

Genetic Improvement of Software for Multiple Objectives

SSBSE 2015, Bergamo

Monday 7 September 2015 9:00-10:30

W. B. Langdon

Department of Computer Science



Genetic Improvement [special issue](#) of Genetic Programming and Evolvable Machines, deadline 19 December 2015



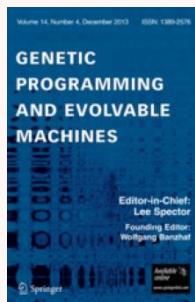
Genetic Improvement of Software for Multiple Objectives

W. B. Langdon

Department of Computer Science



Genetic Improvement [special issue](#) of Genetic Programming and Evolvable Machines, deadline 19 December 2015



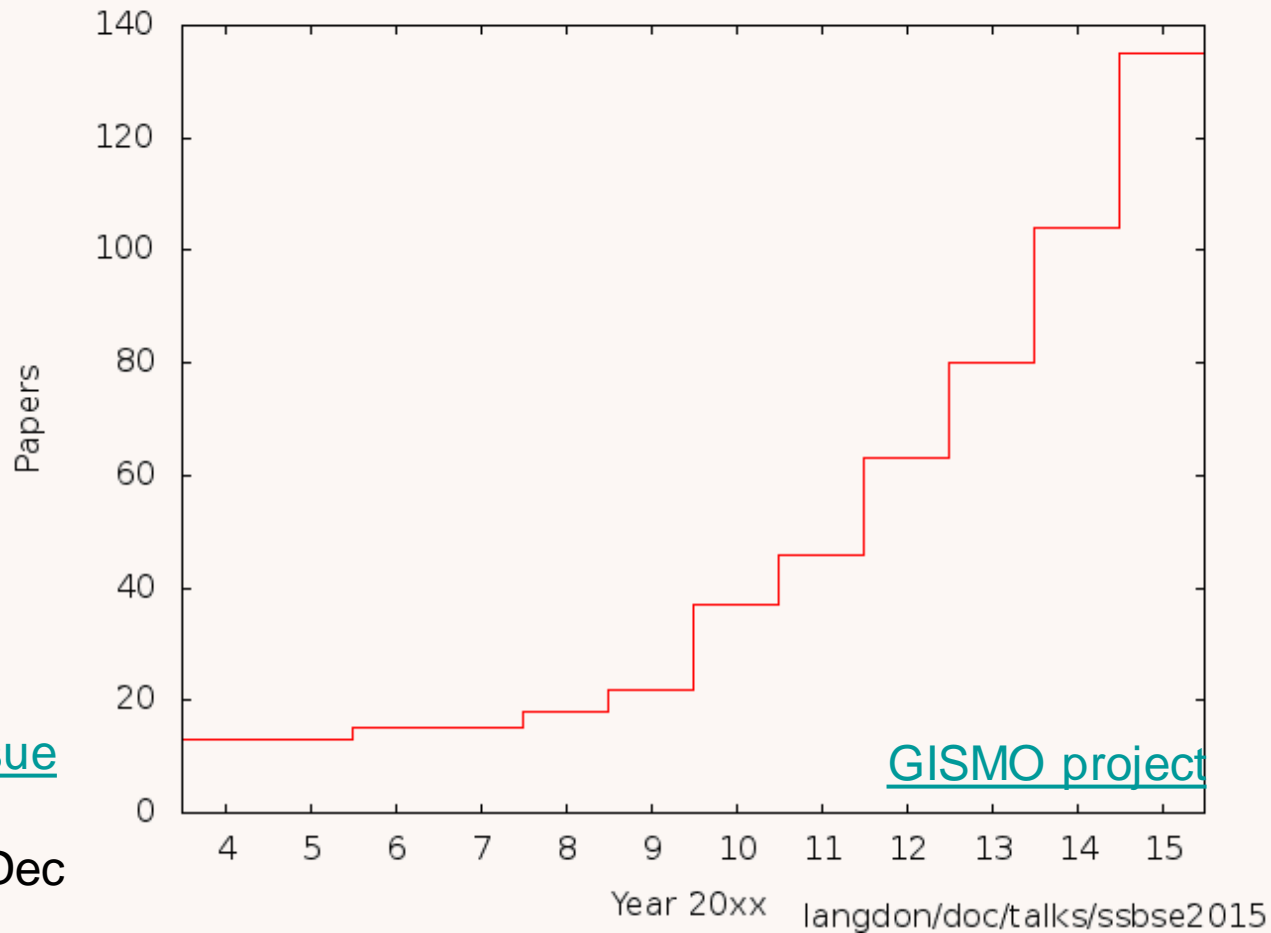
Genetic Improvement of Software for Multiple Objectives

- Introduction
 - Title of talk and [paper](#) from [EPSRC project](#)
 - Genetic Programming example applications
 - GP to create whole programs
 - GI to improve human written programs
- Examples
 - Demonstration systems, automatic bug “fixing”
 - Other Genetic Improvement examples

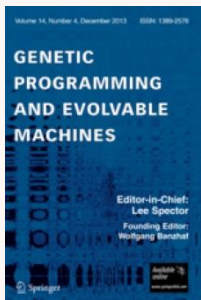
<discussion>

 - Evolving 50000 lines of C++

Recent Growth in Genetic Improvement



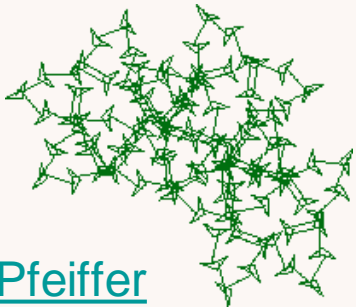
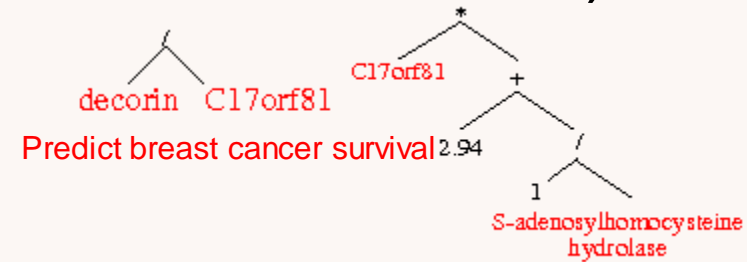
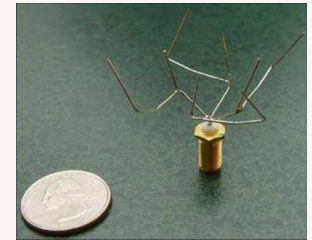
GI special issue
GP+EM
deadline 19 Dec



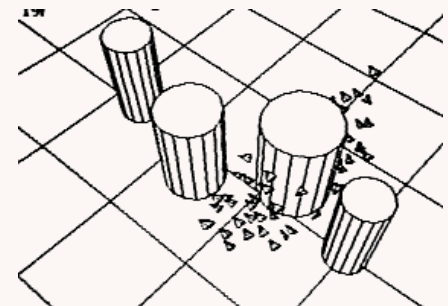
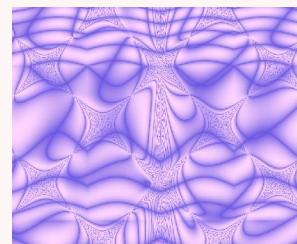
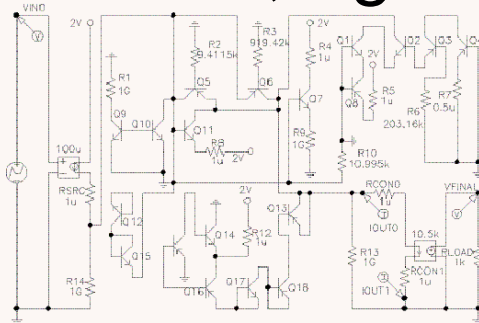
GI entries in GP bibliography 15 Aug 2015

Some applications of Genetic Programming

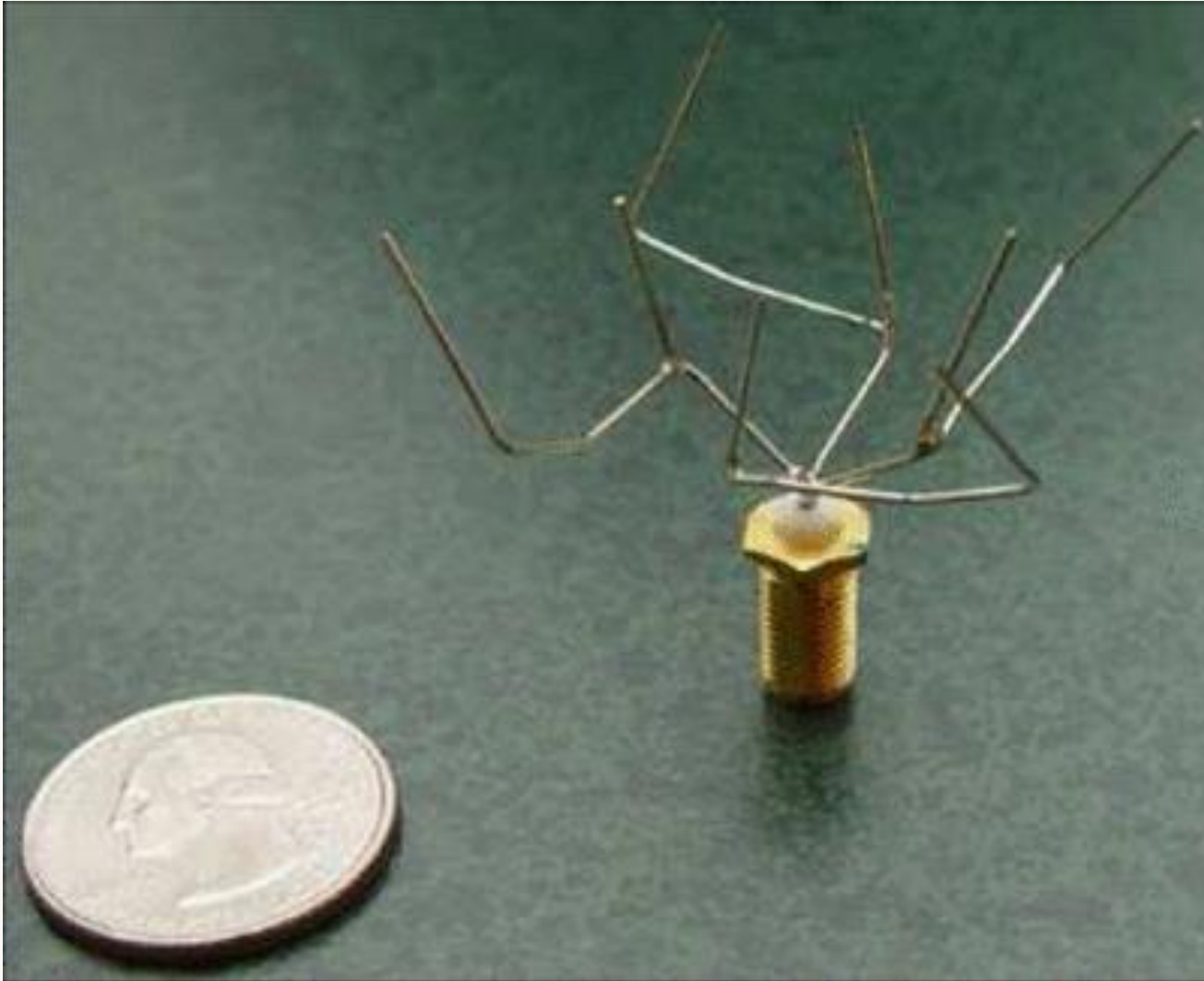
- Most GP generates solutions, e.g.:
 - data modelling,
 - chemical industry: soft sensors,
 - design (circuits, lenses, NASA satellite aerial),
 - image processing,
 - predicting steel hardness,
 - cinema “boids”, eg Cliffhanger



Pfeiffer



NASA satellite ST5 x-band aerial



[ST5-3-10](#)

Genetic Programming to Create Software

- GP has created real programs
 - domain specific hash functions
 - cache management
 - heap management, garbage collection
- These can do better than existing standard approach by GP not only creating code but also tailoring it for specific use



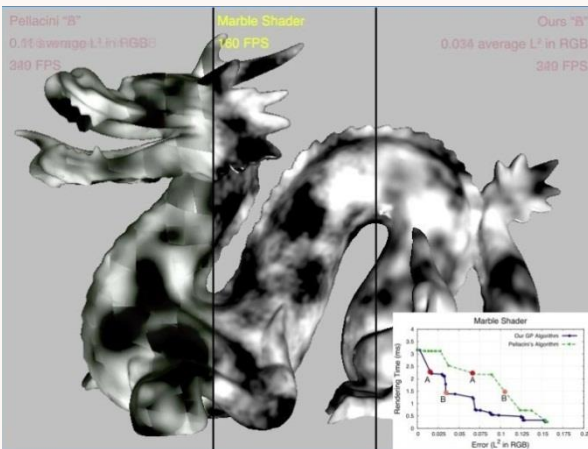
Genetic Programming to Compose Human written Programs

- Gluing together existing programs to create new functionality
 - combining object files
 - web services, mashup
 - [telephone services](#) (Marconi, UK)

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

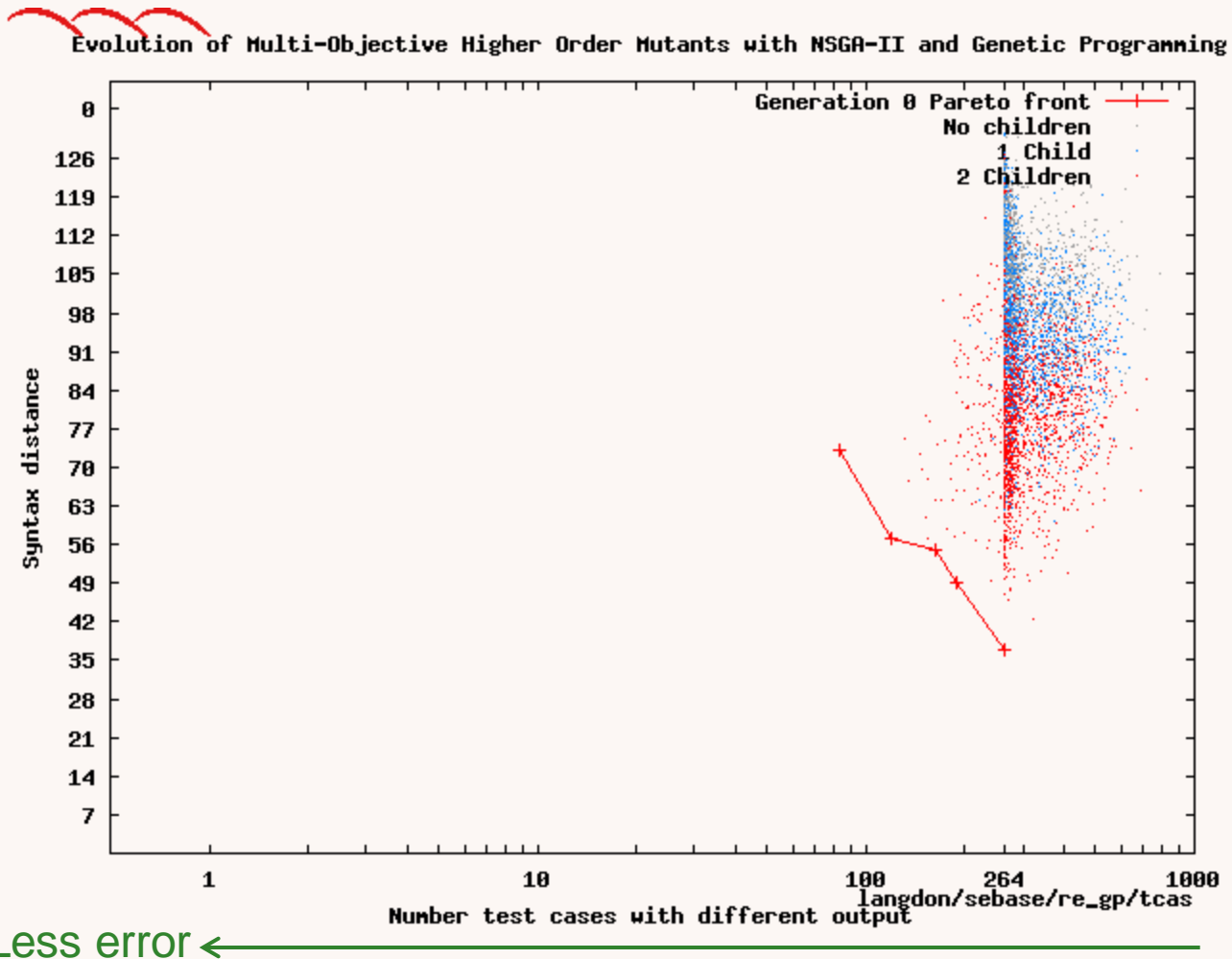
Crest energy [COW 39](#)



```
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

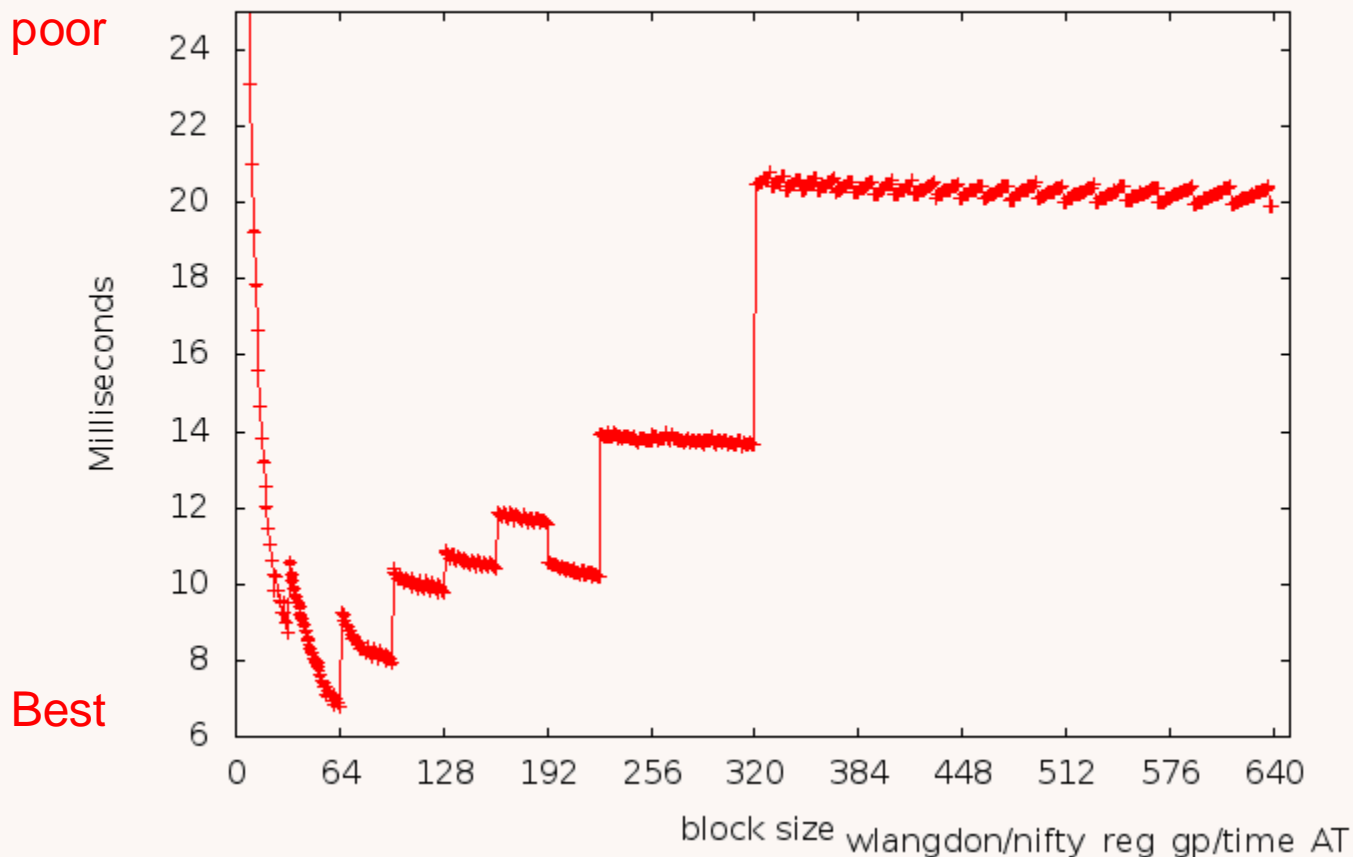
- Run code: example to reproduce bug, a few tests to show fixed code still works.
- Search for replacement C statement within program which “fixes” bug.
- Real bugs in real C programs.
 - 1st prize Human-Competitive GECCO 2009 Gold [Humie](#) \$5000

Improving Parallel GPU Code

Parameter sweeps, eg block size.

Multiple parameters use intelligent search, eg GAs.

Tesla C2050 CUDA nvcc 5.0, WBL 4 Jan 2014



Improving Parallel GPU Code

- Creating parallel code, eg [gzip](#)
- Optimising existing CUDA programs
 - [stereo vision](#)
 - [3D medical brain scans](#)
 - Introducing new functionality into [pknotsRG](#), up to 10000 times faster.
 - DNA lookup [BarraCUDA](#)
GI incorporated into released version and has been downloaded 631 times.

Improving Machine Generated Code

- [pknotsRG](#) 10000 speedup (automatically generated C source code).
- [Mahajan](#) compiler code generation backend
- [Schulte](#) bug fixing binary code
- Improving Java (C#?) byte code

Grow and Graft GP

GP working with human programmer.
Evolve “grow” code (eg bi-translation)
outside then use GP to incorporate “graft” it
into final host source code (pidgin).

Human identifies functionality, guides GP,
locates home in host code.

- [Babel pidgin](#) SSBSE-2014 winner
- [pknotsRG](#) 10000 speedup
- Django web server [citation server](#)

[Saturday 5 Sep 2015]

Extending Genetic Improvement

- Automatic mobile apt feature migration
- Configuring software product lines, composing web services
- Selecting components
- Many version “multiplicity” computing

Multi-objective Genetic Improvement

- Code can be optimised wrt multiple goals.
- Any goal. If it can be measured (potentially) it can be optimised.
- Non functional goals include
 - Accuracy, quality
 - Memory consumption
 - Speed
 - Energy consumption (extend battery life)
 - Bandwidth (particularly if costs user €)
 - Latency

Extending Genetic Improvement

- Plastic surgery importing external code.
Donor potentially multiple authors and sources
miniSAT, Kate call graphs [Saturday 5 Sep 2015]
- Deep parameter tuning.
 - Adding parameters to existing code so enabling external tuning of previously fixed/concealed functionality.

Tools to help GI

- Automatic test case generation
- Test case prioritisation directed to new or mutated code
- Co-evolution
- Software validation. “Proving” fixes& grafts
 - During search by using only “correct” equivalent transformations [Ryan](#), [Fatiregun](#)
 - After evolution: testing and model checking
- Automatic documentation and comments
- Refactoring, including parallel versions

Theory of Genetic Improvement

- Theoretical analysis
- Search spaces
 - Other (non-GP) types of search. GA, swarm, ant, tabu, ngram, EDA, hill-climbing
- Impact “grow” human choices in GGGP
- Impact of restricted (e.g. less than Turing complete) languages

Comments?

Comments?

What next?

Be ambitious: do something impossible

Bowtie2 Example

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-optimize
- 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
 - Keep only essential changes

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

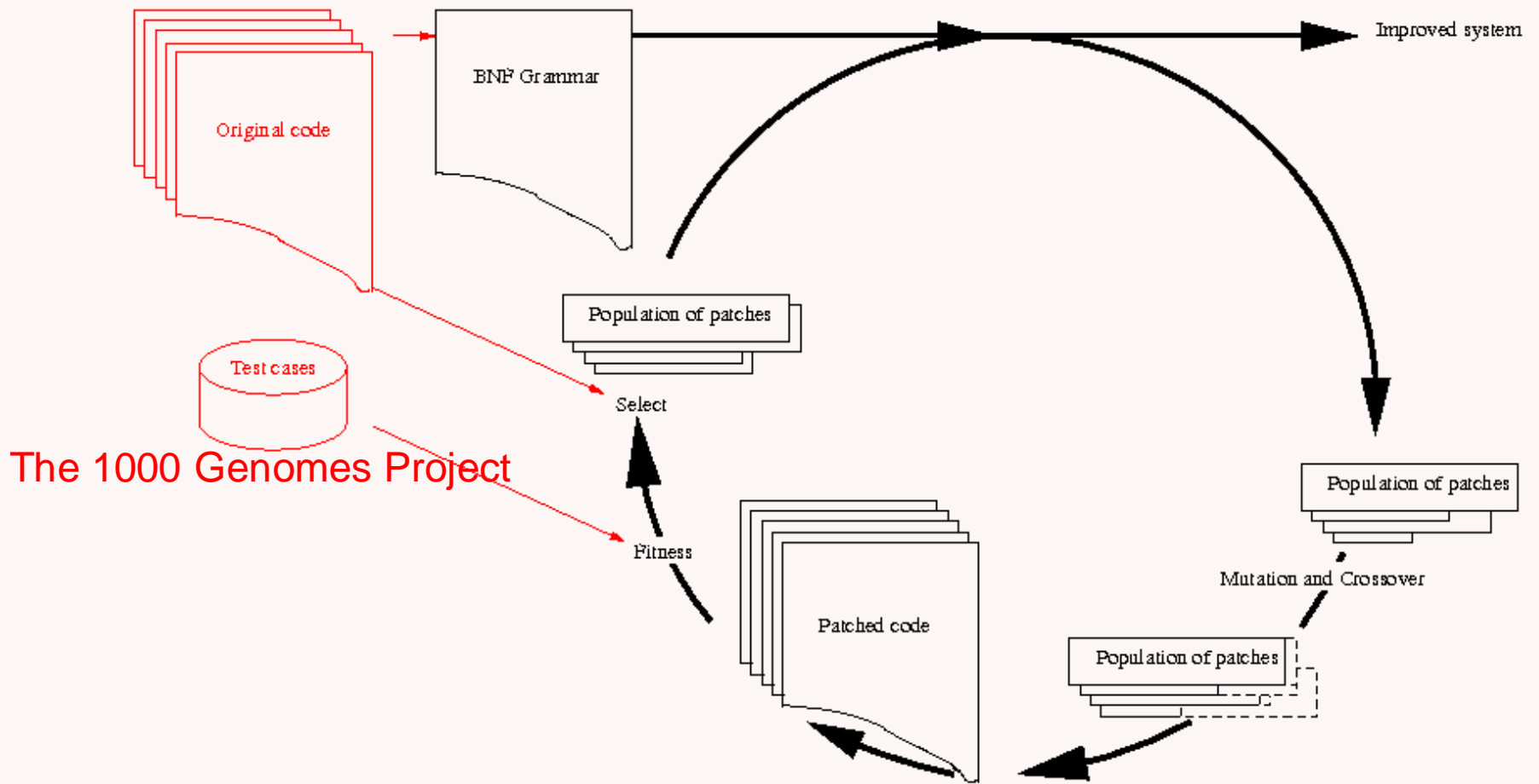
Why Bowtie 2 ?

- 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)
- Why 1000 genomes project? \$120million [180TBytes](#)
 - mapped all common human mutations
 - 604 billion 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



BNF Grammar

```
vmax = vlo;
```

Line 365 of aligner_swsse_ee_u8.cpp

```
<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)

```
<aligner_365> ::= "" <_aligner_365> "{Log_count64++;/*28577*/}\n"
<_aligner_365> ::= "vmax = vlo;"
```

- 2252 IF

```
<pe_118> ::=
    "{Log_count64++;/*20254*/} if" <IF_pe_118> " {\n"
<IF_pe_118> ::= "(!olap)"
```

7 Types of grammar rule (2)

- Type indicated by rule name. xxx is file name. _nn is line number in file.
- 272 for1, for2, 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*/}
```

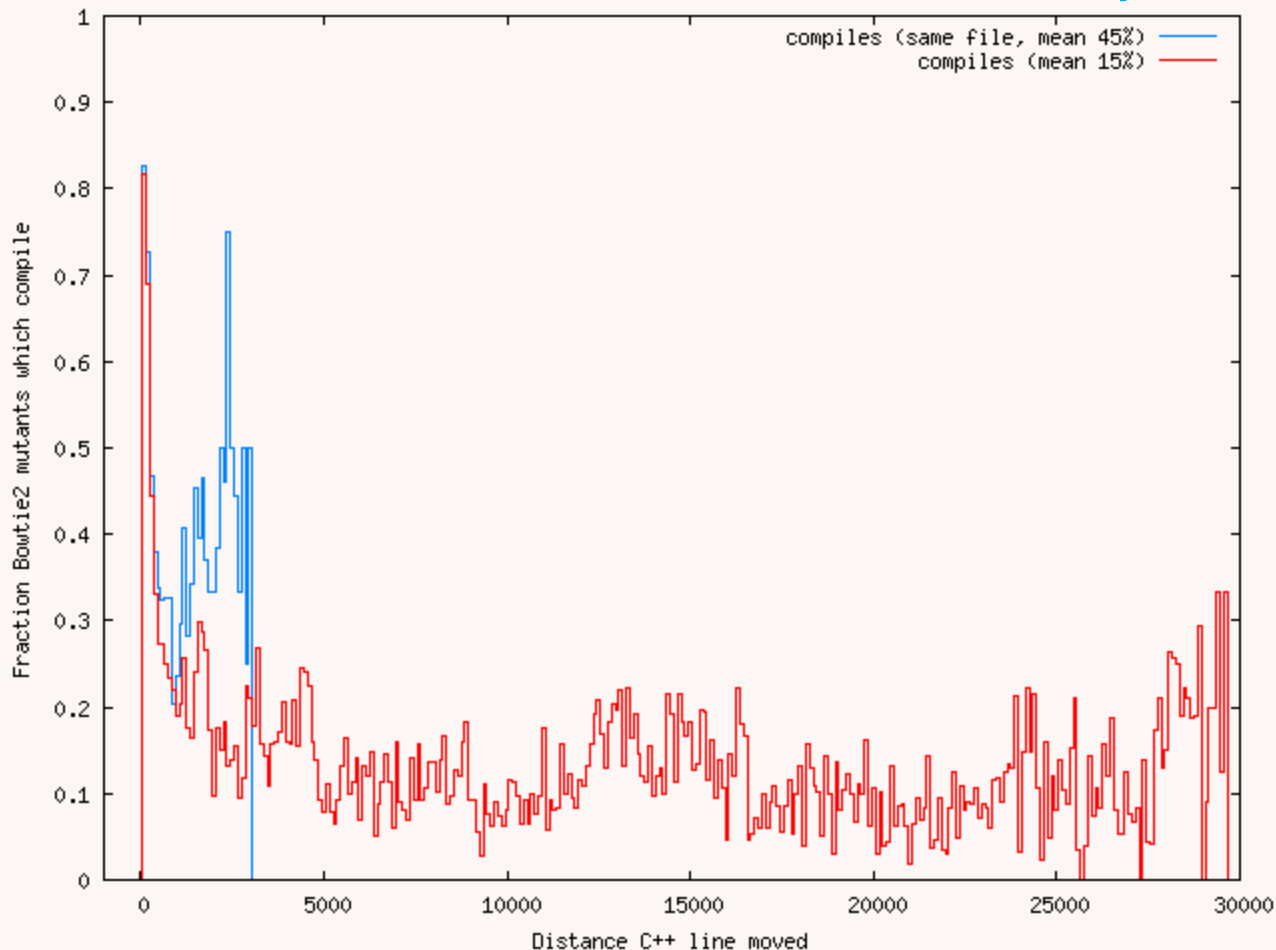
Instrumented original code

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

New code

Compilation Errors

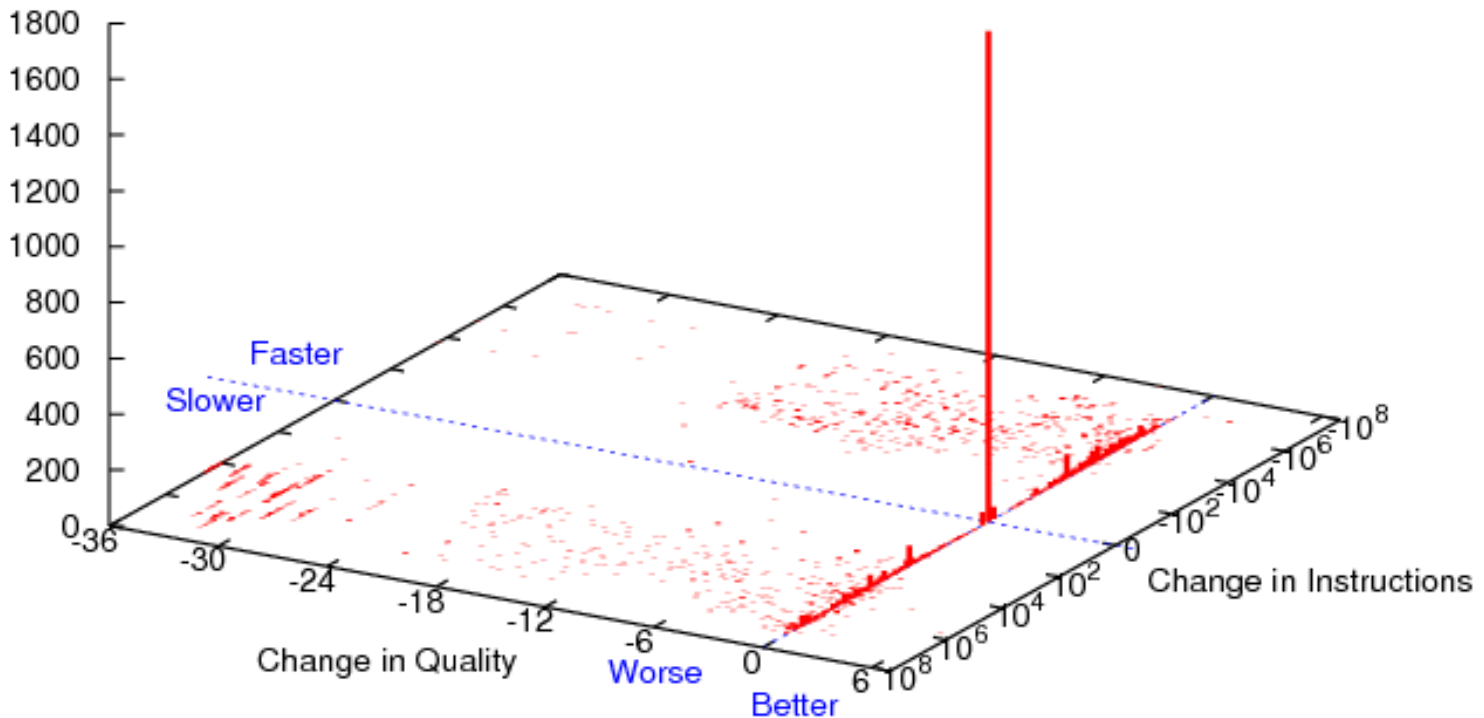
- Use grammar to replace random line, only **15%** compile. But if move <100 lines **82%** compile.
- Restrict moves to same file, **45% compile**



C++ is not fragile

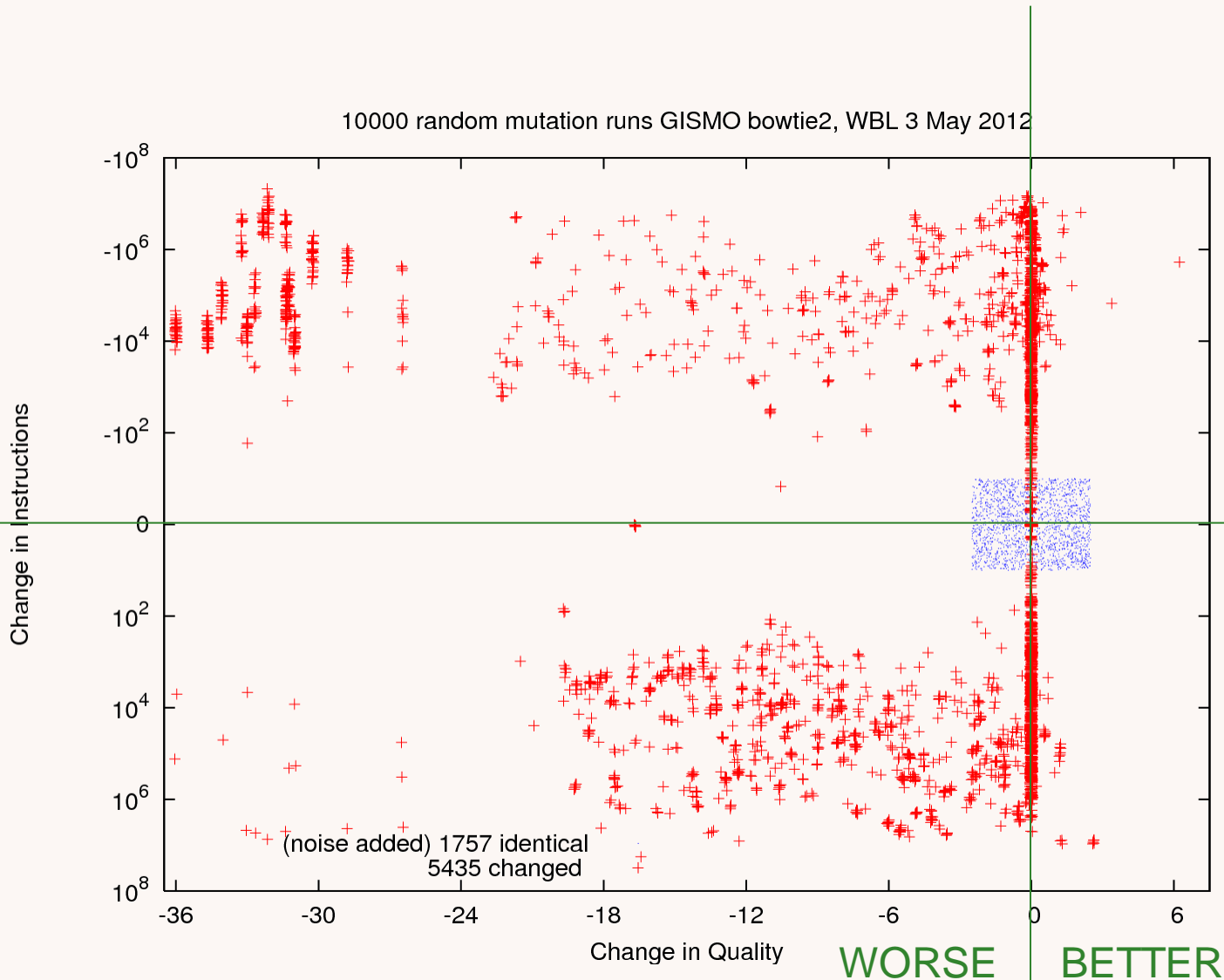
Trading performance v speed

10000 random mutation runs GISMO bowtie2, WBL 3 May 2012



C++ is not fragile

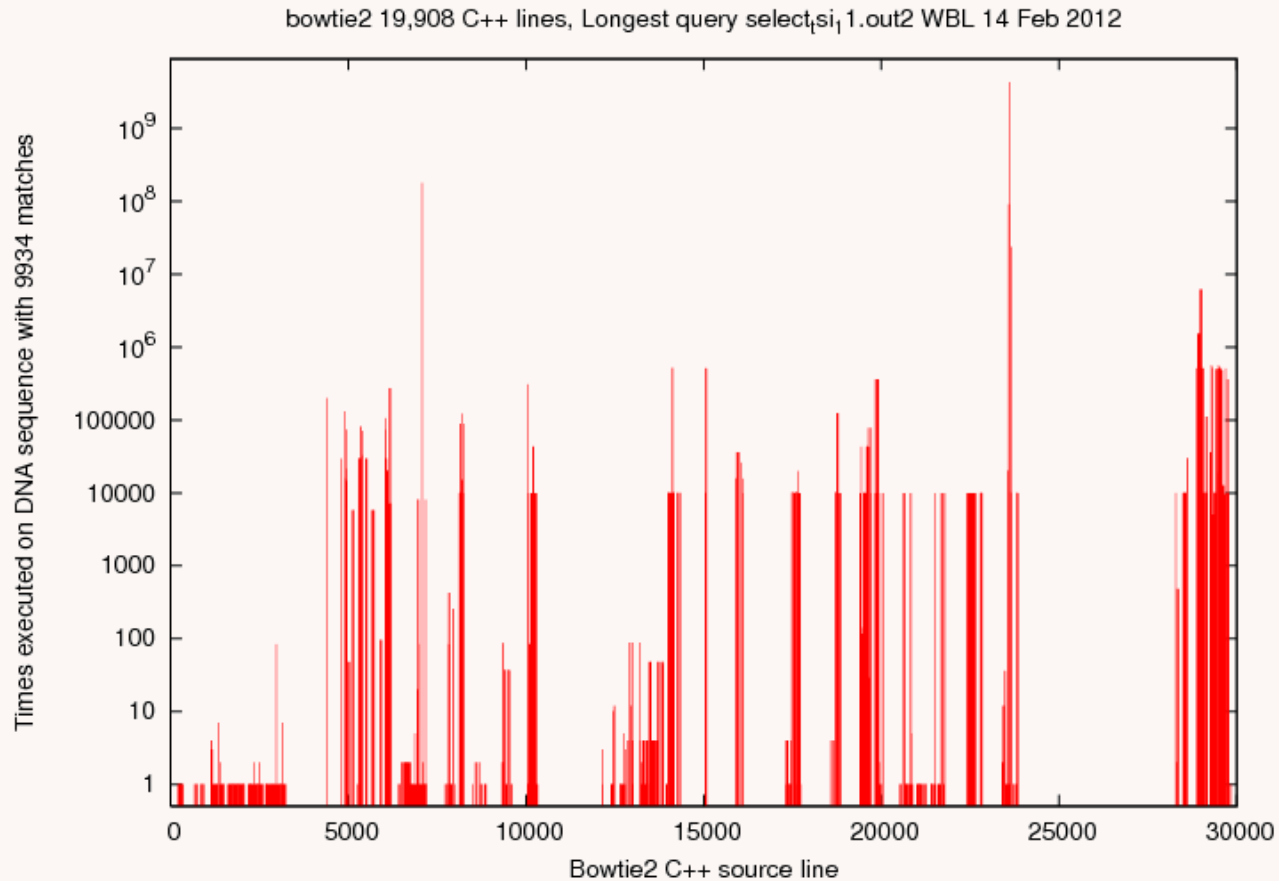
Trading performance v speed



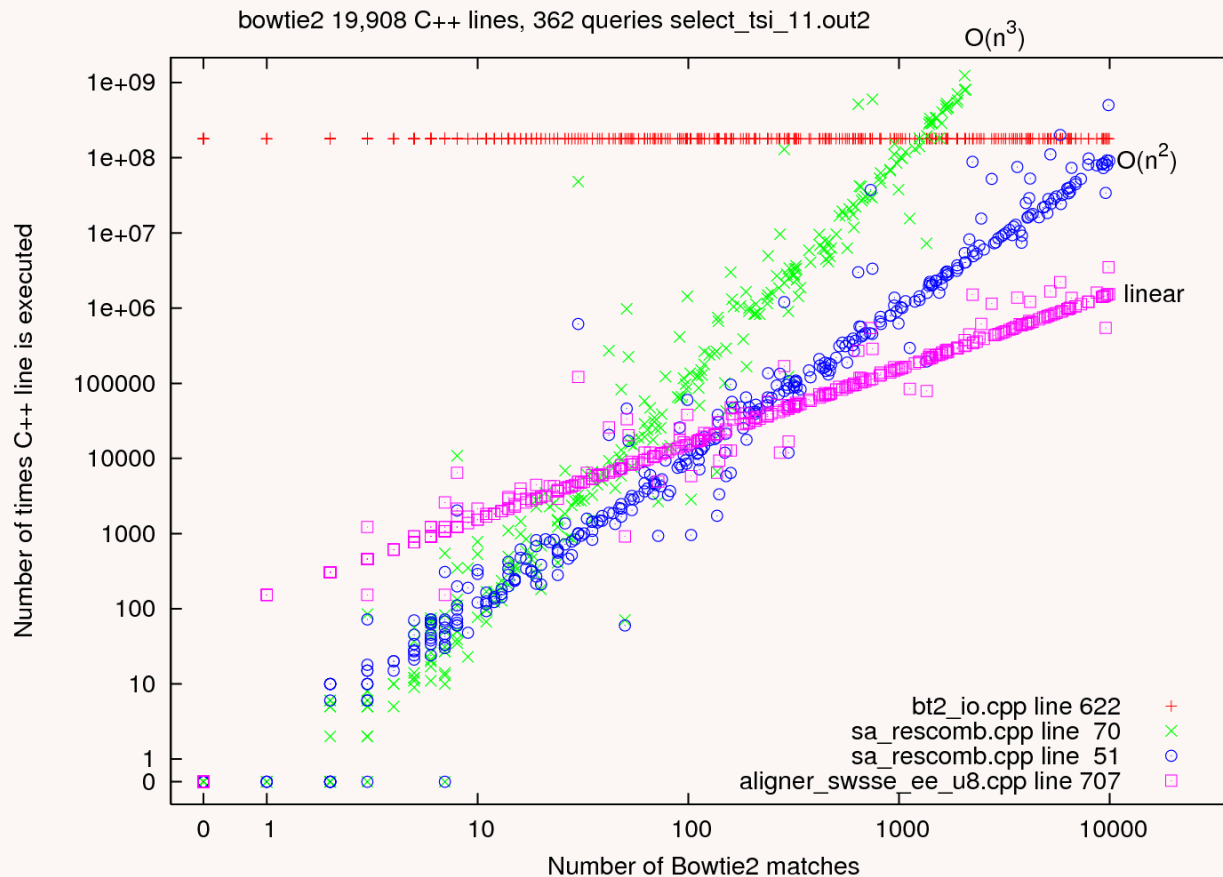
Recap

- 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



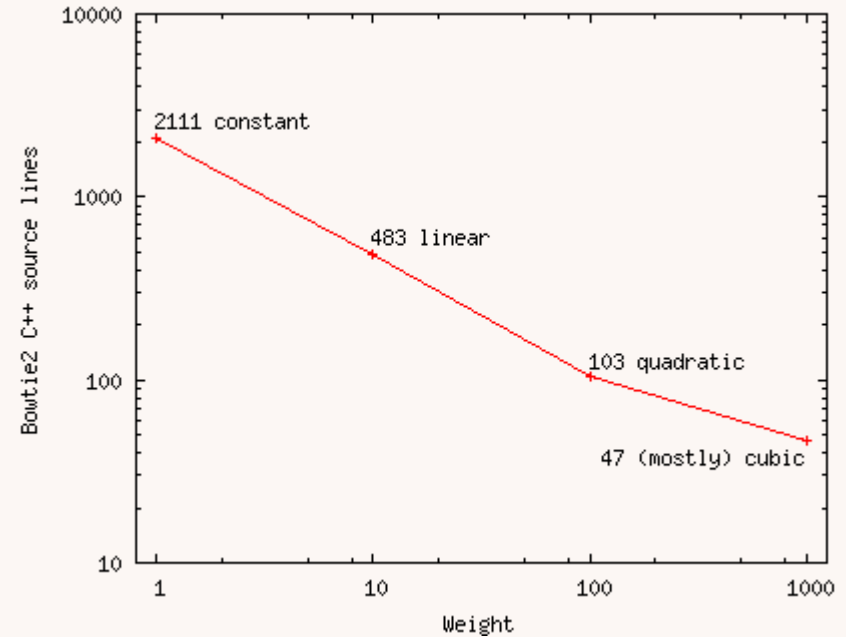
Scaling of Parts of Bowtie2



4 Heavily used Bowtie2 lines which scale differently

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 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
- \approx 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_sws_se_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 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

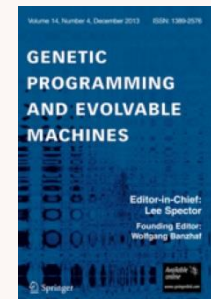
Conclusions

- Genetic programming can automatically create small programs
 - hash algorithms
 - random numbers which take less power, etc.
- “Fix” bugs ($>10^6$ lines of code, 16 programs)
 - auto-port (gzip to GPU). Merge programs (miniSAT [Humie](#))
 - new code to extend application (gggp babel pidgin)
 - speed up GPU image processing
- speed up 50000 lines of code
- **Software is not fragile**
 - break it, bend it, Evolve it

[GI special issue](#)

GP+EM

deadline 19 Dec



END

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

<http://www.epsrc.ac.uk/> 

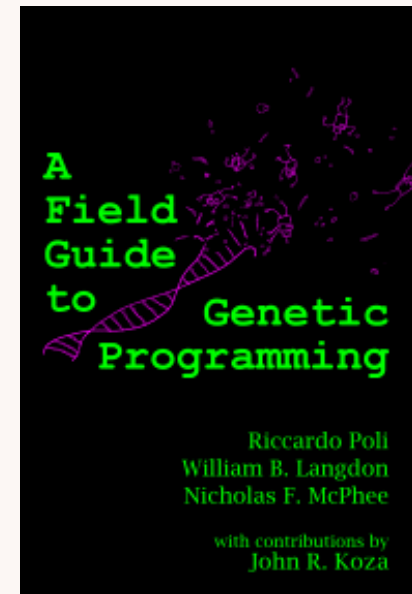
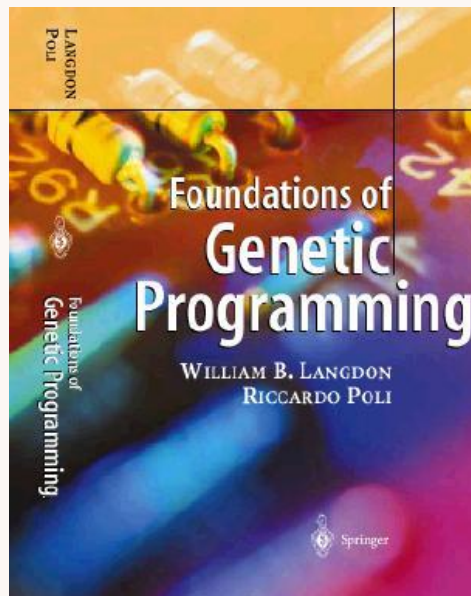
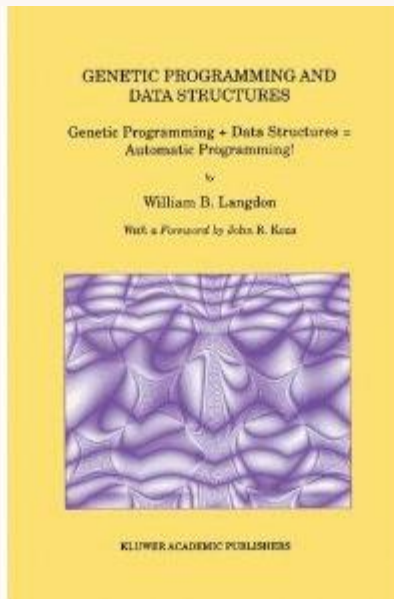
Genetic Improvement



W. B. Langdon

CREST

Department of Computer Science



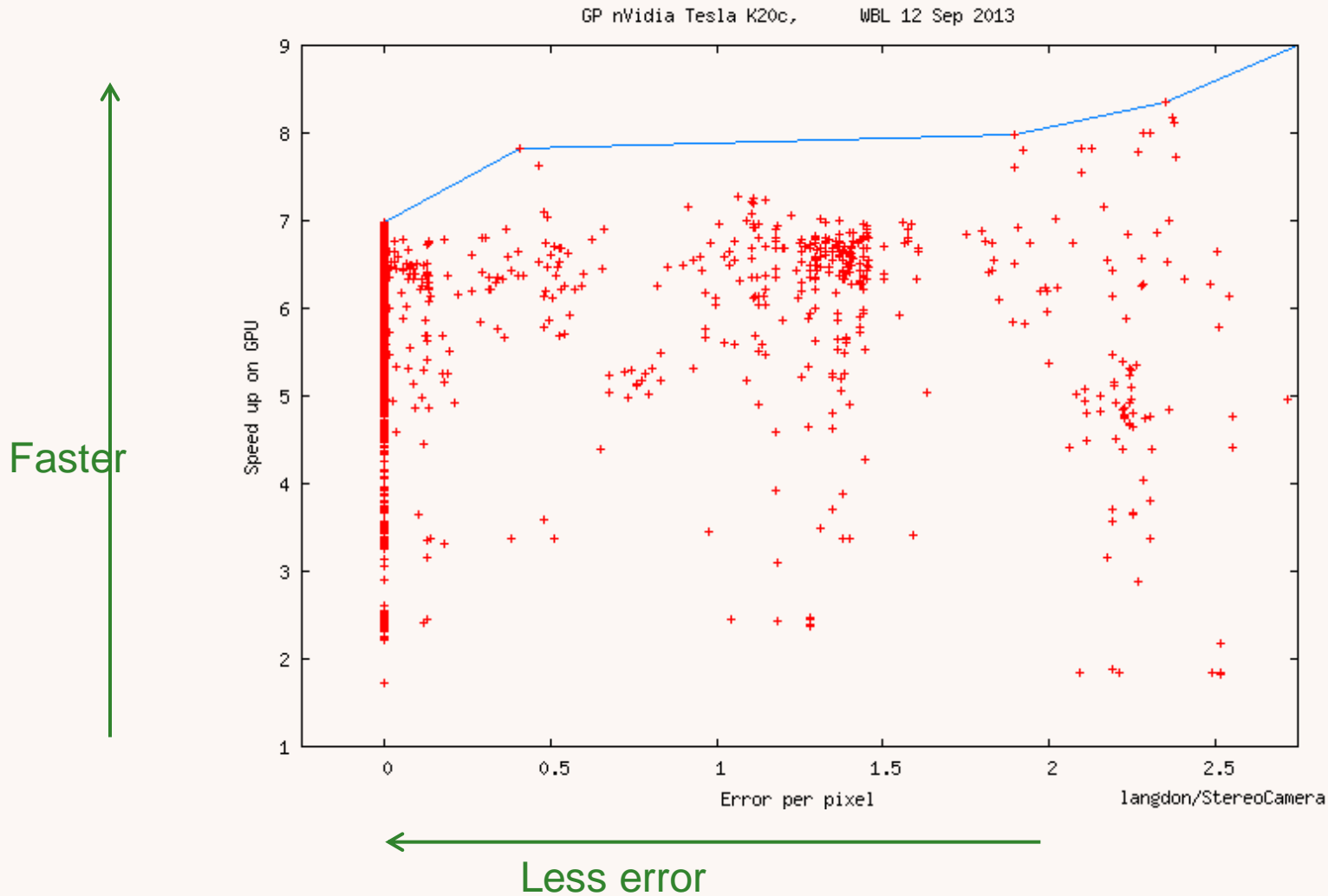
When to Automatically Improve Software

- When to use GP to create source code
 - Small. E.g. glue between systems “mashup”, Grow and Graft GP (GGGP): small additions to big systems
 - Multiple conflicting ill specified non-functional requirements
- Genetic programming as tool. GP tries many possible options. Leave software designer to choose between best.

Where next?

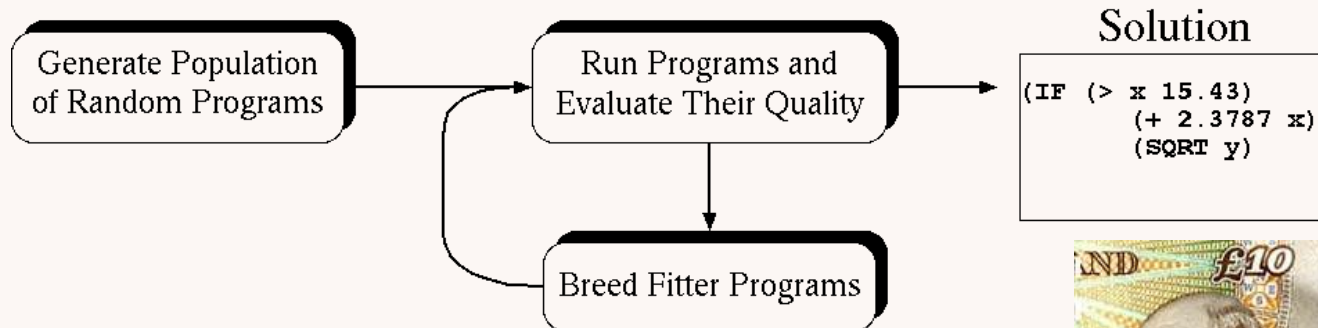
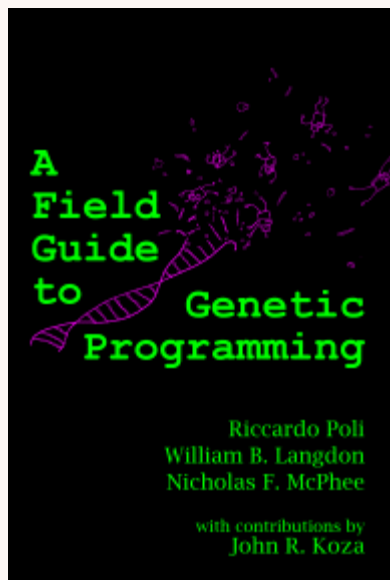
- Theory, analysis
- Self-healing
- Maintenance
- Mashups, SPL
- many versions, bespoke software
- Multiobjective GI
- Human + GP, GGGP
- GPU, mobile apt, embedded
- Testing, validation, coevolution

Tradeoff 2 objectives Pareto front



Genetic Programming

- Evolve a population of computer programs
 - Usually start with random programs, here use human code
 - Programs' 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.



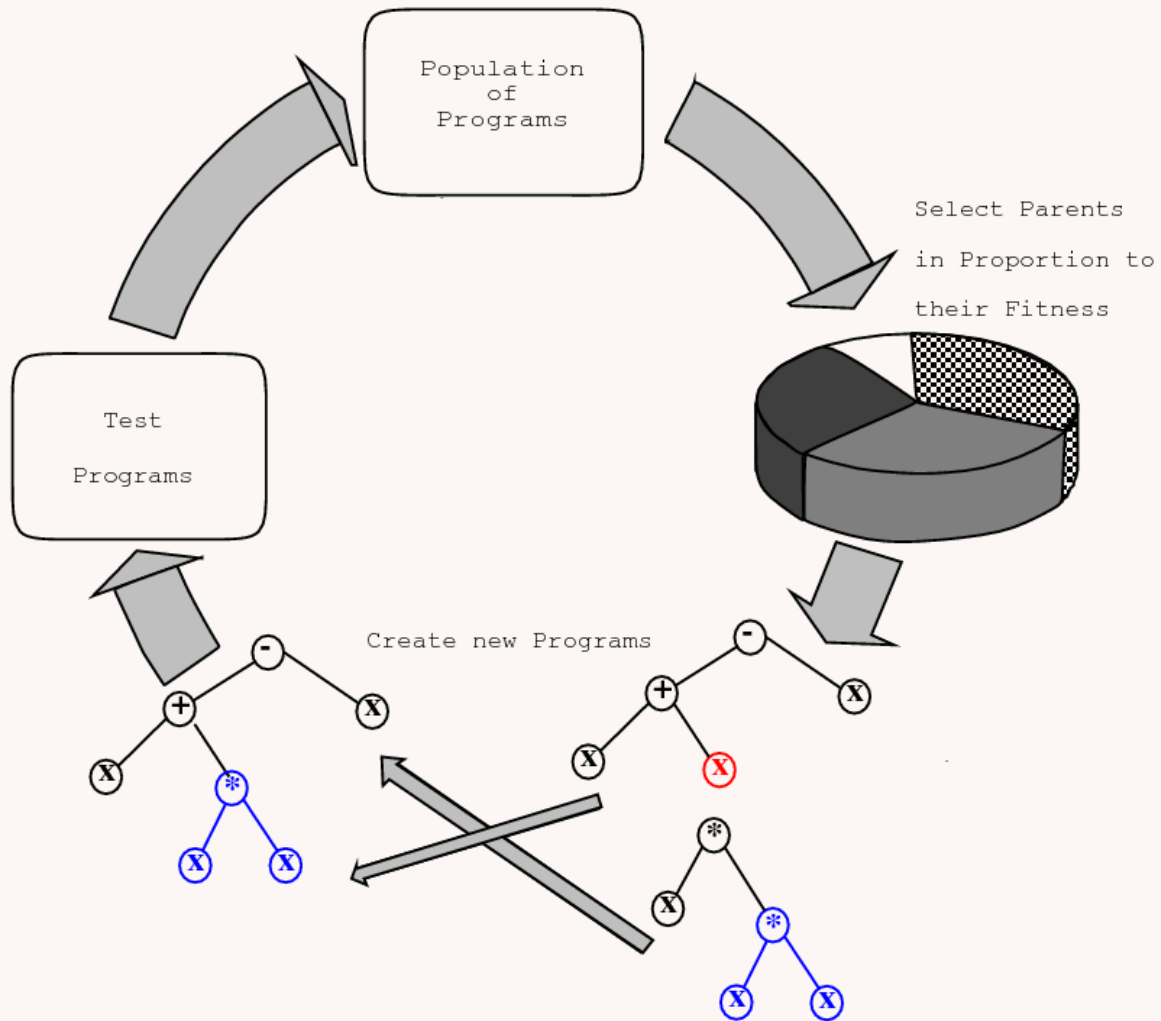
Solution
 (IF (> x 15.43)
 (+ 2.3787 x)
 (SQRT y))



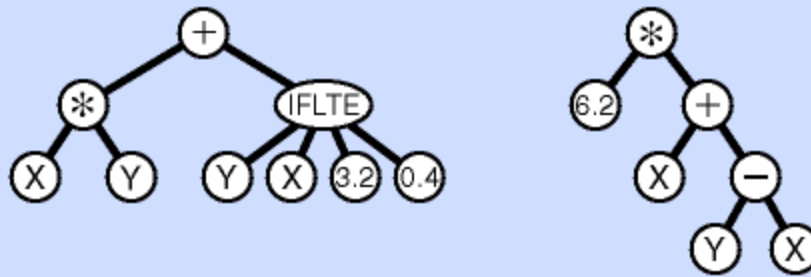
Charles Darwin 1809-1882

Free PDF Free E-book

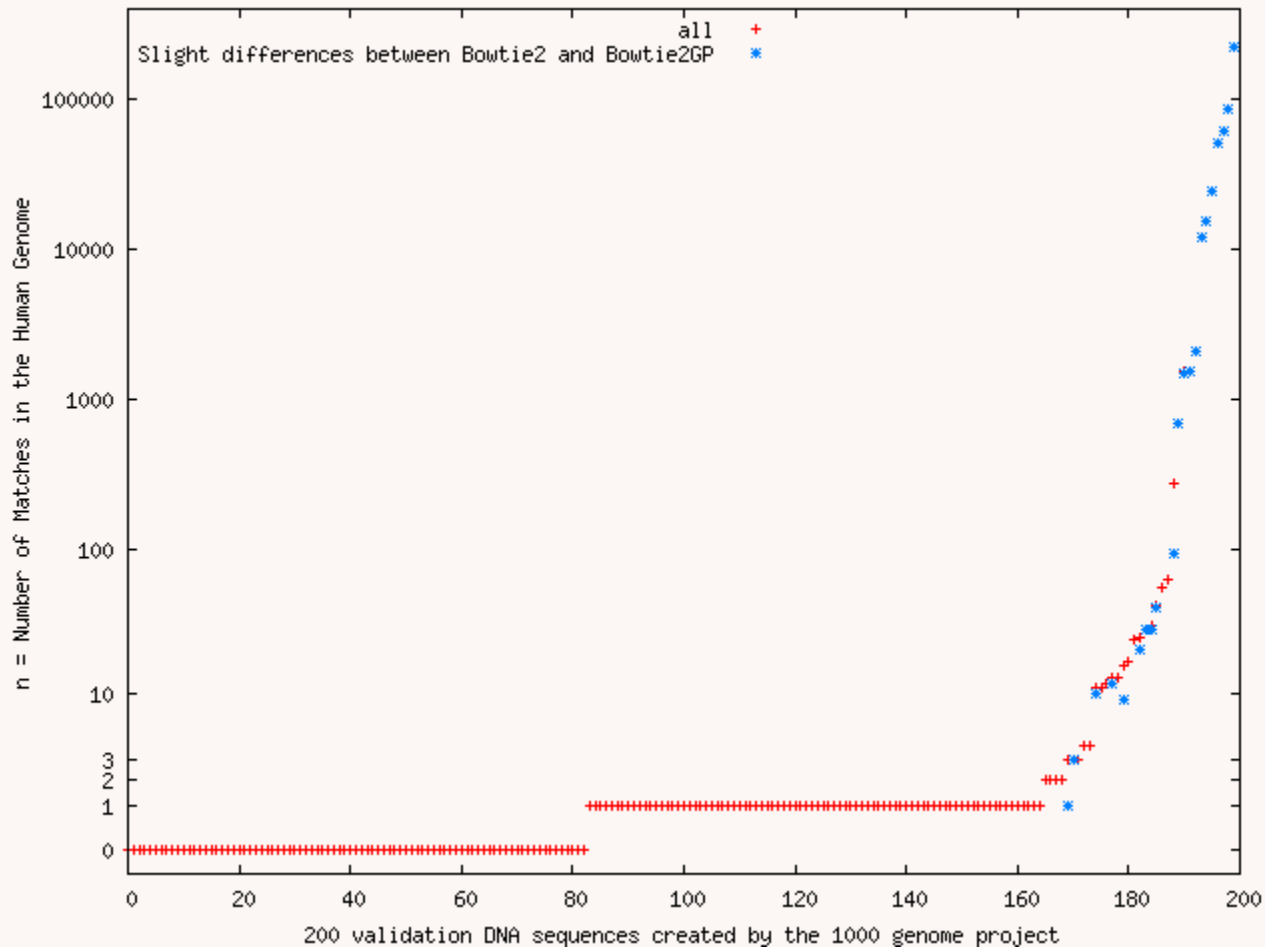
GP Generational Cycle



Creating new programs - Crossover

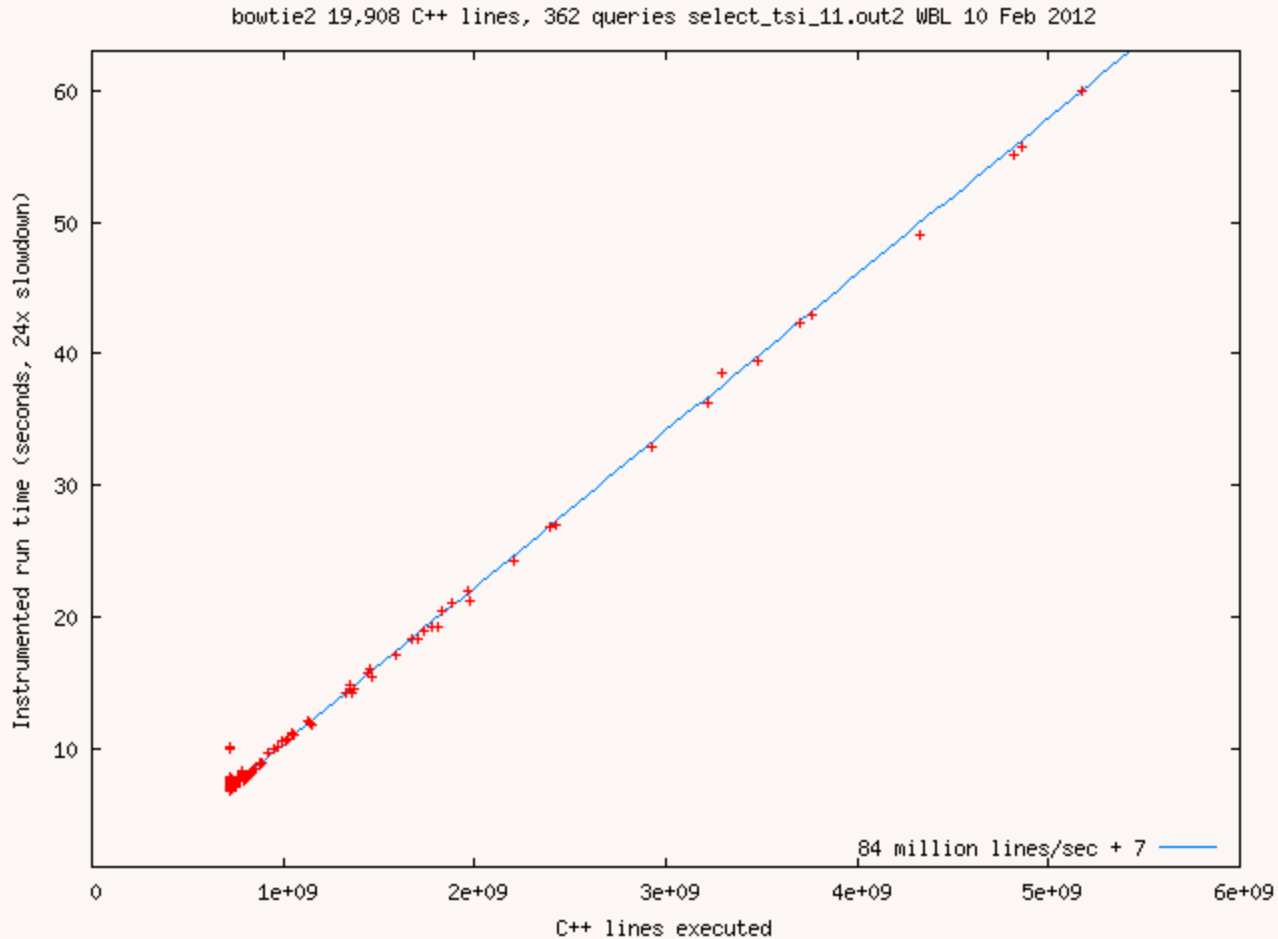


Where does Bowtie2^{GP} improvement arise



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

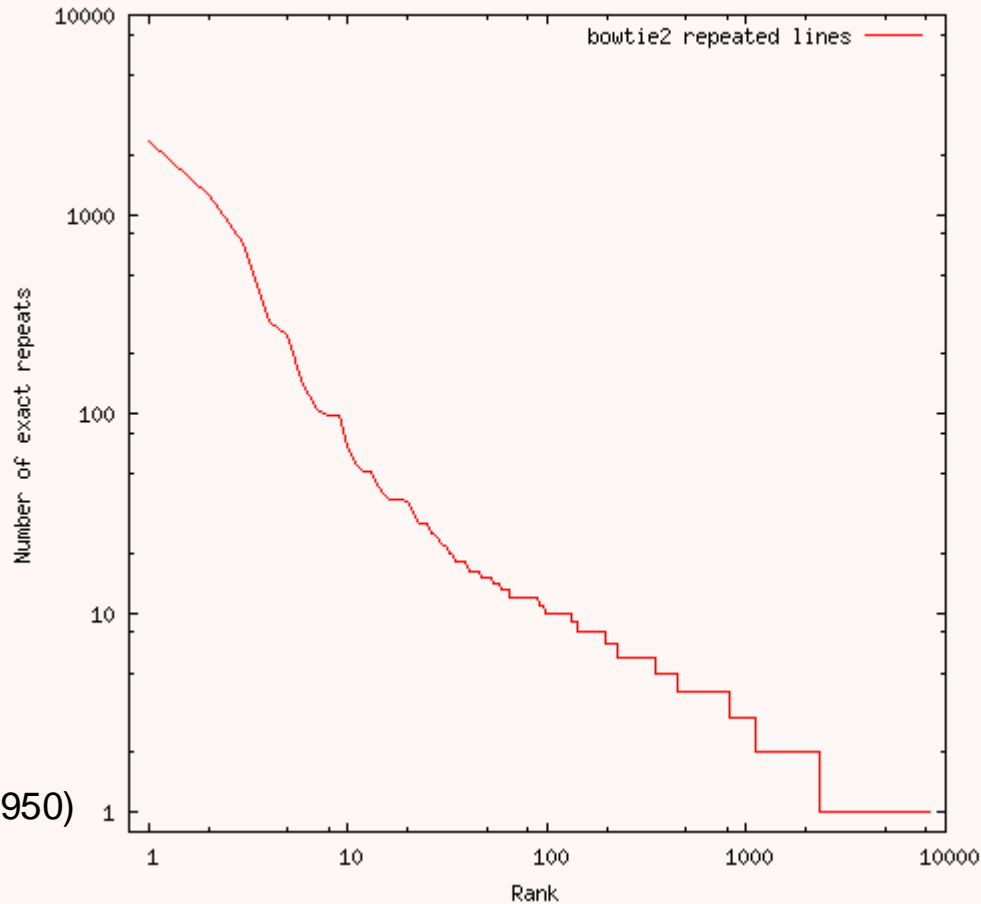
Instrumented Bowtie2



counter increments added to instrument Bowtie2

Zipf's Law

bowtie2 19,908 C++ lines, WBL 13 Jan 2012



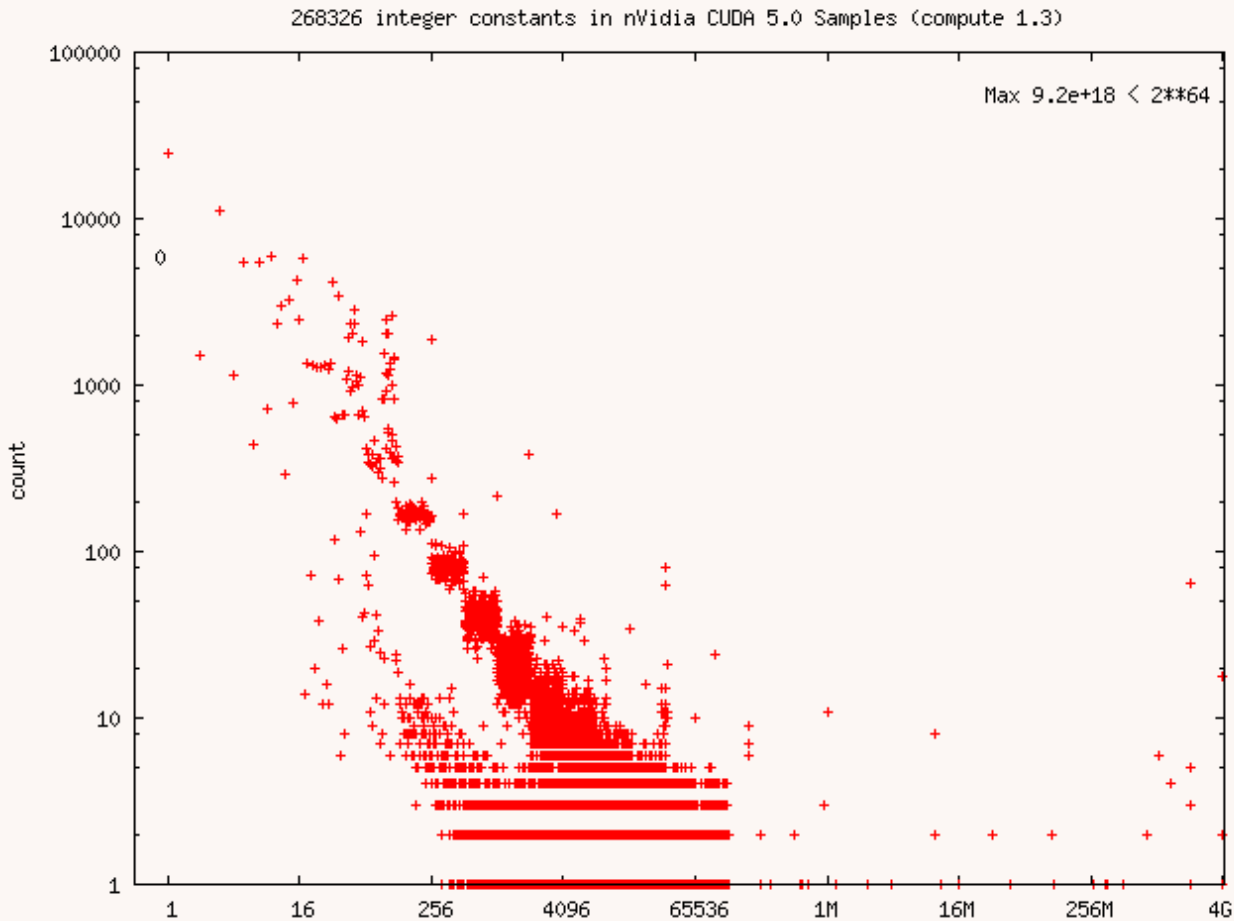
George Zipf (USA, 1902–1950)

$x = \log(\text{rank, order})$

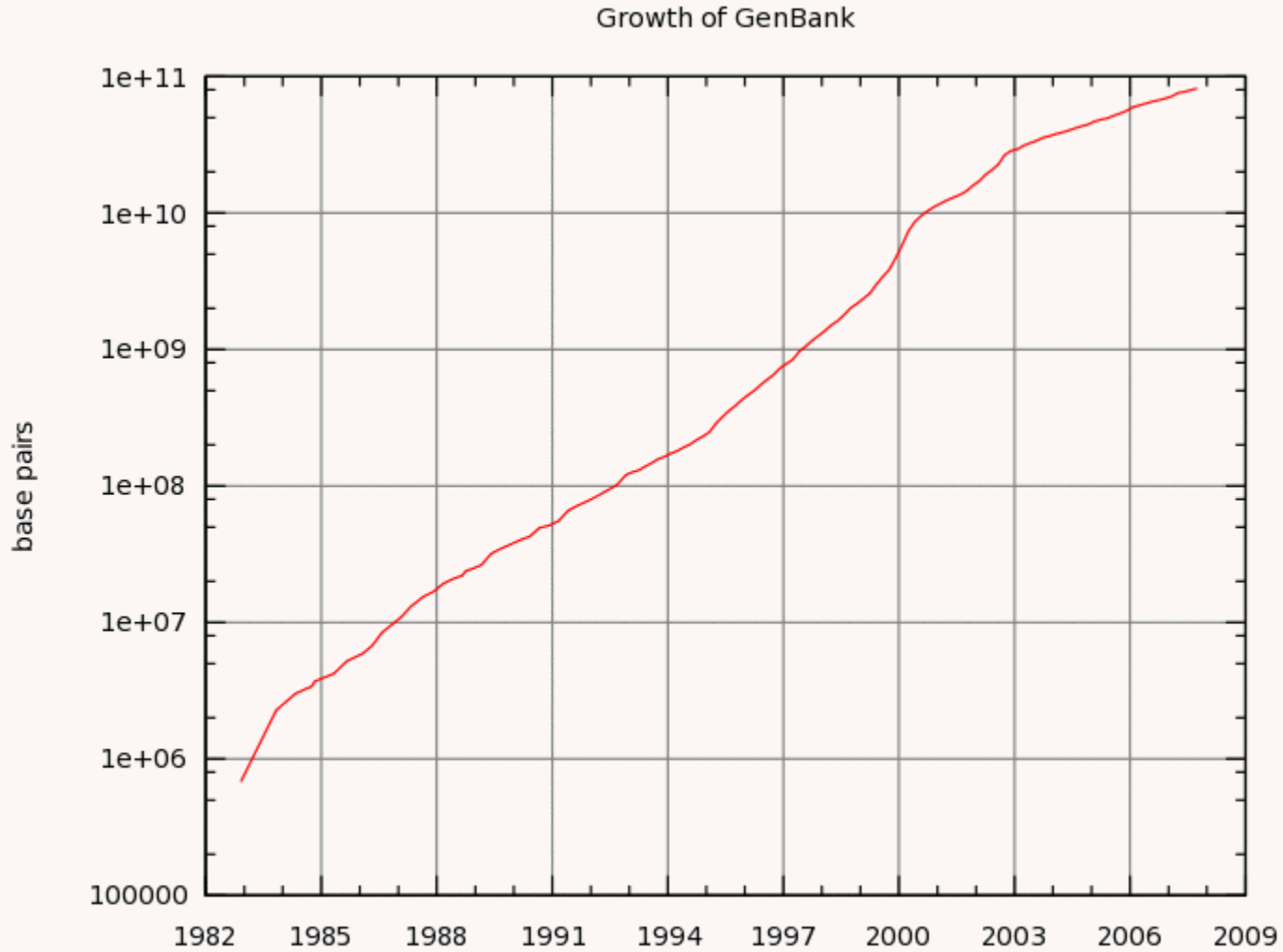
$y = \log(\text{frequency})$.

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



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.


Do Something Impossible

- White Queen claimed to think 6 impossible things before breakfast
- Which impossible thing will you *do*?

The Genetic Programming Bibliography

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

10415 references

RSS Support available through the
Collection of CS Bibliographies. 



Part of gp-bibliography 04-40 Revision: 1.794 29 May 2011



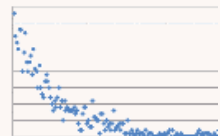
A web form for adding your entries.
Co-authorship community. Downloads

Downloads



A personalised list of every author's
GP publications.

[blog.html](#)



Search the GP Bibliography at

<http://iinwww.ira.uka.de/bibliography/Ai/genetic.programming.html>