

Validation
- Regression Tests
  - RQ1.a
  - Augmented Regression Tests
    - RQ1.b
  - Manual Validation
- Donor Acceptance Tests
  - Host
  - Beneficiary
- Acceptance Tests
  - RQ2

Subjects
- | Subjects | Type | Size KLOC |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Idct</td>
<td>Donor</td>
<td>2.3</td>
</tr>
<tr>
<td>Mytar</td>
<td>Donor</td>
<td>0.4</td>
</tr>
<tr>
<td>Cflow</td>
<td>Donor</td>
<td>25</td>
</tr>
<tr>
<td>Webserver</td>
<td>Donor</td>
<td>1.7</td>
</tr>
<tr>
<td>TuxCrypt</td>
<td>Donor</td>
<td>2.7</td>
</tr>
<tr>
<td>Pidgin</td>
<td>Host</td>
<td>363</td>
</tr>
<tr>
<td>Cflow</td>
<td>Host</td>
<td>25</td>
</tr>
<tr>
<td>SoX</td>
<td>Host</td>
<td>43</td>
</tr>
<tr>
<td>x264</td>
<td>Donor</td>
<td>63</td>
</tr>
<tr>
<td>VLC</td>
<td>Host</td>
<td>422</td>
</tr>
</tbody>
</table>

μTrans
- Stage 1: Static Analysis
- Stage 2: Genetic Programming
- Stage 3: Organ Implantation
- Organ's Test Suite
- Host
- Beneficiary

Genetic Improvement
muScalpel

Implemented in TXL and C, muScalpel realizes μTrans and comprises 28k SLoCs, of which 16k is TXL, and 12k is C. muScalpel implements a custom version of GP. Unlike conventional GP, which creates an initial population from individuals that contain multiple statements, muScalpel generates an initial population of individuals with just 1 statement, uniformly selected. muScalpel’s underlying assumption is that our organs need very few of the statements in their donor. Starting from one LOC gives muScalpel the possibility to find small solutions quickly. muScalpel focuses on evolving the organ’s vein. muScalpel also inherits the limitations of TXL, such as its stack limit which precludes parsing large programs and its default C grammar’s inability to properly handle preprocessor directives.

As we all know, software is often difficult to build and run, due to dependencies on its development environment and target platform. muScalpel is no exception. Please keep in mind that we built and ran muScalpel only on 64-bit Ubuntu 14.04 LTS machine, with 16 GB RAM, SSD and 8 physical cores, with its TXL v10.6a-64 (14.7.13), gcc-4.8, cflow (GNU cflow) 1.4 installed. Any other configurations may have affect on the results of the replication of our experiments.

This website contains the source for muScalpel, muScalpel in binary form, and the data sets, including test suites, that underlie our experiments. To facilitate replicating our results, we have written a sequence of scripts that run a “single” run of each of our experiments. The name of the script identifies the experiment. We have worked hard to make each script bullet-proof and have it thoroughly check your environment for its dependencies and tell you what, if anything, is missing. Despite our best efforts, you may still encounter problems. If that happens, please contact us so we can work with you to resolve them.

Experiment Scripts

- Link to a script that runs all our experiments, as submitted to ISSTA 2015 artifact evaluation track. Here we also provide a dockerized version of our experiments.

* [http://crest.cs.ucl.ac.uk/autotransplantation/MuScalpel.html](http://crest.cs.ucl.ac.uk/autotransplantation/MuScalpel.html)
GI Applications

Bug fixing
and other … 
Claire Le Goues, Stephanie Forrest, Westley Weimer: Current challenges in automatic software repair.

* http://dijkstra.cs.virginia.edu/genprog/
GI Applications

- Bug fixing
- Improving energy consumption
- Porting old code to new hardware
- Grafting new functionality into an existing system
- Specialising software for a particular problem class
- Other
What if fitness is expensive to compute?

GI4GI: Improving Genetic Improvement Fitness Functions
Mark Harman & Justyna Petke
(Genetic Improvement Workshop 2015)
GI4GI: Energy Optimisation Example

many factors affecting energy consumption, including:

screen behaviour

memory access

device communications

CPU utilisation
GI4GI: Energy Optimisation Example

a hardware-dependent linear energy model for GI:

\[
\text{power} = C_{\text{const}} + C_{\text{ins}} \frac{\text{ins}}{\text{cycle}} + C_{\text{flops}} \frac{\text{flops}}{\text{cycle}} + C_{\text{tca}} \frac{\text{tca}}{\text{cycle}} + C_{\text{mem}} \frac{\text{mem}}{\text{cycle}}
\]

\[
\text{energy} = \text{seconds} \times \text{power}
\]

Post-compiler software optimization for reducing energy (ASPLOS’14) Schulte et al.
GI4GI: Energy Optimisation Example

Idea:

Use GI to evolve a fitness function $f$ for energy consumption.

Use $f$ to improve energy consumption of software.
GI4GI
GI Growth
1st International Genetic Improvement Workshop
at GECCO 2015, Madrid, Spain
www.geneticimprovementofsoftware.com
GI Growth

Special Session on GI *http://www.wcci2016.org/

Special Issue on GI
Conclusions

Functional Requirements

humans have to define these

Non-Functional Requirements

we can optimise these
Summary

Search Based Optimisation

Software Engineering

Genetic Improvement

Combinatorial Interaction Testing

Genetic Improvement
Research Opportunities
Contact me if you want to visit CREST:

j.petke at ucl.ac.uk

Centre for Research on Evolution, Search and Testing
University College London
COWs
CREST Open Workshop
Roughly one per month
Discussion based
Recorded and archived
http://crest.cs.ucl.ac.uk/cow/
COWs

http://crest.cs.ucl.ac.uk/cow/
COWs

#Total Registrations 1512
#Unique Attendees 667
#Unique Institutions 244
#Countries 43
#Talks 421

(Last updated on November 4, 2015)

http://crest.cs.ucl.ac.uk/cow/
CREST Open Workshop (COW)

The 41st CREST Open Workshop
Software Engineering And Computer Science Using Information
Date: 27 -28 April 2015
Venue: Engineering Front Executive Suite, Roberts Building, UCL

The 40th CREST Open Workshop
SSBSE 2015 Challenge: Collaborative Jam Session
Date: 30 - 31 March 2015
Venue: Engineering Front Executive Suite, Roberts Building, UCL

The 39th CREST Open Workshop
Measuring, Testing and Optimising Computational Energy Consumption
Date: 23 - 24 February 2015
Venue: Engineering Front Executive Suite, Roberts Building, UCL

The 38th CREST Open Workshop
Working Tutorial on Statistical Methods in Experimental Software Engineering
Date: 26 - 27 January 2015
Venue: Engineering Front Executive Suite, Roberts Building, UCL

The 37th CREST Open Workshop
Working Tutorial on Empirical Software Engineering Methods
Date: 24 - 25 November 2014
Venue: Engineering Front Executive Suite, Roberts Building, UCL

The 36th CREST Open Workshop
App Store Analysis
Date: 27 - 28 October 2014
Dynamic Adaptive Search Based Software Engineering

EPSRC Grant

Stirling
York
Birmingham
UCL

DAASE

Justyna Petke
Summary

Search Based Optimisation
Software Engineering

Genetic Improvement

Combinatorial Interaction Testing

COWs
Visitor Scheme
Open positions
Pictures used with thanks from these sources

Stonehenge: By Yuanyuan Zhang [All right reserved] via Flickr

Pickering’s Harem: [Public domain], via Wikimedia Commons

IBM 026 Card Punch: By Ben Franske (Own work) [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC-BY-SA-3.0-2.5-2.0-1.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

BBC_Micro: [Public domain], via Wikimedia Commons

IBM PC: By Boffy B (Own work) [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC-BY-SA-3.0-2.5-2.0-1.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

IMac: By Matthieu Riegler, Wikimedia Commons [CC-BY-3.0 (http://creativecommons.org/licenses/by/3.0)], via Wikimedia Commons

Ada Lovelace: By Alfred Edward Chalon [Public domain], via Wikimedia Commons

Programmer: undesarchiv, B 145 Bild-F031434-0006 / Gathmann, Jens / CC-BY-SA [CC-BY-SA-3.0-de (http://creativecommons.org/licenses/by-sa/3.0/de/deed.en)], via Wikimedia Commons

Bath Abbey: By Yuanyuan Zhang [All right reserved] via Flickr