CREST

Genetic Programming to Improve Software

W. B. Langdon

Centre for Research on Evolution, Search and Testing

Computer Science, UCL, London

GISMOE: Genetic Improvement of Software for Multiple Objectives

Genetic Improvement Programming

- Why
- Background
  - What is Genetic Programming
  - GP to improve human written programs
- Examples
  - Demonstration systems, automatic bug fixing
  - Evolving code for a new environment (gzip)
  - Improving non-functional properties RN/12/09
- Implications

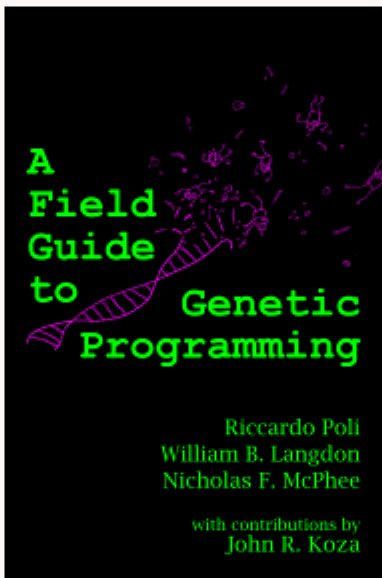
When to Automatically Improve Software

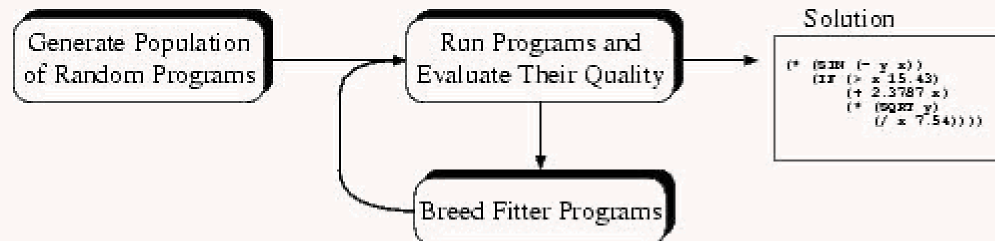
- When to use GP to create source code
  - Small. E.g. glue between systems “mashup”
  - Hard problems. Many skills needed.
  - Multiple conflicting ill specified non-functional requirements
- Genetic programming as tool. GP tries many possible options. Leave software designer to choose between best.

CREST

Genetic Programming

- A population of randomly created programs
  - whose fitness is determined by running them
  - Better programs are selected to be parents
  - New generation of programs are created by randomly combining above average parents or by mutation.
  - Repeat generations until solution found.

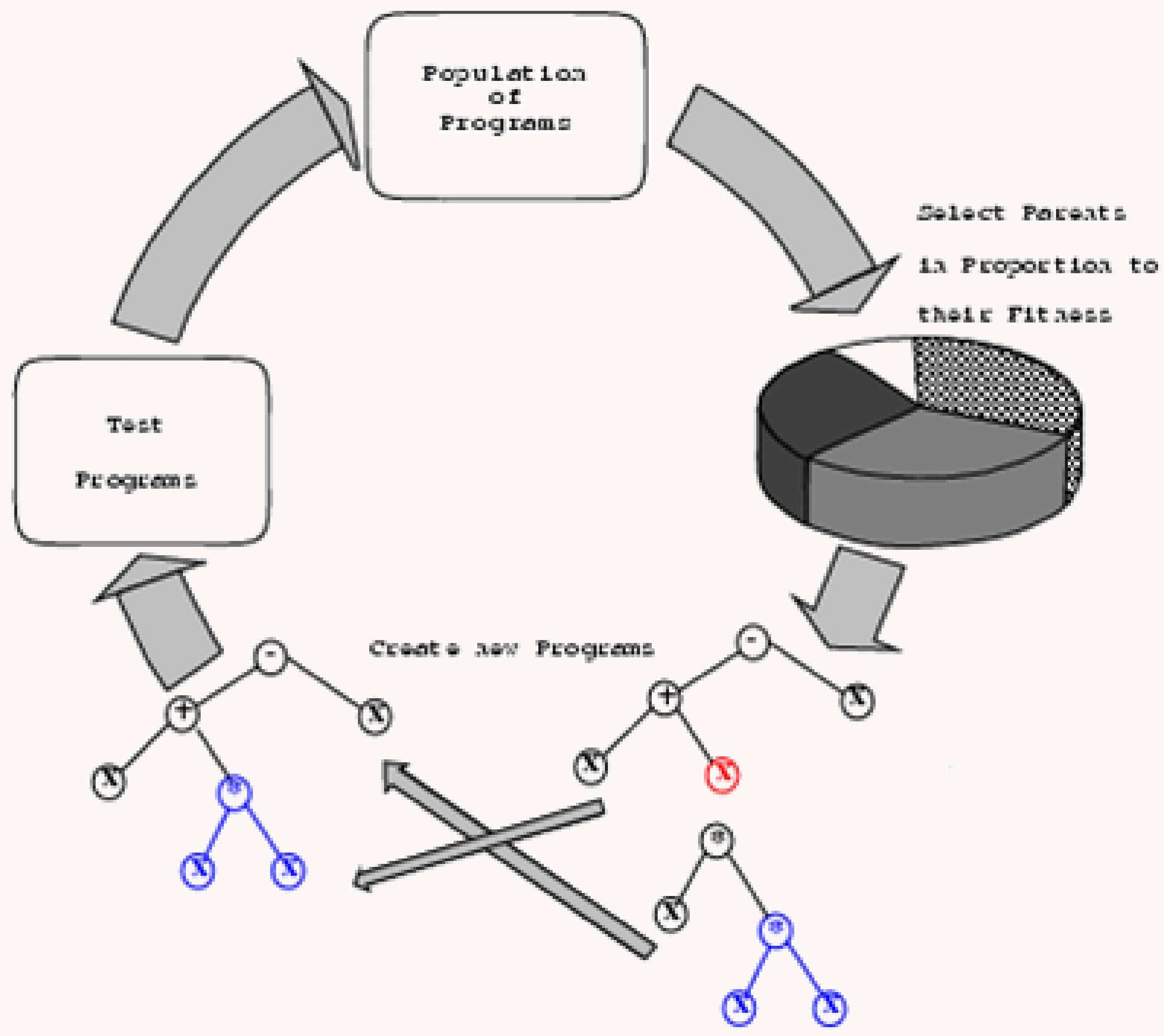




Free PDF

Free E-book

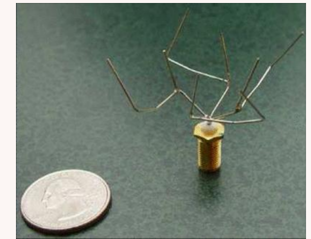
GP Generational Cycle

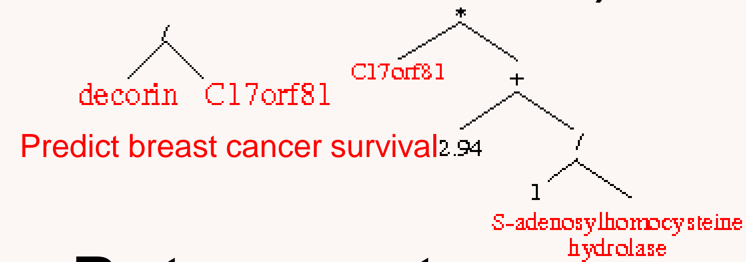


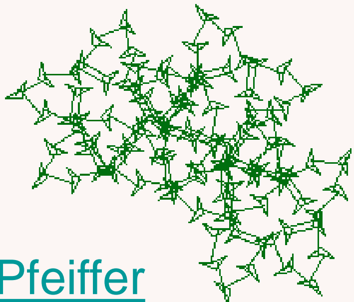
CREST

Some applications of Genetic Programming

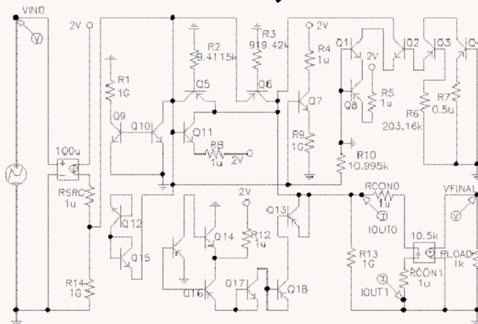
- Most GP generates solutions, e.g.:
  - data modelling,
  - chemical industry: soft sensors,
  - design (circuits, lenses, NASA satellite aerial),
  - image processing,
  - predicting steel hardness,
  - cinema “boids”, Cliff hanger, Batman returns

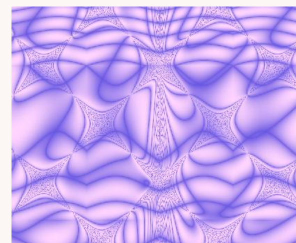


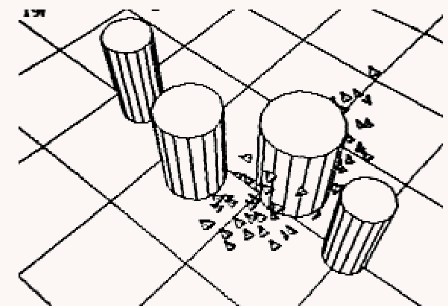




Pfeiffer







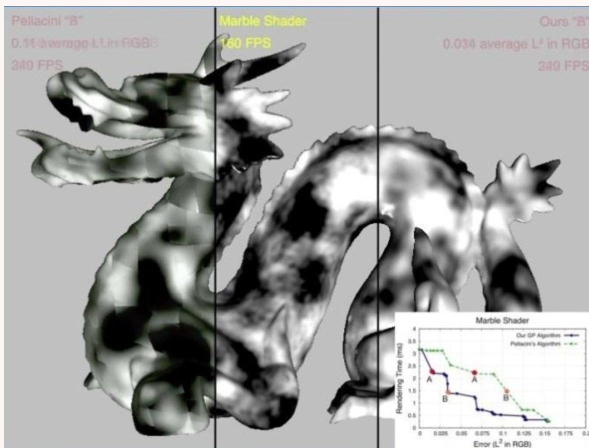
Genetic Programming to Improve Human written Programs

- Gluing together existing programs to create new functionality
  - combining web services, mashup
- Tailoring for specific use
  - domain specific hash functions
  - cache management
  - heap management, garbage collection
  - evolving communications protocols

CREST

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

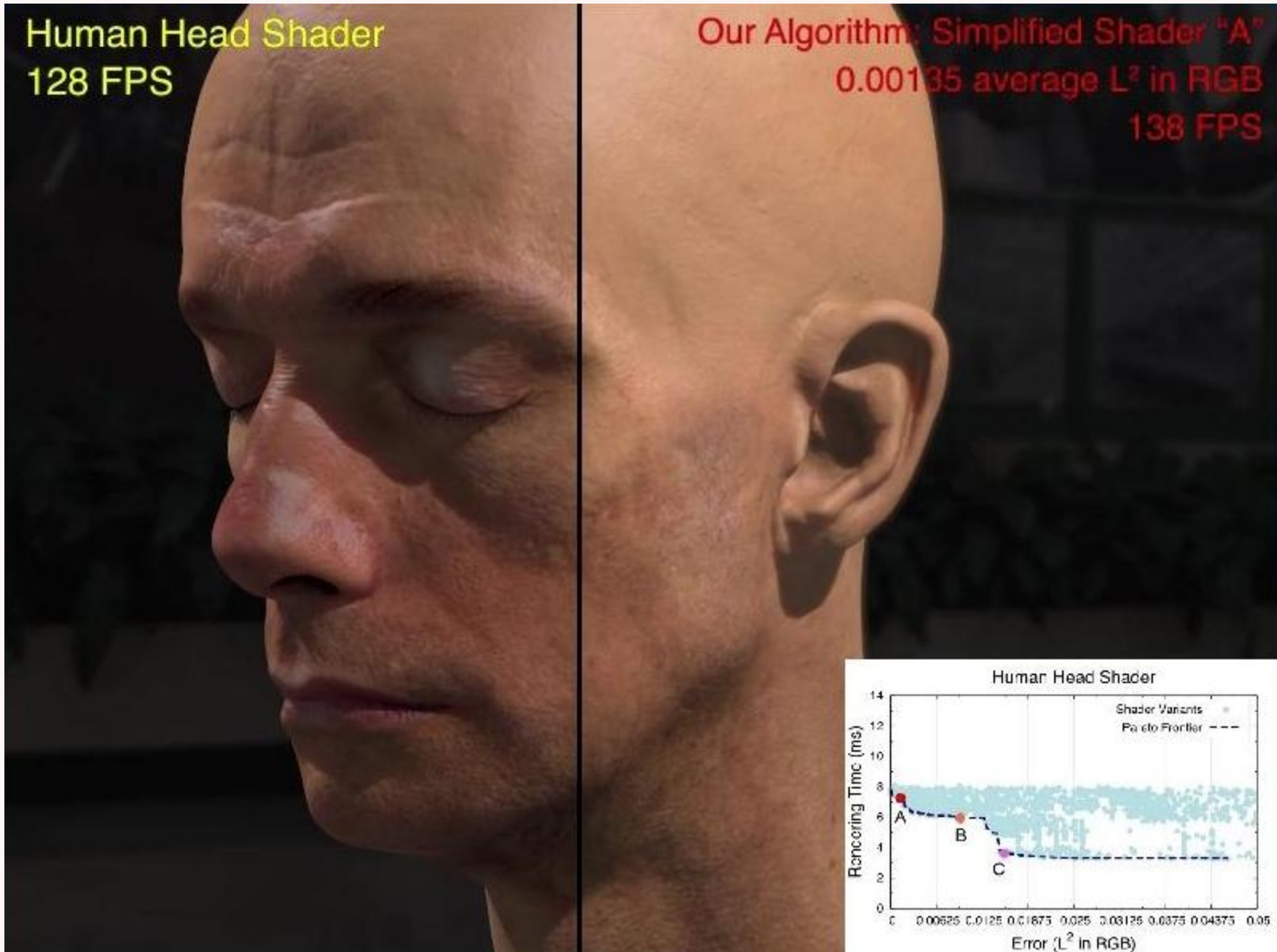



```
int Factorial(int a)
{
  if  (a <= 0)
      return  1;
  else
      return  (a * Factorial(a-1));
}
```

Factorial source code,
87% reduction in instructions, [white,2011]

CREST

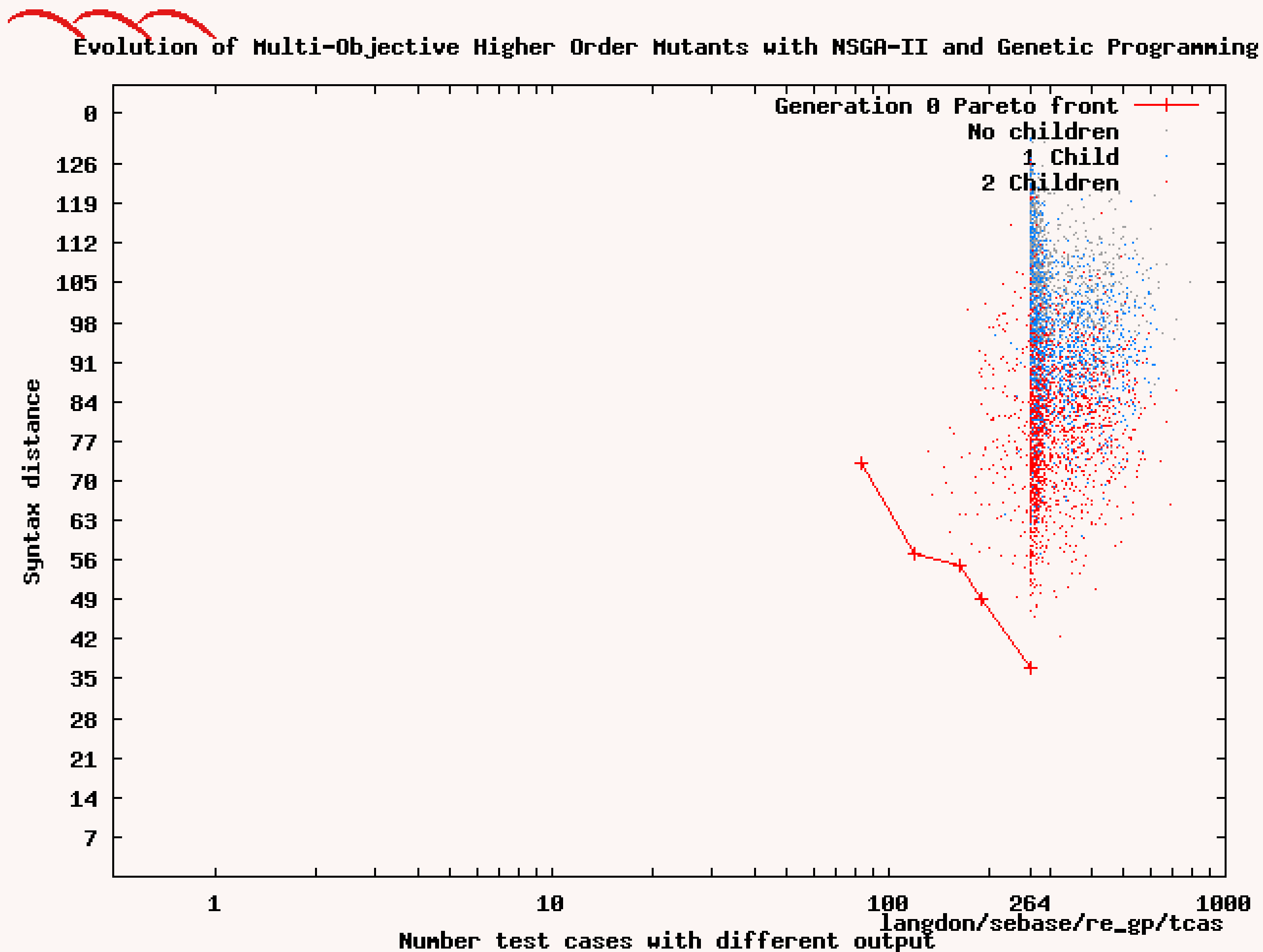
Improving GPU code



Sitthi-amorn, SIGGRAPH Asia 2011

CREST

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

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 modern DNA lookup tool
- Tailor existing system for specific use:
  - nextgen DNA from 1000 genome 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 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 BLAST functionality but slower

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 genome 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
  - New code is syntactically correct
  - Compilation errors mostly variable out-of-scope

GP Evolving Patches to Bowtie2

CREST

BNF Grammar

```
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_ee_u8.cpp

```
<aligner_swsse_ee_u8_363>  ::="" <_aligner_swsse_ee_u8_363>
.                                "{Log_count64++;/*28575*/}\n"
<_aligner_swsse_ee_u8_363> ::="vhi = _mm_cmpeq_epi16(vhi, vhi);"

<aligner_swsse_ee_u8_364>  ::="" <_aligner_swsse_ee_u8_364>
.                                "{Log_count64++;/*28576*/}\n"
<_aligner_swsse_ee_u8_364> ::="vlo = _mm_xor_si128(vlo, vlo);"

<aligner_swsse_ee_u8_365>  ::="" <_aligner_swsse_ee_u8_365>
.                                "{Log_count64++;/*28577*/}\n"
<_aligner_swsse_ee_u8_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

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

2 lines from grammar

```
<_aligner_swsse_ee_u8_707><_aligner_swsse_ee_u8_365>
```

Fragment of list of mutations

Says replace line 707 of file aligner_swsse_ee_u8.cpp by line 365

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

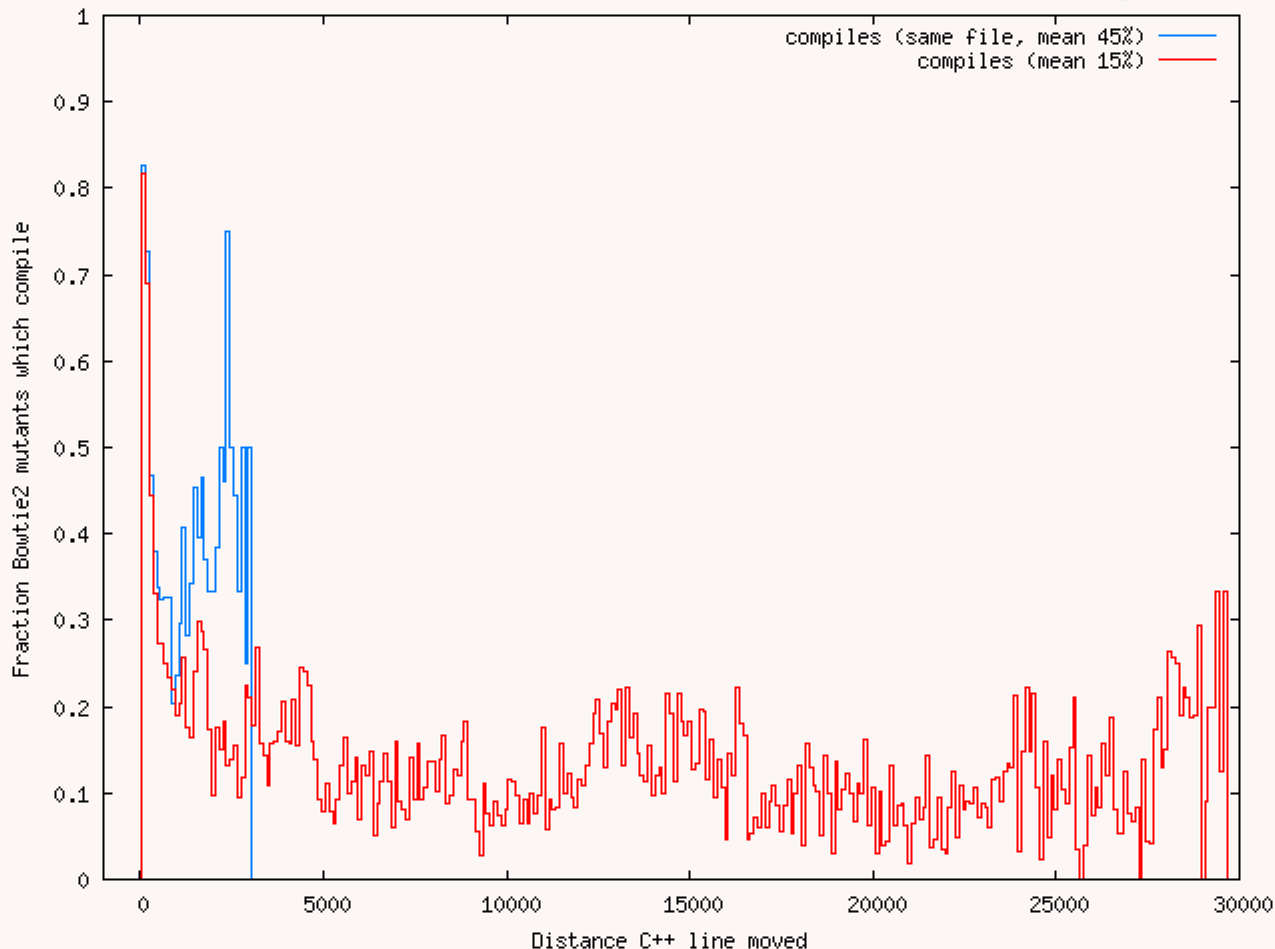
Instrumented original code

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

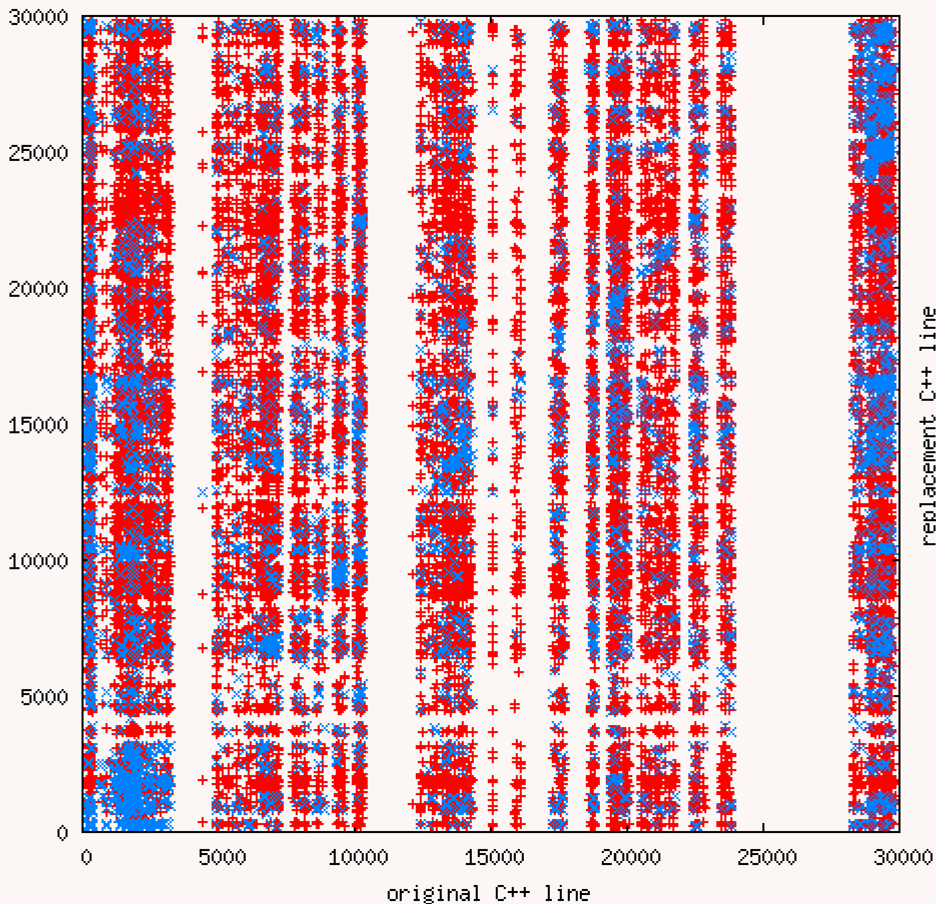
New code

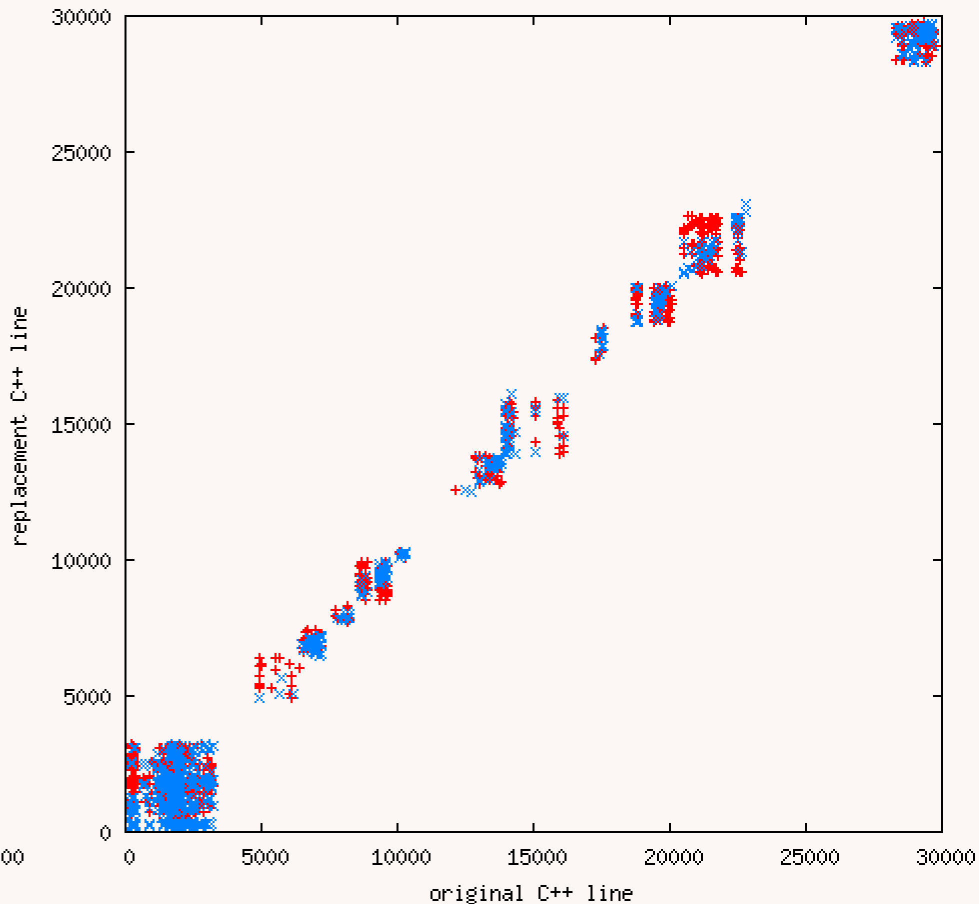
Compilation Errors

- Use grammar to replace random line, only 15% compile. But if move <100 lines 82% compile.
- Restrict moves to same file, 45% compile



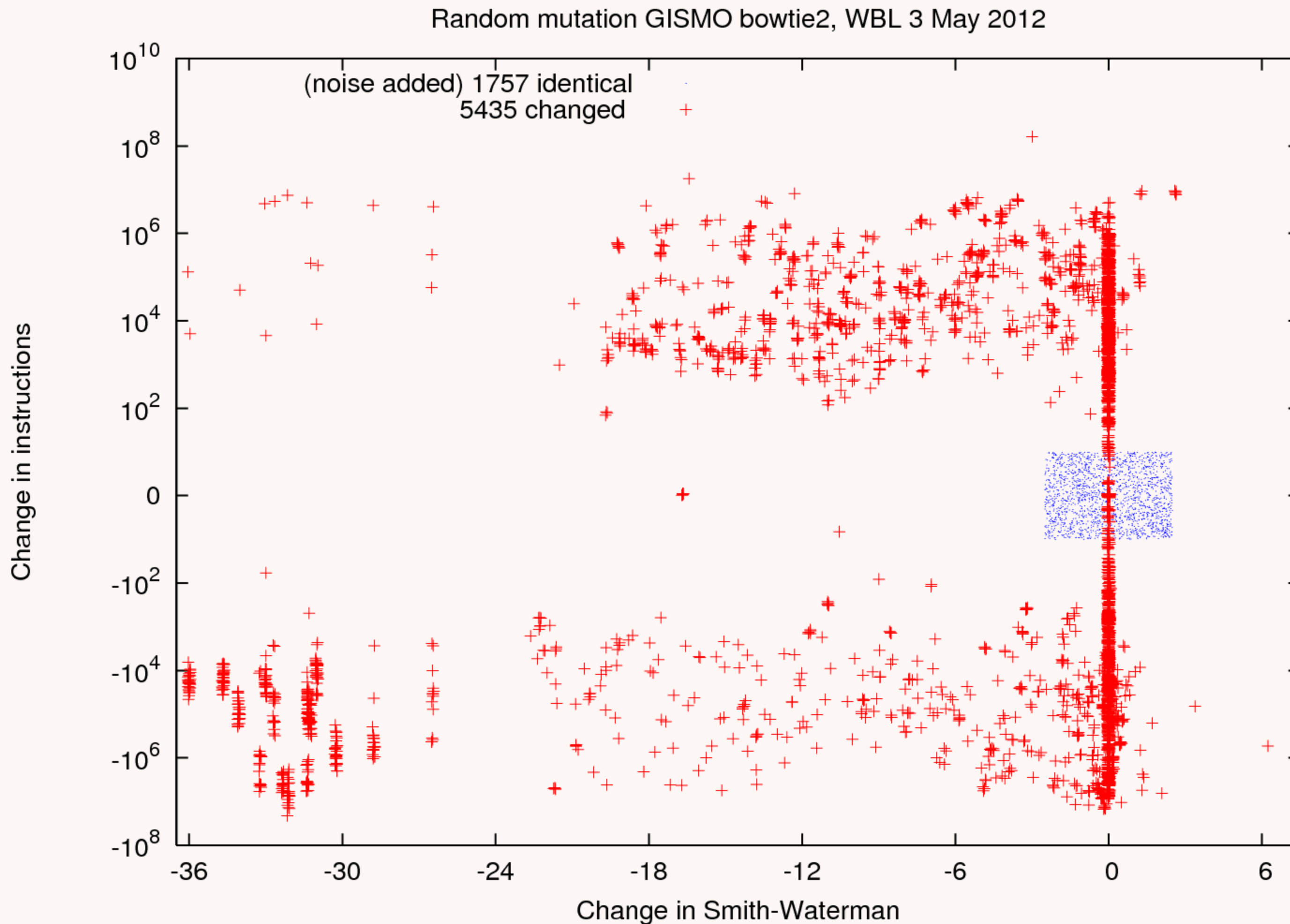
Reducing Compilation Errors





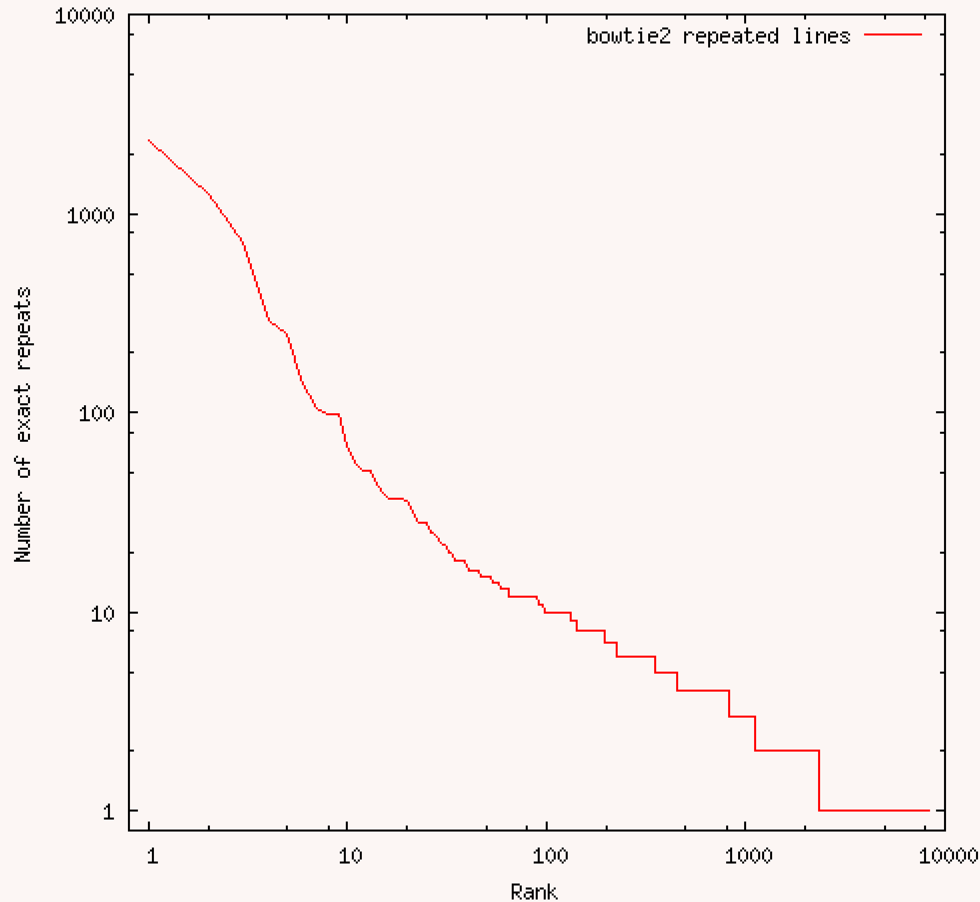
CREST

C++ is not fragile
Trading performance v speed



CREST

Zipf's Law

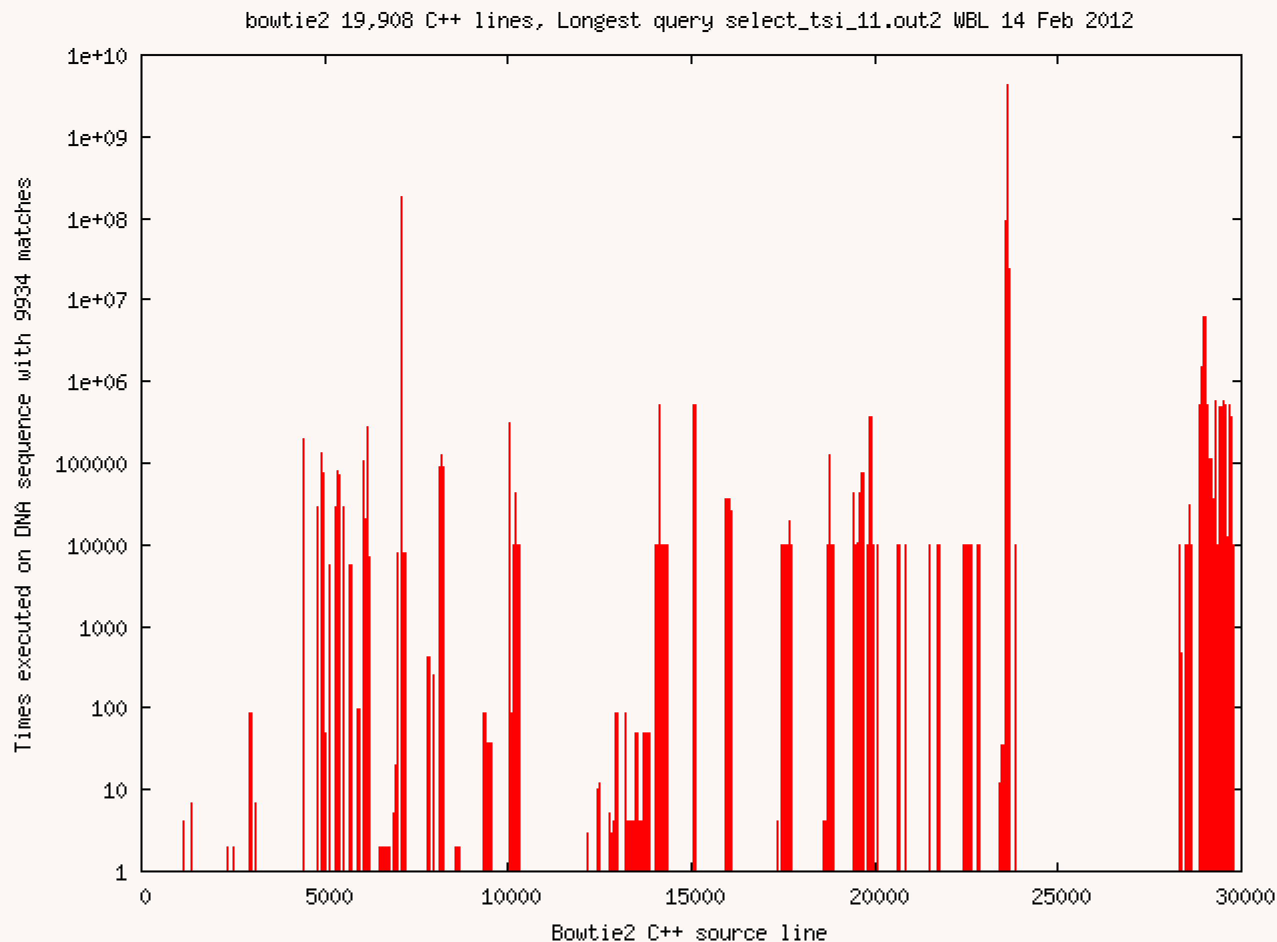


Distribution of exactly repeated Bowtie2 C++ lines of code after macro expansion, follows Zipf's law, which predicts straight line with slope -1.

Recap

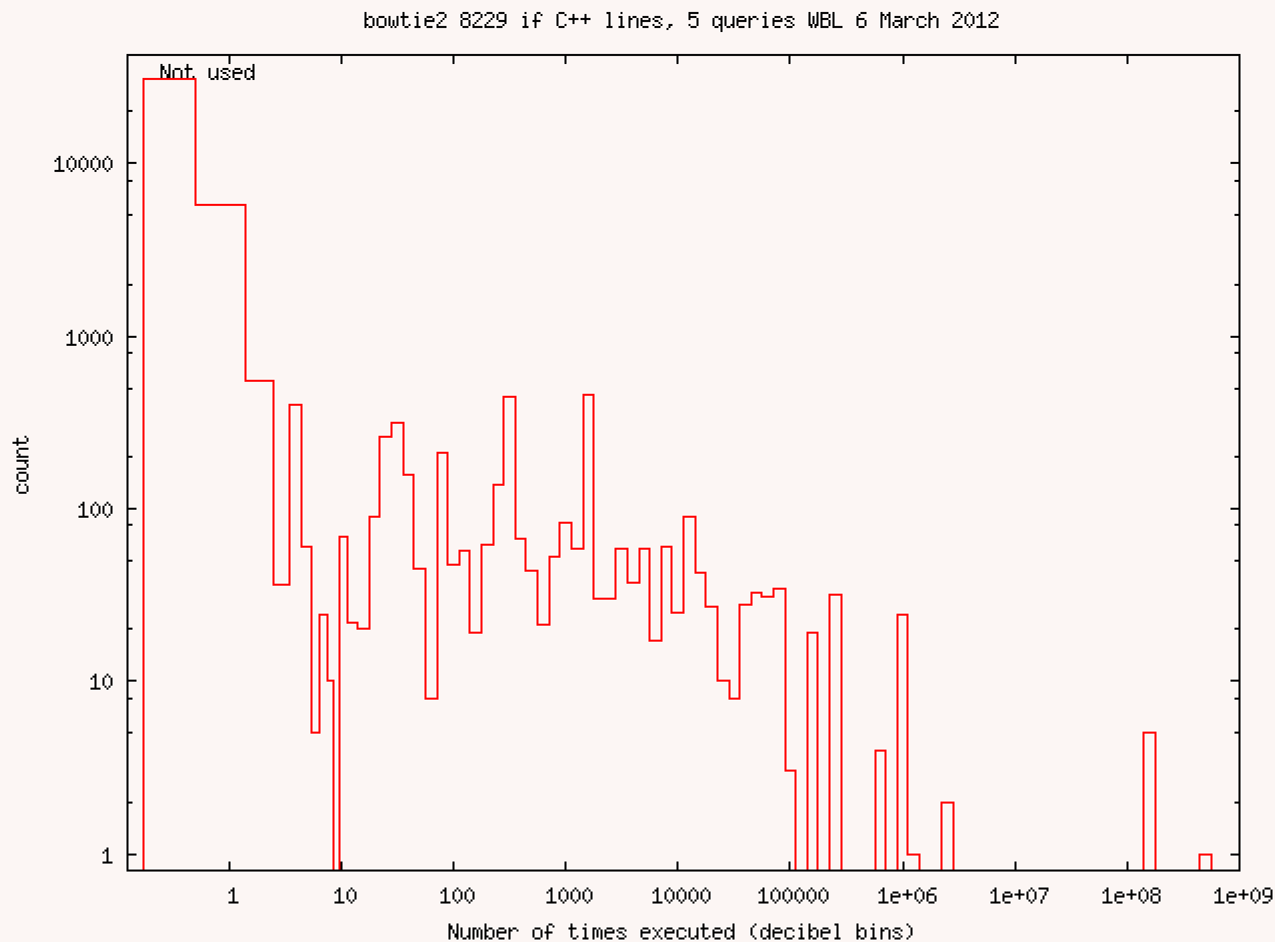
- Representation
  - List of changes (delete, replace, insert). New rule must be of same type
- Genetic operations
  - Mutation (append one random change)
  - Crossover (append other parent)
- Apply change to grammar then use it to generate new C++ source code.

Which Parts of Bowtie2 are Used



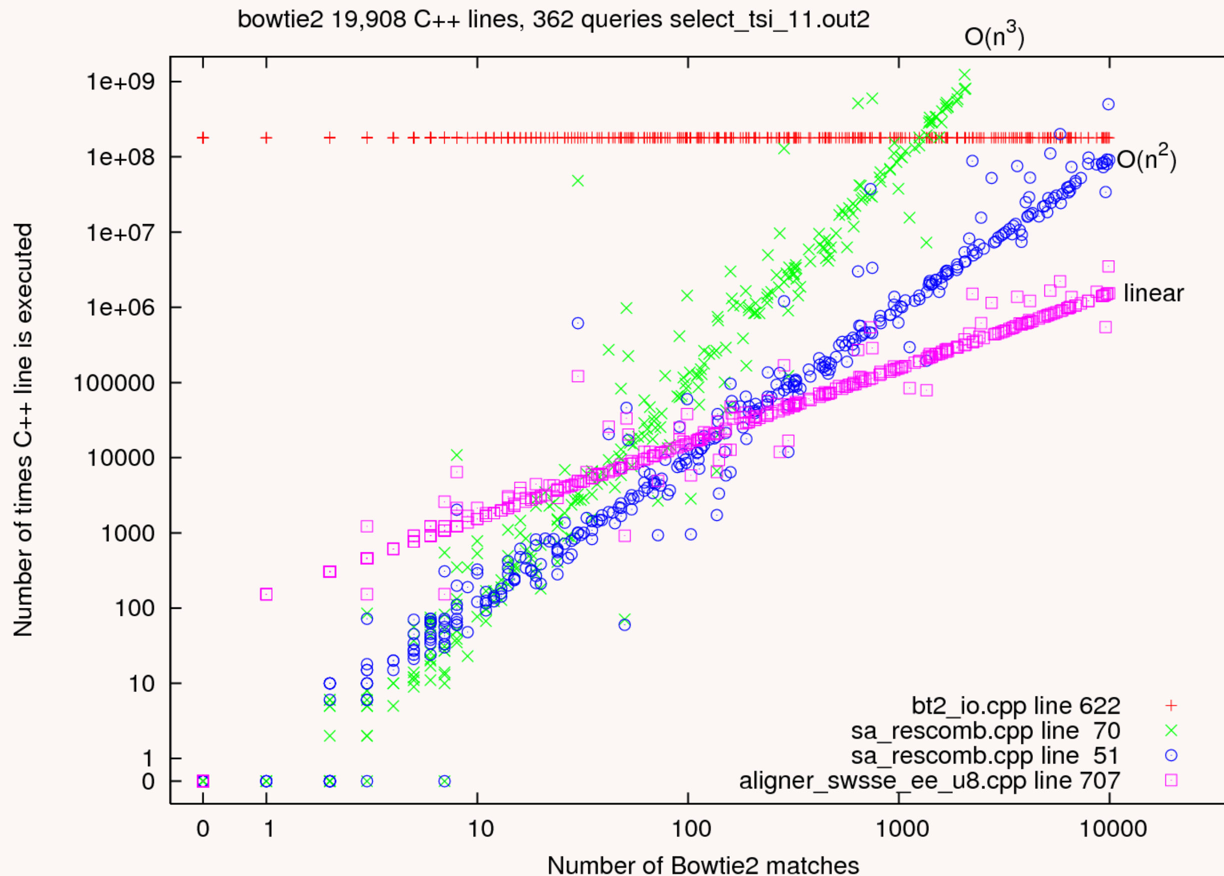
CREST

Histogram of Bowtie2 lines used



CREST

Scaling of Parts of Bowtie2

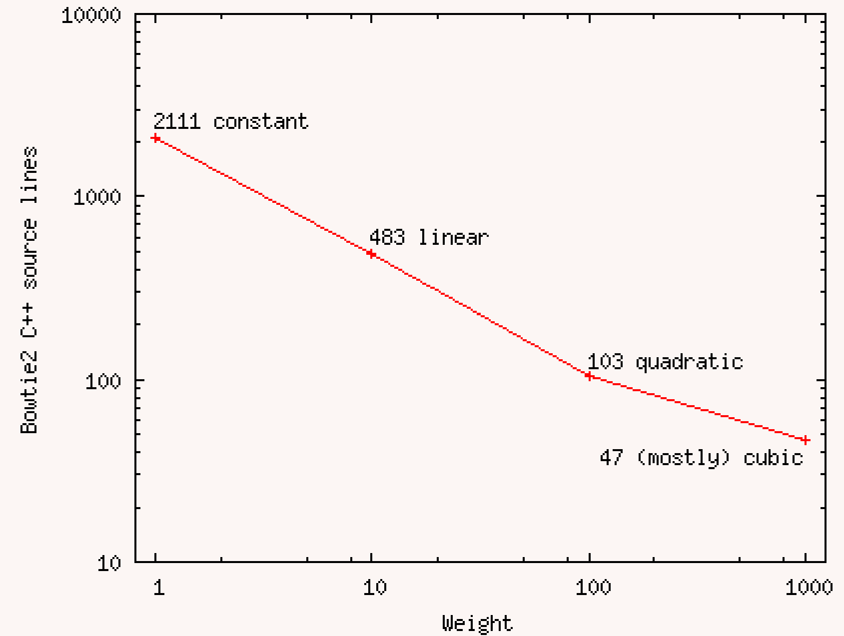


4 Heavily used Bowtie2 lines which scale differently

CREST

Focusing Search

C++ Lines	Files	Bowtie2
50745	50 .cpp, 67 .h	All C++ source files
19908	40 .cpp	no conditional compilation no header files.
2744	21 .cpp	no unused lines
		Weights target high usage
39	6 .cpp	evolve
7	3 .cpp	clean up



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 genome 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 (selected from 1000 genome FTP site, see RN/12/09)
  - 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

Weight	Mutation	Source file	line	type	Original Code	New Code
999	replaced	bt2_io.cpp	622	for2	i < offsLenSampled	i < this->_nPat
1000	replaced	sa_rescomb .cpp	50	for2	i < satup_->offs.size()	0
1000	disabled		69	for2	j < satup_->offs.size()	
100	replaced	aligner_swsse_ee_u8.cpp	707		vh = _mm_max_epu8(vh, vf);	vmax = vlo;
1000	deleted		766		pvFStore += 4;	
1000	replaced		772		_mm_store_si128(pvHStore, vh);	vh = _mm_max_epu8(vh, vf);
1000	deleted		778		ve = _mm_max_epu8(ve, vh);	

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

Results

- Patched code (no instrument) run on 200 DNA sequences (randomly chosen from same scanner but different people)
- Runtime 3:56:01 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 genome” nextgen DNA sequences
  - 70+ faster on average
  - Very small *improvement* in Bowtie2 results
- Trade off specific to particular use. For another use case re-optimise

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. WCCI 2010
- speed up 50000 lines of code

END

http://www.cs.ucl.ac.uk/staff/W.Langdon/

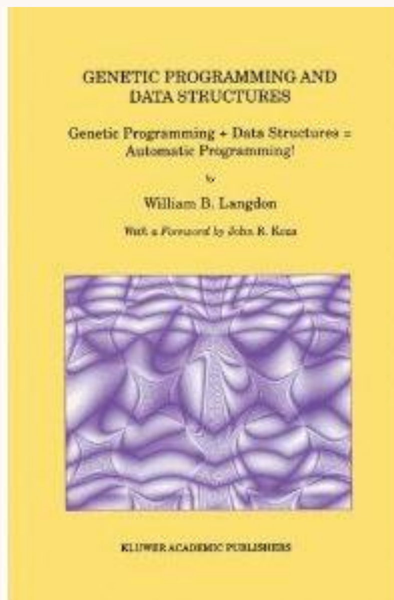
CREST

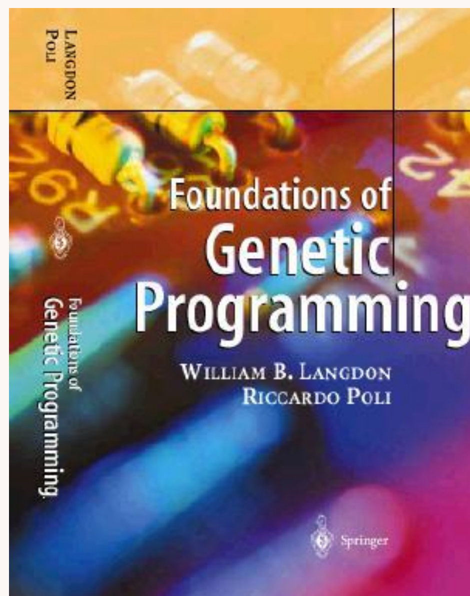
Genetic Improvement Programming

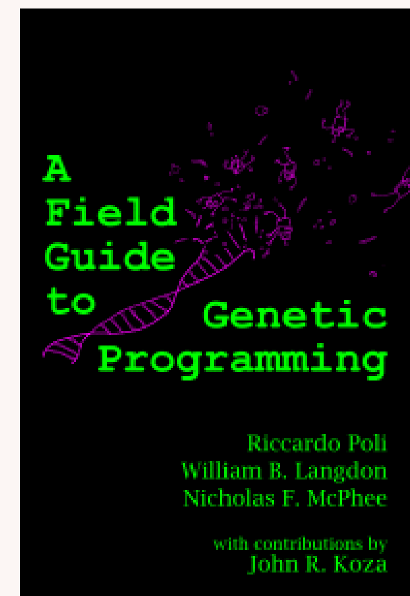
W. B. Langdon

CREST

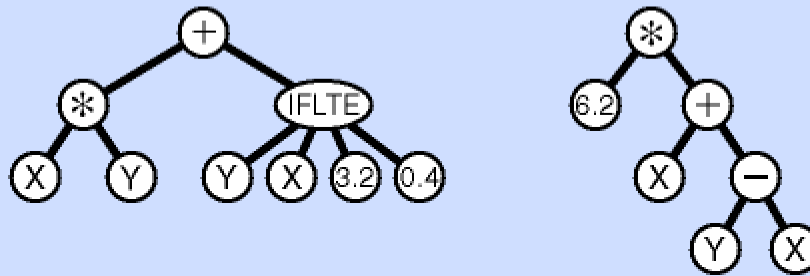
Department of Computer Science







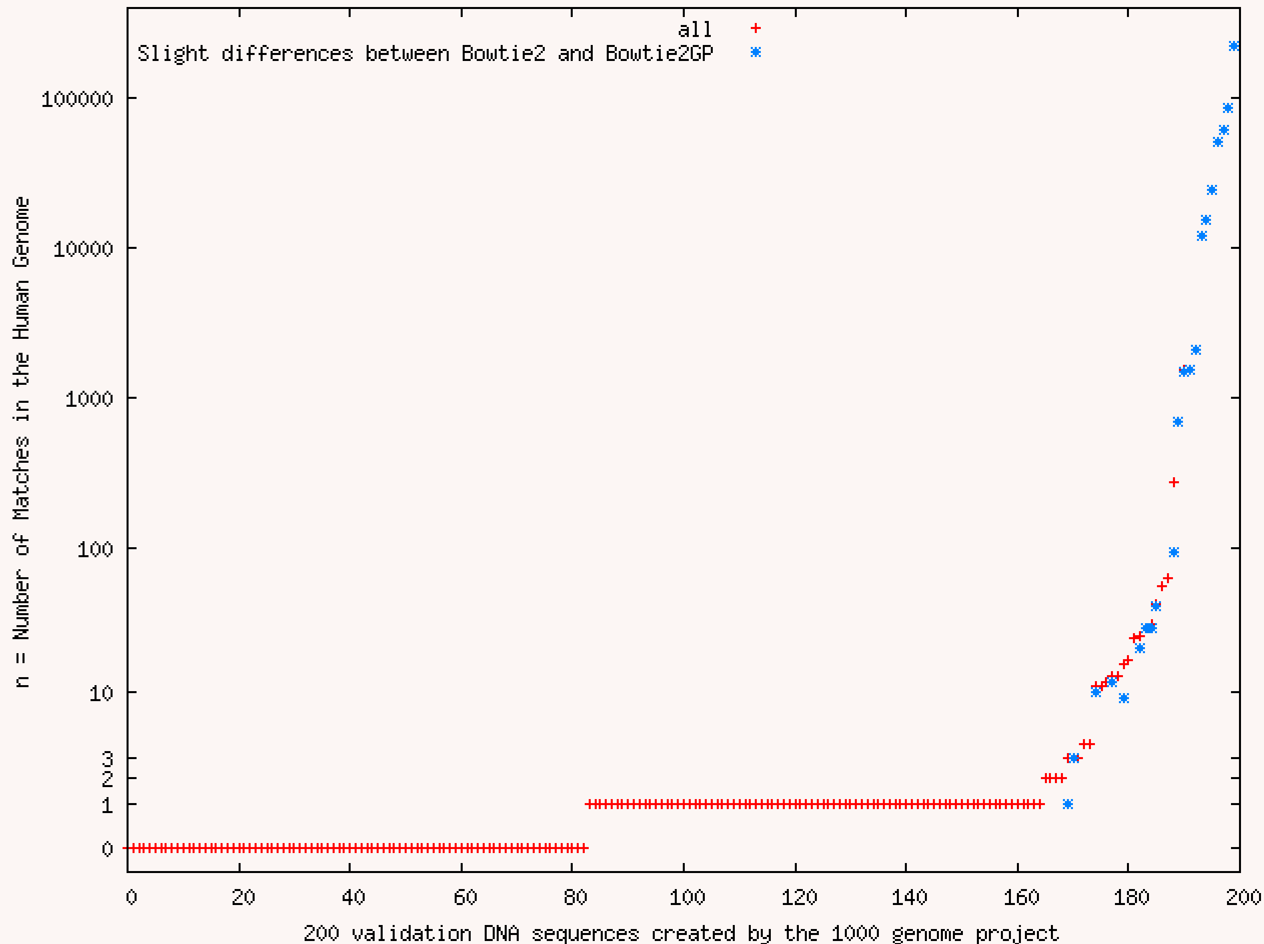
Creating new programs - Crossover



Movie

CREST

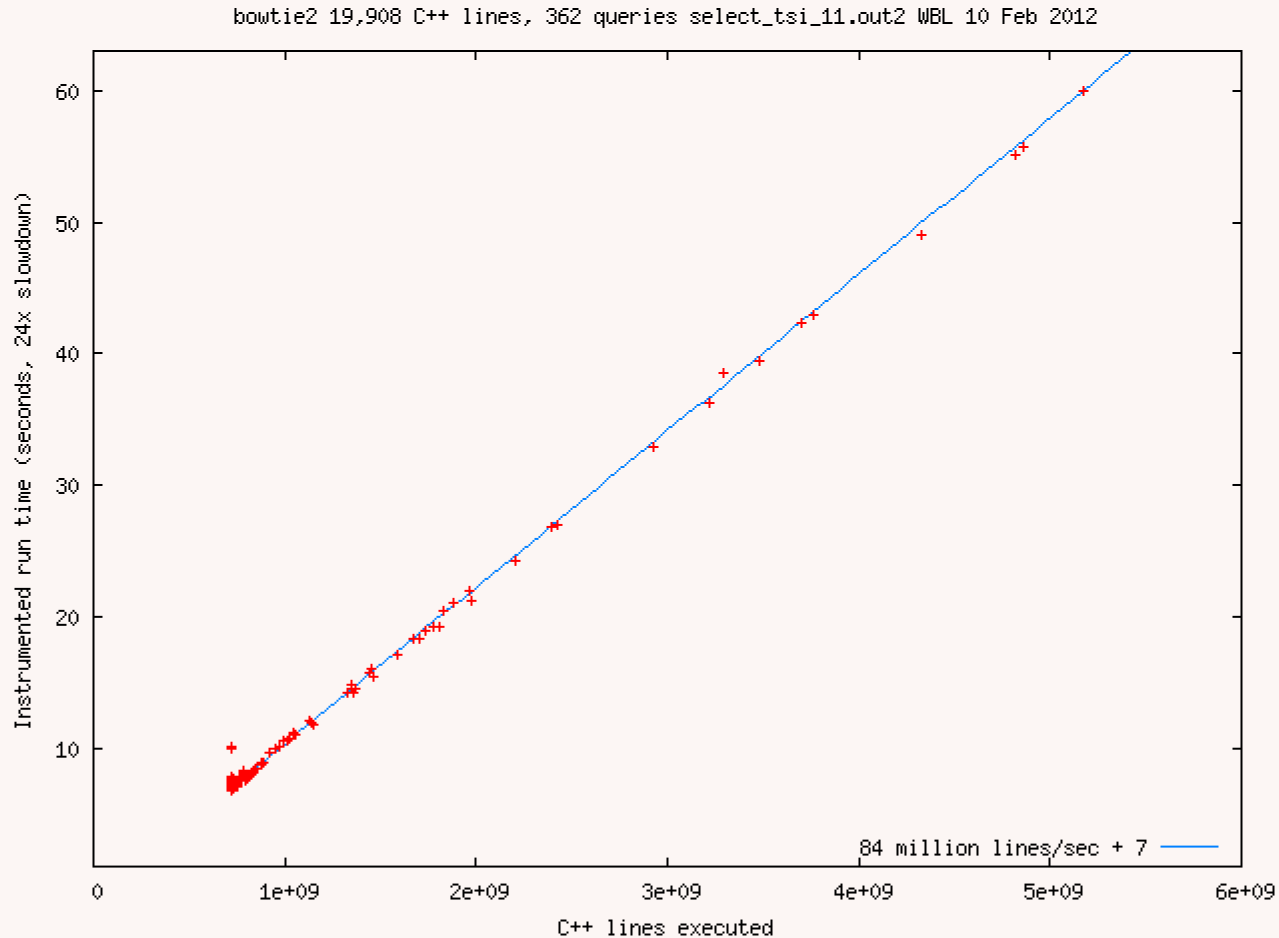
Where does Bowtie2^{GP} improvement arise



Mostly identical. Improvement with DNA which makes Bowtie2 work hard. NB nonlinear Y-scale

CREST

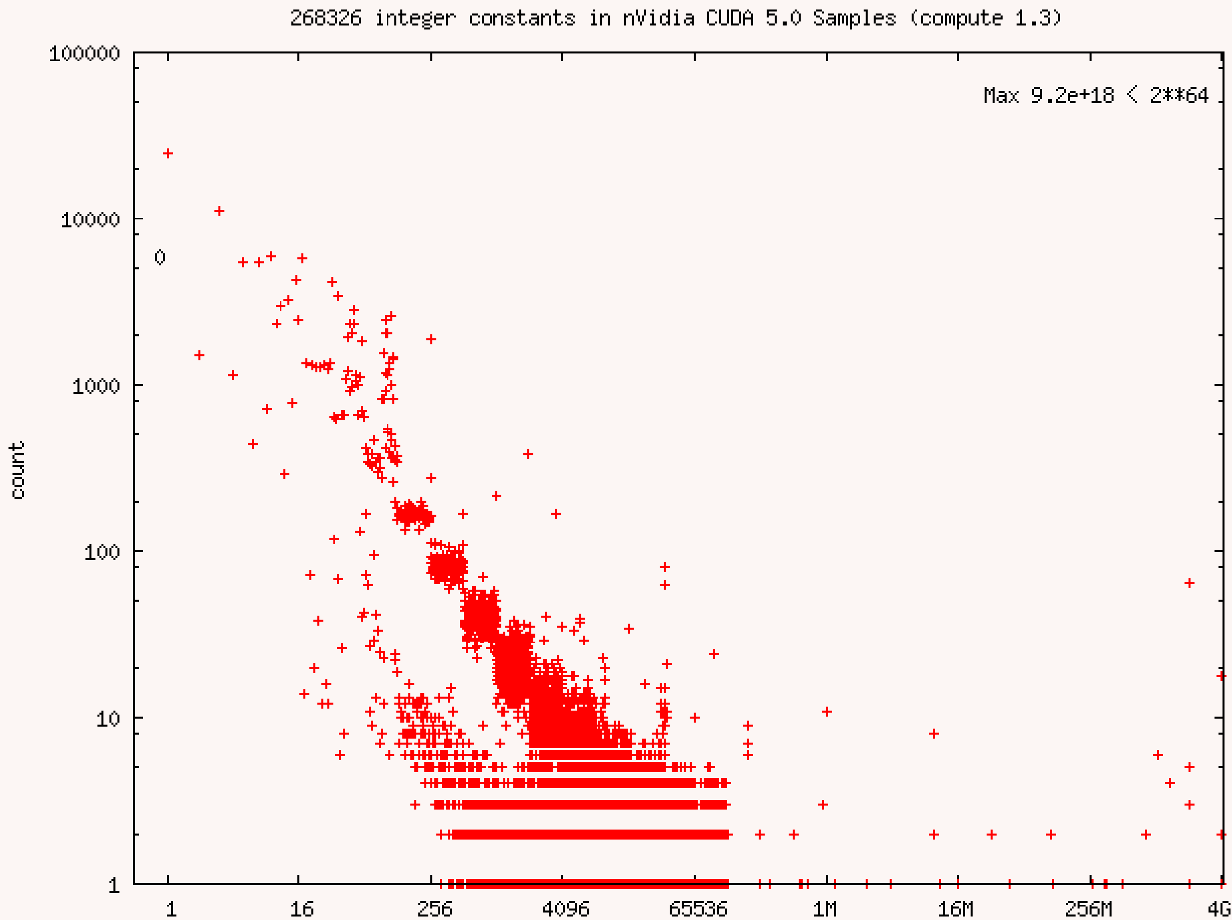
Instrumented Bowtie2



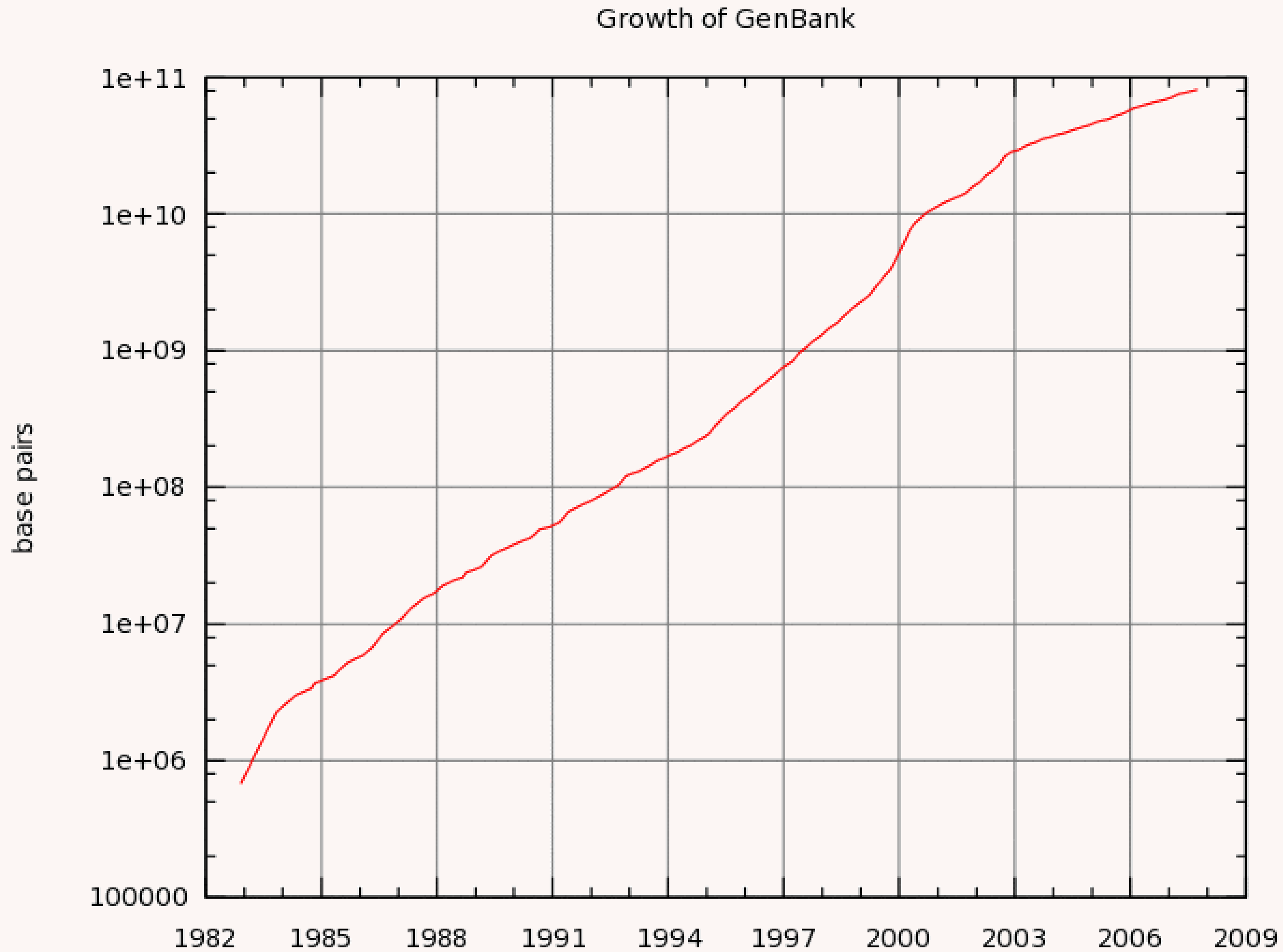
counter increments added to instrument Bowtie2

CREST

What my favourite number?



"Moore's Law" in Sequences



The Genetic Programming Bibliography

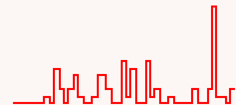
http://www.cs.bham.ac.uk/~wbl/biblio/

8601 references and 8197 online publications

RSS Support available through the Collection of CS Bibliographies.

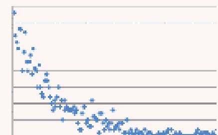
A web form for adding your entries. Co-authorship community. Downloads

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