Revolutionising the process of software development

Justyna Petke

Centre for Research in Evolution, Search and Testing
University College London
Genetic Improvement of Software

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.”
Who are the programmers
Who are the programmers
Who are the programmers
Who are the programmers

it’s always been people
Who are the programmers

it’s always been people
Who are the programers

lots of people
Why people?

human computers seem quaint today
will human programmers seem quaint tomorrow?
programming is changing
Functional Requirements

Non-Functional Requirements

Requirements
Functional Requirements

Non-Functional Requirements

- Execution Time
- Memory
- Bandwidth
- Energy
- Size

functionality of the Program
Software Design Process
Software Design Process
Software Design Process
Software Design Process
Software Design Process
Multiplicity

- Multiple Platforms
- Conflicting Objectives
- Multiple Devices

CREST
Genetic Improvement

Justyna Petke
Which requirements must be human coded?

**Functional Requirements**

humans have to define these

**Non-Functional Requirements**
Which requirements are essential to human?

**Functional Requirements**
- Humans have to define these.

**Non-Functional Requirements**
- We can optimise these.
Can it work?
Software Uniqueness

500,000,000 LoC

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


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

"The space of candidate programs is far smaller than we might suppose."
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.”
Non-Functional Requirements

How can we optimise these?
Genetic Improvement

The GISMOE challenge: Constructing the Pareto Program Surface Using Genetic Programming to Find Better Programs.

Mark Harman¹, William B. Langdon¹, Yue Jia¹, David R. White², 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 trade offs. The software designer navigates this diverse Pareto surface of candidate implementations, gaining insight into the trade offs and selecting solutions for different platforms and environments, thereby stretching beyond the reach of current compiler technology. Further, this environment will focus 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 balance. There are many others such as bandwidth and time.

http://www.cs.ucl.ac.uk/staff/ucacbbl/gismo/
Which requirements are essential to human?

Functional Requirements
- humans have to define these

Non-Functional Requirements
- we can optimise these
Genetic Improvement Framework

1. Software
2. Sensitivity Analysis
3. Search
   - Test data
   - Fitness
4. Non-functional property Test
5. Improved Software
Efficiency Improvement

W. B. Langdon and M. Harman
*Optimising Existing Software with Genetic Programming.*
Transactions on Evolutionary Computation (TEC) 2015
<table>
<thead>
<tr>
<th>Mutation</th>
<th>Source file</th>
<th>Line (type)</th>
<th>Orig. code</th>
<th>New code</th>
</tr>
</thead>
<tbody>
<tr>
<td>replaced</td>
<td>bt2_io.cpp</td>
<td>622 (for2)</td>
<td>i &lt; offsLenSampled</td>
<td>i &lt; this-&gt;_nPat</td>
</tr>
<tr>
<td>replaced</td>
<td>sa_rescomb.cpp</td>
<td>50 (for2)</td>
<td>i &lt; satup_-&gt;offs.size()</td>
<td>0</td>
</tr>
<tr>
<td>disabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replaced</td>
<td>aligner_swsse_ee_u8.cpp</td>
<td>707</td>
<td>vh = _mm_max_epu8(vh, vf);</td>
<td>vmax = vlo;</td>
</tr>
<tr>
<td>deleted</td>
<td>aligner_swsse_ee_u8.cpp</td>
<td>766</td>
<td>pvFStore += 4;</td>
<td></td>
</tr>
<tr>
<td>replaced</td>
<td>aligner_swsse_ee_u8.cpp</td>
<td>772</td>
<td>_mm_store_si128(pvHStore, vh); vh = _mm_max_epu8(vh, vf);</td>
<td>ve = _mm_max_epu8(ve, vh);</td>
</tr>
<tr>
<td>deleted</td>
<td>aligner_swsse_ee_u8.cpp</td>
<td>778</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Runtime reduction from 12 days to 4 hours**
Efficiency Improvement

CUDA → Sensitivity Analysis → GP → CUDA Improv

- Test data
- Fitness
- Non-functional property Test

W.B. Langdon, B.Y.H. Lam, J. Petke & M. Harman
*Improving CUDA DNA Analysis Software with Genetic Programming*
Genetic and Evolutionary Computation Conference (GECCO) 2015
Efficiency Improvement

Challenge: Use genetic improvement to improve program efficiency of a state-of-the-art bioinformatics program for DNA sequence mapping called BarraCUDA, consisting of 8,000+ lines of code.

Results: The improved version of BarraCUDA is up to 3x faster than the original on large real-world datasets. The new version has been adopted into development and has been downloaded over 1,000 times so far. Ported by IBM to one of their supercomputers and adopted by Lab7.

W.B. Langdon, B.Y.H. Lam, J. Petke & M. Harman
Improving CUDA DNA Analysis Software with Genetic Programming
Genetic and Evolutionary Computation Conference (GECCO) 2015
Genetic Improvement Framework

- Software
- Sensitivity Analysis
- Search
  - Test data
  - Fitness
- Non-functional property Test
- Improved Software
Software Specialisation with Transplants

- Sensitivity Analysis
- Search
  - Test data
  - Fitness
- Non-functional property Test
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?
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..

Genetic Improvement

Justyna Petke
Motivation for choosing a SAT solver

MiniSAT-hack track in SAT solver competitions
Contributions

Introduction of multi-donor software transplantation
Contributions

Introduction of multi-donor software transplantation

Use of genetic improvement as means to specialise software
Software Specialisation with Transplants

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

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

European Conference on Genetic Programming (EuroGP) 2014
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)
Question

How much runtime improvement can we achieve?
Software Specialisation with Transplants

- Sensitivity Analysis
- GP
- Non-functional property Test
- Test data
- Fitness
- Multi-donor transplant
  Specialised for a particular application domain
  17% faster

Justyna Petke, Mark Harman, William B. Langdon and Westley Weimer
*Using Genetic Improvement & Code Transplants to Specialise a C++ program to a Problem Class*
European Conference on Genetic Programming (EuroGP) 2014
Real-world cross-system transplantation

- Sensitivity Analysis
- Test data
- Search
- Fitness
- Non-functional property Test
Real-world cross-system transplantation

Donor feature

Sensitivity Analysis

Search

Fitness

Test data

Non-functional property Test

Host feature

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

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

ACM Distinguished Paper Award at ISSTA 2015

Gold ‘Humie’ Award Winner at GECCO 2016

Code ‘transplant’ could revolutionise programming

coverage in

over 2,000 shares of article in

BBC WORLD SERVICE

Click
Why Autotransplantation?

Why not handle H.264?

~100 players

Check open source repositories

Start from scratch

CREST

Genetic Improvement

Justyna Petke
Automated Software Transplantation

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

Donor
- ENTRY
- Organ

Host

Organ Test Suite
μTrans

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

Host
Donor

Implantation Point
Organ Entry

Organ Test Suite
Host Beneficiary

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>
<tr>
<td>x264</td>
<td>Donor</td>
<td>63</td>
</tr>
<tr>
<td>VLC</td>
<td>Host</td>
<td>422</td>
</tr>
</tbody>
</table>

- **Minimal size**: 0.4k
- **Max size**: 422k
- **Average Donor**: 16k
- **Average Host**: 213k

---

**Permission**
Request permissions from Permissions@acm.org.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires specific permission and/or a fee. To request such permission, write to Permissions@acm.org.
Experimental Methodology and Setup

- Host
  - Implantation Point
- Donor
  - OE

- Organ Test Suite

- 64 bit Ubuntu 14.10
- 16 GB RAM
- 8 threads
Empirical Study

in 12 out of 15 experiments
we successfully autotransplanted
new functionality
within 26 hours performed a task that took developers an avg of 20 days of elapsed time
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

o 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
Memory vs speed trade offs

1. Sensitivity Analysis
2. Search
3. Non-functional property Test
4. Test data
5. Fitness
Memory vs speed trade offs

- System malloc
- Sensitivity Analysis
- GP
- Test data
- Fitness
- Non-functional property Test
- System optimised malloc

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

Fan Wu, Westley Weimer, Mark Harman, Yue Jia and Jens Krinke

Deep Parameter Optimisation
Genetic and Evolutionary Computation Conference (GECCO) 2015
Reducing Energy Consumption

- Sensitivity Analysis
- Search
- Test data
- Fitness
- Non-functional property Test

Genetic Improvement

Justyna Petke
Reducing Energy Consumption

Energy consumption can be reduced by up to 25%

Bobby R. Bruce Justyna Petke Mark Harman
Reducing Energy Consumption Using Genetic Improvement
Conference on Genetic and Evolutionary Computation (GECCO 2015)
Grow and Graft
new functionality

? → Sensitivity Analysis → Search

Test data → Fitness

Non-functional property Test

Genetic Improvement

Justyna Petke
Grow and Graft
new functionality

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)
Genetic Improvement Applications

- Improving software efficiency
- Improving energy consumption
- Porting old code to new hardware
- Grafting new functionality into an existing system
- Specialising software for a particular problem class
- Other
Genetic Improvement Visibility

First International Workshop on Genetic Improvement at GECCO 2015, Madrid, Spain
www.geneticimprovementofsoftware.com

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

Second International Workshop on Genetic Improvement at GECCO 2016, Denver, Colorado
Genetic Improvement of Software

Functional Properties
- New Feature
- Functionality Improvement

Non-functional Properties
- Execution Time
- Memory
- Energy
Genetic Improvement of Software

Functional Properties
- New Feature
- Functionality Improvement
- Bug Repair

Non-functional Properties
- Execution Time
- Memory
- Bandwidth
- Energy
- Size
Pictures used with thanks from these sources

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

BBC_Micro: [Public domain], 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