GECCO 2014 – GP call for papers

The Genetic and Evolutionary Computation Conference will be held on Saturday 12 July to Wednesday 16 July 2014

Important dates

Abstract submission: January 15, 2014
Full papers: January 29, 2014

http://www.sigevo.org/gecco

http://www.sigevo.org/gecco-2014
Genetic Improvement Programming

W. B. Langdon
Centre for Research on Evolution, Search and Testing
Computer Science, UCL, London

GISMOE: Genetic Improvement of Software for Multiple Objectives
Genetic Programming to Improve Existing Software

• Why

• Background
  – GP to write software
  – GP to improve human written programs

• Examples
  – Demonstration systems, automatic bug fixing
  – Evolving code for a new environment (gzip)
  – Improving non-functional properties IEEE TEC
  – Faster parallel code for stereo imaging

• Implications
When to Automatically Improve Software

• When to use GP to create source code
  – Small. E.g. glue between systems “mashup”
  – Hard problems. Many skills needed.
  – Multiple conflicting ill specified non-functional requirements

• Genetic programming as tool. GP tries many possible options. Leave software designer to choose between best.
Tradeoff 2 objectives Pareto front

- Faster
- Less error
Some applications of Genetic Programming

• Most GP generates solutions, e.g.:
  – data modelling,
  – chemical industry: soft sensors,
  – design (circuits, lenses, NASA satellite aerial),
  – image processing,
  – predicting steel hardness,
  – cinema “boids”, Cliff hanger, Batman returns
Genetic Programming to Create Software

• GP has created real programs
  – domain specific hash functions
  – cache management
  – heap management, garbage collection
  – evolving communications protocols

• These can do better than existing standard approach by GP not only creating code but also tailoring it for specific use

W. B. Langdon, UCL
Genetic Programming to Improve Human written Programs

• Gluing together existing programs to create new functionality
  – combining object files
  – web services, mashup
GP to Improve human written programs

• Finch: evolve Java byte code
  – no compilation errors, 6 benchmarks
• Improving GPU shaders
• Functionality v speed or battery life

```cpp
int Factorial(int a) {
    if (a <= 0)
        return 1;
    else
        return (a * Factorial(a-1));
}
```

Factorial source code, 87% reduction in instructions, [white,2011]
GP Evolving Pareto Trade-Off

Movie to tradeoff between 2 objectives
GP Automatic Bug Fixing

• Run code: example to reproduce bug, a few tests to show fixed code still works.
• Search for replacement C statement within program which fixes bug.
• Real bugs in real C programs.
  – 1st prize Human-Competitive GECCO 2009 Gold Humie
GP Automatic Coding

• Show a machine optimising existing human written code to trade-off functional and non-functional properties.
  – E.g. performance versus:
    Speed or memory or battery life.

• Trade off may be specific to particular use. For another use case re-optimise

• Use existing code as test “Oracle”. (Program is its own functional specification)
GP Automatic Coding 2

• Target non-trivial open source system:
  – Bowtie2 state-of-the-art DNA lookup tool

• Tailor existing system for specific use:
  – nextgen DNA from 1000 genomes project

• Use existing system as test “Oracle”
  – Smith-Waterman exact algorithm (slow)

• Use inputs & answer to train GP.

• Clean up new code
Problems with BLAST

• BLAST contains biologists heuristics and approximations for mutation rates. It is the “gold standard” answer.
  – A few minutes per look up

• “Next Gen” DNA sequencing machines generate 100s millions short noisy DNA sequences in about a day.

• BLAST originally designed for longer sequences. Expects perfect data. Human genome database too big for PC memory.
Human Generated Solutions

• More than 140 bioinformatic sequence tools
• All human generated (man years)
• Many inspired by BLAST but tailored to
  – DNA or Proteins
  – Short or long sequences. Any species v man.
  – Noise tolerance. Etc. etc.
• Manual trade-off lose accuracy for speed
  – Bowtie 35million matches/hour but no indels
  – Bowtie2 more functionality but slower

W. B. Langdon, UCL
Why Bowtie 2?

- Target Bowtie2 DNA sequencing tool
  - 50000 line C++, 50 .cpp 67 .h files, scripts, makefile, data files, examples, documentation
- SourceForge
- New rewrite by author of successful C Bowtie

- Aim to tailor existing system for specific (important data source)

- 1000 genomes project
  - Project aims to map all human mutations
  - 100s millions of short human DNA sequences
  - Download raw data via FTP
Evolving Bowtie2

• Convert code to grammar
• Grammar used to both instrument code and control modifications to code
• Genetic programming manipulates patches
  • Small movement/deletion of existing code
  • New program source is syntactically correct
  • Compilation errors mostly variable out-of-scope
GP Evolving Patches to Bowtie2

Original code

BNF Grammar

Population of patches

Test cases

Select

Fitness

Patched code

Population of patches

Improved system

Mutation and Crossover
vhi = _mm_cmpeq_epi16(vhi, vhi); // all elts = 0xffff
vlo = _mm_xor_si128(vlo, vlo); // all elts = 0
vmax = vlo;

Lines 363-365 aligner_swsse_eu8.cpp

<aligner_swsse_eu8_363> ::= "" <_aligner_swsse_eu8_363>
      "{Log_count64++;/*28575*/}\n"<_aligner_swsse_eu8_363> ::= "vhi = _mm_cmpeq_epi16(vhi, vhi);"

<aligner_swsse_eu8_364> ::= "" <_aligner_swsse_eu8_364>
      "{Log_count64++;/*28576*/}\n"<_aligner_swsse_eu8_364> ::= "vlo = _mm_xor_si128(vlo, vlo);"

<aligner_swsse_eu8_365> ::= "" <_aligner_swsse_eu8_365>
      "{Log_count64++;/*28577*/}\n"<_aligner_swsse_eu8_365> ::= "vmax = vlo;"

Fragment of Grammar (Total 28765 rules)
7 Types of grammar rule

• Type indicated by rule name
• Replace rule only by another of same type
• 5792 statement (eg assignment, Not declaration)
• 2252 IF
  • <pe_118> ::= "\{Log_count64++;*/20254*/\} if " <IF_pe_118> " \{\n"
  • <IF_pe_118> ::= "(!olap)"
• 272 for1, for, for3
  • <sam_36> ::= "for(" <for1_sam_36> ";" <for2_sam_36> ";" <for3_sam_36> ") \{\n"
• 106 WHILE
  • <pat_731> ::= "while" <WHILE_pat_731> " \{\n"
  • <WHILE_pat_731> ::= "(true)"
• 24 ELSE
  • <aln_sink_951> ::= "else {" <ELSE_aln_sink_951> " \{Log_count64++;*/21439*/\};\n"
  • <ELSE_aln_sink_951> ::= "met.nunp_0++;"
Representation

- GP evolves patches. Patches are lists of changes to the grammar.
- Append crossover adds one list to another
- Mutation adds one randomly chosen change
- 3 possible changes:
  - Delete line of source code (or replace by “”, 0)
  - Replace with line of Bowtie2 (same type)
  - Insert a copy of another Bowtie2 line
Example Mutating Grammar

```cpp
<_aligner_swsse_ee_u8_707> ::= "vh = _mm_max_epu8(vh, vf);"
<_aligner_swsse_ee_u8_365> ::= "vmax = vlo;"
```

2 lines from grammar

```cpp
_vh = _mm_max_epu8(vh, vf);{Log_count64++;/*28919*/}
```

Instrumented original code

```cpp
vmax = vlo;{Log_count64++;/*28919*/}
```

New code

Fragment of list of mutations
Says replace line 707 of file aligner_swsse_ee_u8.cpp by line 365
Compilation Errors

- Use grammar to replace random line, only 15% compile. But if move <100 lines 82% compile.
- Restrict moves to same file, 45% compile
C++ is not fragile
Trading performance v speed

Random mutation GISMO bowtie2, WBL 3 May 2012

(noise added) 1757 identical
5435 changed

SLOWER
FASTER

Change in instructions

Change in Smith-Waterman

WORSE  BETTER
Recap

• Representation
  – List of changes (delete, replace, insert). New rule must be of same type

• Genetic operations
  – Mutation (append one random change)
  – Crossover (append other parent)

• Apply change to grammar then use it to generate new C++ source code.
Which Parts of Bowtie2 are Used

bowtie2 19,908 C++ lines, Longest query select_tsi_11.out2 WBL 14 Feb 2012

Times executed on DNA sequence with 9934 matches

W. B. Langdon, UCL
Scaling of Parts of Bowtie2

4 Heavily used Bowtie2 lines which scale differently
Focusing Search

<table>
<thead>
<tr>
<th>C++ Lines</th>
<th>Files</th>
<th>Bowtie2</th>
</tr>
</thead>
<tbody>
<tr>
<td>50745</td>
<td>50 .cpp, 67 .h</td>
<td>All C++ source files</td>
</tr>
<tr>
<td>19908</td>
<td>40 .cpp</td>
<td>no conditional compilation no header files.</td>
</tr>
<tr>
<td>2744</td>
<td>21 .cpp</td>
<td>no unused lines</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Weights target high usage</td>
</tr>
<tr>
<td>39</td>
<td>6 .cpp</td>
<td>evolve</td>
</tr>
<tr>
<td>7</td>
<td>3 .cpp</td>
<td>clean up</td>
</tr>
</tbody>
</table>
Testing Bowtie2 variants

• Apply patch generated by GP to instrumented version of Bowtie2
• “make” only compiles patched code
  – precompile headers, no gcc optimise
• Run on small but diverse random sample of test cases from 1000 genomes project
• Calculate fitness
• Each generation select best from population of patched Bowtie2
Fitness

• Multiple objective fitness
  • Compiles? No→no children
  • Run patched Bowtie2 on 5 example DNA sequences from The 1000 Genomes Project
  • Compare results with ideal answer (Smith-Waterman)
  • Sort population by
    – Number of DNA which don’t fail or timeout
    – Average Smith-Waterman score
    – Number of instrumented C++ lines executed (minimise)
  • Select top half of population.
• Mutate, crossover to give 2 children per parent.
• Repeat 200 generations
Run time errors

- During evolution 74% compile
- 6% fail at run time
  - 3% segfault
  - 2% cpulimit expired
  - 0.6% heap corruption, floating point (e.g. divide by zero) or Bowtie2 internal checks
- 68% run ok
GP Evolution Parameters

- Pop 10, 200 generations
- 50% append crossover
- 50% mutation (3 types delete, replace, insert)
- Truncation selection
- 5 test examples, reselected every generation
- ≈25 hours
Clean up evolved patch

• Allowed GP solution to grow big
• Use fixed subset (441 DNA sequences) of training data
• Remove each part of evolved patch one at time
• If makes new bowtie2 (more than a little) worse restore it else remove it permanently
• 39 changes reduced to 7
• Took just over an hour (1:08:38)
## Patch

<table>
<thead>
<tr>
<th>Weight</th>
<th>Mutation</th>
<th>Source File</th>
<th>Line</th>
<th>Type</th>
<th>Original Code</th>
<th>New Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>999</td>
<td>replaced</td>
<td>bt2_io.cpp</td>
<td>622</td>
<td>for2</td>
<td>i &lt; offsLenSampled</td>
<td>i &lt; this-&gt;_nPat</td>
</tr>
<tr>
<td>1000</td>
<td>replaced</td>
<td>sa_rescomb.cpp</td>
<td>50</td>
<td>for2</td>
<td>i &lt; satup_-&gt;offs.size()</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>disabled</td>
<td></td>
<td>69</td>
<td>for2</td>
<td>j &lt; satup_-&gt;offs.size()</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>replaced</td>
<td></td>
<td>707</td>
<td></td>
<td>vh = _mm_max_epu8(vh, vf); vmax = vlo;</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>deleted</td>
<td>aligner_sws.cpp</td>
<td>766</td>
<td></td>
<td>pvFStore += 4;</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>replaced</td>
<td>se_ee_u8.cpp</td>
<td>772</td>
<td></td>
<td>_mm_store_si128(pvHStore, vh); vh = _mm_max_epu8(vh, vf);</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>deleted</td>
<td></td>
<td>778</td>
<td></td>
<td>ve = _mm_max_epu8(ve, vh);</td>
<td></td>
</tr>
</tbody>
</table>

- Evolved patch 39 changes in 6 .cpp files
- Cleaned up 7 changes in 3 .cpp files
- 70+ times faster

offsLenSampled=179,215,892   _nPat=84
Results

- Patched code (no instrument) run on 200 DNA sequences (randomly chosen from same scanner but different people)
- Runtime 4 hours v. 12.2 days
- Quality of output
  - 89% identical
  - 9% output better (higher mean Smith-Waterman score). Median improvement 0.1
  - 0.5% same
  - 1.5% worse (in 4th and 6th decimal place).
Results

• Wanted to trade-off performance v. speed:
  – On “1000 genomes” nextgen DNA sequences
  – 70+ faster on average
  – Very small *improvement* in Bowtie2 results
Conclusions

• Genetic programming can automatically re-engineer source code. E.g.
  – hash algorithm
  – Random numbers which take less power, etc.
  – mini-SAT
• fix bugs (>10^6 lines of code, 16 programs)
• create new code in a new environment (graphics card) for existing program, gzip
• speed up GPU image processing
• speed up 50000 lines of code
GECCO 2014 – GP call for papers

The Genetic and Evolutionary Computation Conference will be held on Saturday 12 July to Wednesday 16 July 2014

GECCO Vancouver July 12-16, 2014
Genetic Programming

Important dates

Abstract submission: January 15, 2014
Full papers: January 29, 2014

http://www.sigevo.org/gecco

GECCO 2014 - 2014 - Genetic Evolutionary Computation Conference
Creating new programs - Crossover

Movie
Where does Bowtie2\textsuperscript{GP} improvement arise

Mostly identical. Improvement with DNA which makes Bowtie2 work hard. NB nonlinear Y-scale
counter increments added to instrument Bowtie2
Zipf’s Law

Distribution of exactly repeated Bowtie2 C++ lines of code after macro expansion, follows Zipf’s law, which predicts straight line with slope -1.
What my favourite number?
“Moore’s Law” in Sequences
The Genetic Programming Bibliography

http://www.cs.bham.ac.uk/~wbl/biblio/

9018 references and 8614 online publications

RSS Support available through the Collection of CS Bibliographies.

A web form for adding your entries. Co-authorship community. Downloads

A personalised list of every author’s GP publications.

blog.html

Search the GP Bibliography at http://liinwww.ira.uka.de/bibliography/Ai/genetic.programming.html