

# Correctness Attraction:

## A Study of Stability of Software Behavior Under Runtime Perturbation

Benjamin Danglot, Philippe Preux, Benoit Baudry, Martin Monperrus,  
Empirical Software Engineering, August 2018, 23(4) pp 2086-2119

[doi:10.1007/s10664-017-9571-8](https://doi.org/10.1007/s10664-017-9571-8)



Started 1996

Impact Factor 2.933

Editor-in-Chief: R. Feldt &  
T. Zimmermann

Springer

Editorial Board includes:  
Massimiliano Di Penta, Mark  
Harman, Miryung Kim, Tim  
Menzies, [Martin Monperrus](#),  
Federica Sarro, Martin  
Shepperd, Paolo Tonella,  
Shin Yoo, Andreas Zeller

[W. B. Langdon](#)

Slides for Software Systems Engineering SSE [Reading Group](#), 19 September 2018





# Introduction

Prof. Martin Monperrus presented  
“Correctness Attraction...” at ICSE-2018  
as a “Journal first paper”

# Correctness Attraction

- What does the title mean
- Why this paper
- Who are these guys
- Why is correctness attraction important
- What should we do
  - Stop teaching perfect programming
  - Continue to show software is not fragile
  - Evolve increased levels of robustness
- Conclusions

# Authors

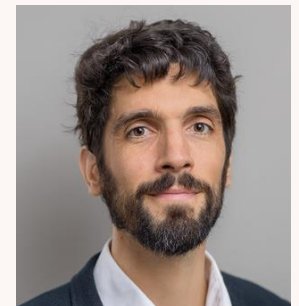
- Benjamin Danglot 
  - PhD student, INRIA
- Philippe Preux
  - Prof. University of [Lille](#) 3, INRIA
  - adaptive systems, machine learning
- Benoit Baudry 
  - Prof KTH Stockholm
    - software diversification
- Martin Monperrus
  - Prof KTH Stockholm
    - How to automatically repair software?
    - How to construct self-healing software?



[H4](#)



[H42](#)



[H26](#)

# What does title mean?

Correctness Attraction:

A Study of Stability of Software Behavior Under Runtime Perturbation

- **Correctness attraction**
  - output is not changed by perturbation during test execution (NB using perfect test oracle)
- **A Study of**
  - Experiments
- **Stability of Software Behavior**
  - Tendency of program output to be unchanged
- **Runtime Perturbation**
  - change an expression's value as program is run

# Why this paper

- Tie in with UCL Software Systems Engineering
  - Genetic Improvement
  - “Software is not fragile”
  - errors failing to escape code
  - equivalent mutants in mutation testing
- Over turn fear of automated software modification
- If software is naturally somewhat correct, can we evolve it so as to increase its tendency to be correct?

# What did they do

- Java source code consider either int or boolean expressions
- Wrap all target expressions in perturb function:
  - $p(\text{index}, \text{input})$
  - usually  $p()$  returns input (no semantic change)
  - exactly once one  $p()$  makes one change
- First run normally for each test case to find how many times each  $p()$  is called
- Exhaustively try all (hence small examples)

# What did they do

- 10 small (42 to 568 lines) diverse Java: quicksort, zip, sudoku, md5, rsa, rc4, canny, lcs, laguerre, linreg
- PONE add one, MONE -1, PZERO return 0, PBOOL flip boolean
- PONE approx three million experiments.  $\approx$ two million (68%) pass perfect oracle
- Some parts of programs more stable
- Some types of perturbation (e.g. boolean) more disruptive than others, c.f. Yue Jia's [equivalent mutations](#)



# PONE on quicksort

- 41 int expressions
- Most PONE perturbation sites always or nearly always still pass all tests
- 4 locations where adding one almost always causes quicksort to fail

**Table 5** The breakdown of correctness attraction per perturbation point in Quicksort for integer point

IndexLoc	#Perturb. Execs	$\phi$ : #Success	$\chi$ : #Failure	$\xi$ : #Exception	$\Phi$ : Correctness ratio
0	1751	1202	543	6	68%
1	1751	1641	0	110	93%
2	1751	1751	0	0	100%
3	1751	1751	0	0	100%
4	1751	1751	0	0	100%
5	1751	1751	0	0	100%
6	1751	1751	0	0	100%
7	1751	1751	0	0	100%
8	1751	1751	0	0	100%
9	1751	1739	0	12	99%
10	5938	5938	0	0	100%
11	5938	5938	0	0	100%
12	8459	5946	2496	17	70%
13	8459	8459	0	0	100%
14	8459	8442	0	17	99%
15	4272	4272	0	0	100%
16	9495	6676	2691	128	70%
17	9495	9477	0	18	99%
18	9495	9495	0	0	100%
19	5308	5308	0	0	100%
20	4187	4187	0	0	100%
21	4187	3616	571	0	86%
22	3616	105	3506	5	2%
23	3616	0	3564	52	0%
24	3616	3616	0	0	100%
25	3616	3616	0	0	100%
26	1751	1633	118	0	93%
27	1751	1751	0	0	100%
28	840	275	565	0	32%
29	840	840	0	0	100%
30	1751	1751	0	0	100%
31	1751	1632	119	0	93%
32	891	321	570	0	36%
33	891	801	0	90	89%
34	3616	2361	1250	5	65%
35	3616	0	3616	0	0%
36	3616	1515	2101	0	41%
37	3616	742	2822	52	20%
38	3616	553	3058	5	15%
39	3616	1420	2149	47	39%
40	3616	0	3616	0	0%

**Table 5** The breakdown of correctness attraction per perturbation point in Quicksort for integer point

IndexLoc	#Perturb. Execs	$\phi$ : #Success	$\chi$ : #Failure	$\xi$ : #Exception	$\Phi$ : Correctness ratio
0	1751	1202	543	6	68%
1	1751	1641	0	110	93%
2	1751	1751	0	0	100%
3	1751	1751	0	0	100%
4	1751	1751	0	0	100%
5	1751	1751	0	0	100%
6	1751	1751	0	0	100%
7	1751	1751	0	0	100%
8	1751	1751	0	0	100%
9	1751	1739	0	12	99%
10	5938	5938	0	0	100%
11	5938	5938	0	0	100%
12	8459	5946	2496	17	70%
13	8459	8459	0	0	100%
14	8459	8442	0	17	99%
15	4272	4272	0	0	100%
16	9495	6676	2691	128	70%
17	9495	9477	0	18	99%
18	9495	9495	0	0	100%
19	5308	5308	0	0	100%
20	4187	4187	0	0	100%
21	4187	3616	571	0	86%
22	3616	105	3506	5	2%
23	3616	0	3564	52	0%
24	3616	3616	0	0	100%
25	3616	3616	0	0	100%
26	1751	1633	118	0	93%
27	1751	1751	0	0	100%
28	840	275	565	0	32%
29	840	840	0	0	100%
30	1751	1751	0	0	100%
31	1751	1632	119	0	93%
32	891	321	570	0	36%
33	891	801	0	90	89%
34	3616	2361	1250	5	65%
35	3616	0	3616	0	0%
36	3616	1515	2101	0	41%
37	3616	742	2822	52	20%
38	3616	553	3058	5	15%
39	3616	1420	2149	47	39%
40	3616	0	3616	0	0%

# PONE on ten programs

Results like quicksort but variation between programs

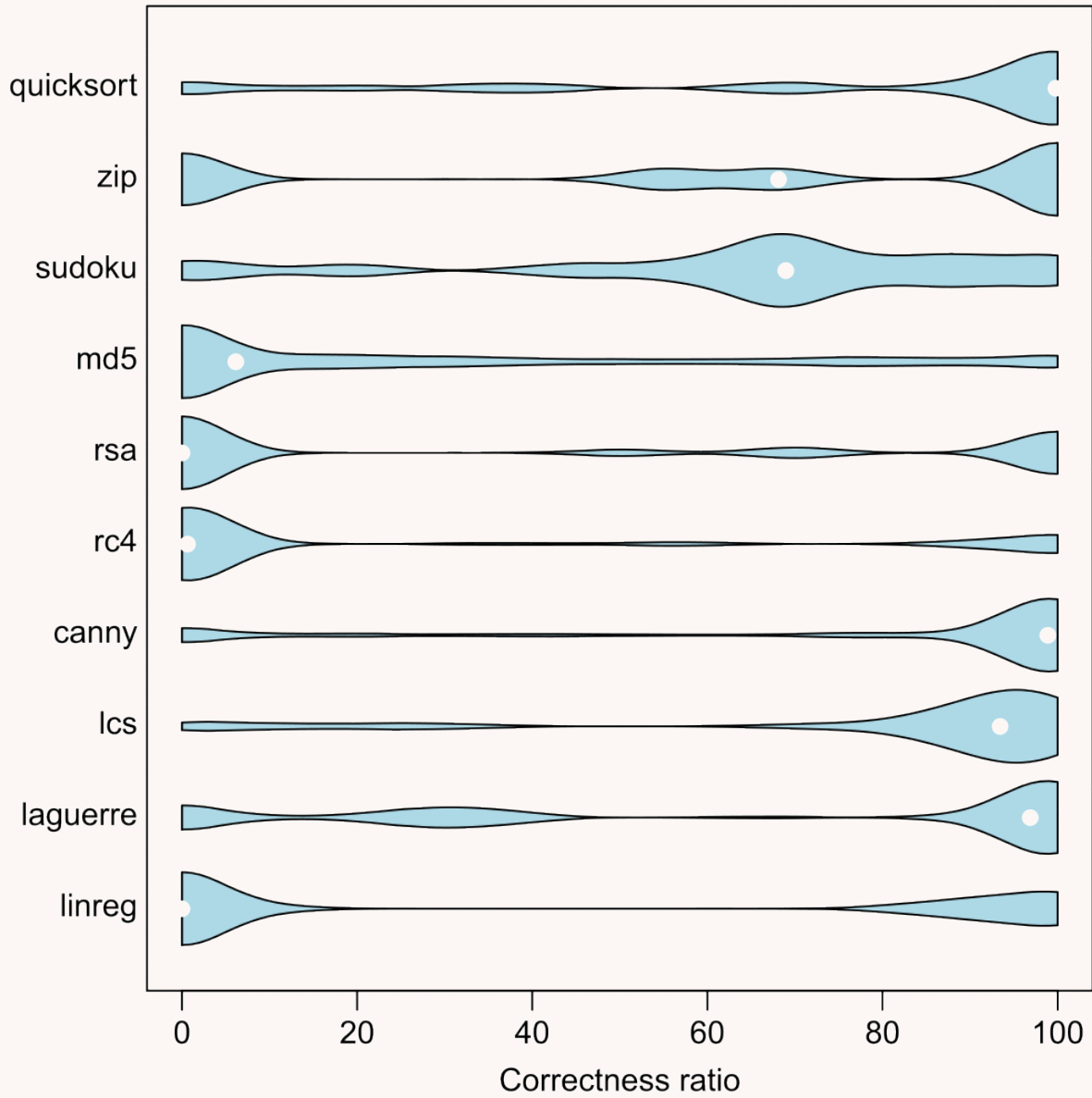
Empir Software Eng (2018) 23:2086–2119

2097

**Table 6** PONE Results

Subject	$N_{pp}^{int}$	—Search space—	# Fragile exp.	#Robust exp.	# Antifragile exp.	$\Phi$ : Correctness ratio
quicksort	41	151444	6	10	19	————— 77%
zip	19	38840	5	2	5	————— 76%
sudoku	89	98211	12	27	8	————— 68%
md5	164	237680	102	24	7	— 29%
rsa	117	2576	55	8	32	————— 54%
rc4	115	165140	60	7	12	————— 38%
canny	450	616161	58	129	133	————— 94%
lcs	79	231786	10	47	13	————— 89%
laguerre	72	423454	15	24	15	————— 90%
linreg	75	543720	43	18	11	————— 47%
Total	1221	2509012	366	296	255	————— 66%

CREST  
PONE perturbability profiles ten Java programs



# Summary: PONE +1 once per test

- The ten Java programs can have an error injected without always failing
- There are a small number of fragile int expressions.
- Most int expressions can be perturbed and yet the program remains correct
- “Dijkstra’s view that software is fragile is not always true, correctness is rather a stable equilibrium than an unstable one.”  
page 2098

# Minus one

Effect of subtracting one from exactly one int expression once when Java program is run very similar to effect of adding one.

**MONE  $\approx$  PONE**

# Setting to zero

Effect of replacing value of one int expression exactly once at runtime is similar to effect of adding or subtracting one, but slightly more disruptive.

“we observe 63% of correctness attraction with PZERO and 77% with PONE.”



# Flipping boolean

- Flipping one boolean expression value exactly once at runtime tends to be more disruptive than a similar runtime change to int expressions in Java programs.
- Nonetheless many PBOOL changes give right answer across the whole test suite.
- Mean correctness ratio across ten Java programs PBOOL 37% (PONE 66%).
- Some Java programs where PBOOL is usually disruptive

# PBOOL on quicksort

**Table 7** The breakdown of correctness attraction per boolean perturbation point in Quicksort

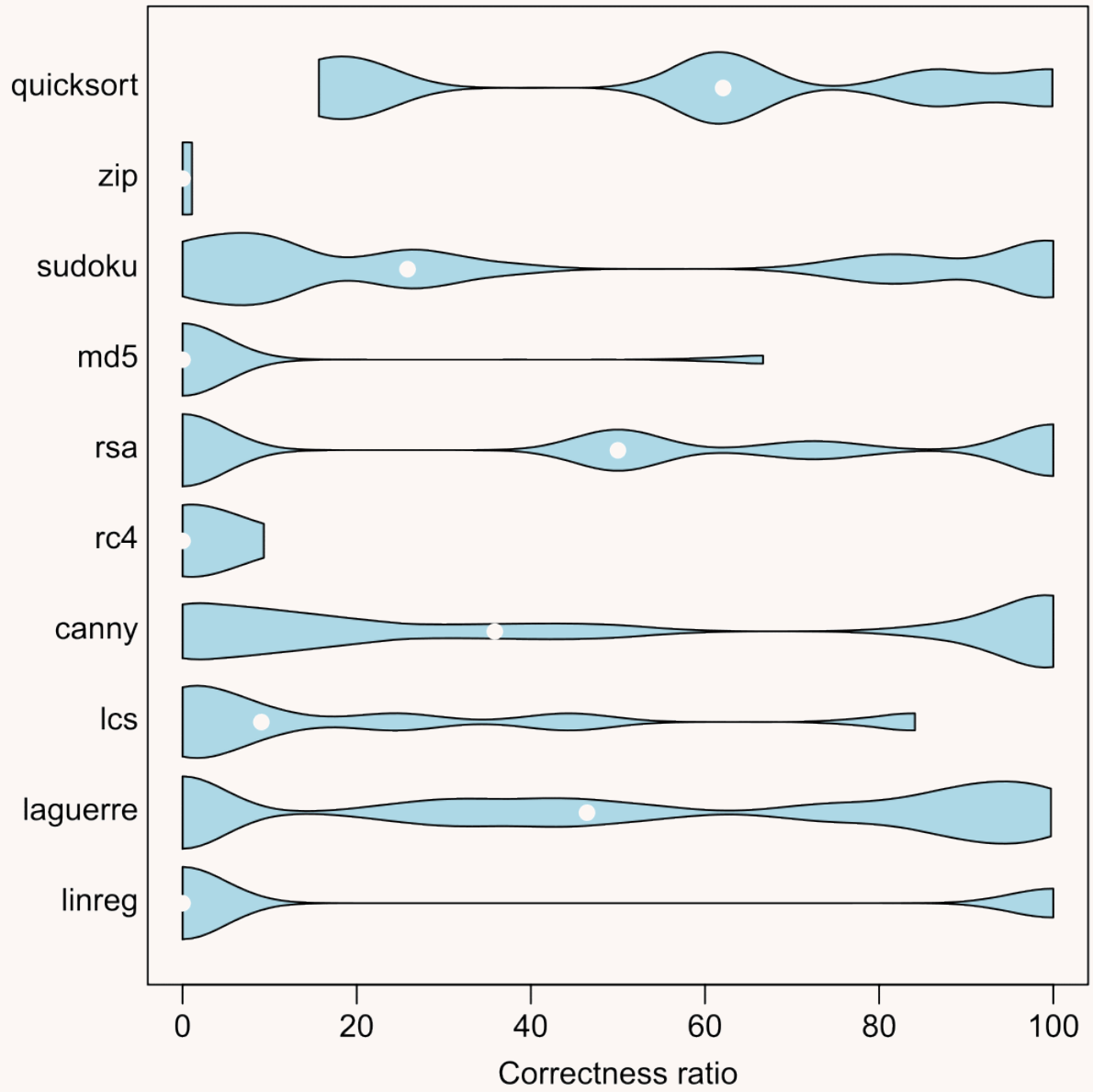
IndexLoc	#Perturb. Execs	$\phi$ : #Success	$\chi$ : #Failure	$\xi$ : #Exception	$\Phi$ : Correctness ratio
0	5938	5932	0	6	————— 99%
1	8459	1779	6663	17	— 21%
2	9495	1486	7991	18	- 15%
3	4187	3616	571	0	————— 86%
4	1751	1087	664	0	————— 62%
5	1751	1072	679	0	————— 61%

# PBOOL on ten Java programs

**Table 8** The Results of the PBOOL Experiment

Subject	$N_{pp}^{bool}$	—Search space—	# Fragile exp.	#Robust exp.	# Antifragile exp.	$\Phi$ : Correctness ratio
quicksort	6	31581	2	2	-	——— 47.41%
zip	6	14280	5	-	-	0.78%
sudoku	26	28908	14	9	1	——— 52.8%
md5	10	12580	9	-	-	0.95%
rsa	20	620	7	2	5	——— 49.68%
rc4	7	10540	7	-	-	7.59%
canny	79	77038	28	10	14	————— 71.55%
lcs	9	25165	6	1	-	——— 55.13%
laguerre	25	109837	8	11	-	————— 83.81%
linreg	15	98140	10	-	4	0.04%
total	203	408689	96	35	24	——— 36.974%

# The distribution of PBOOL Perturbability



# Critique

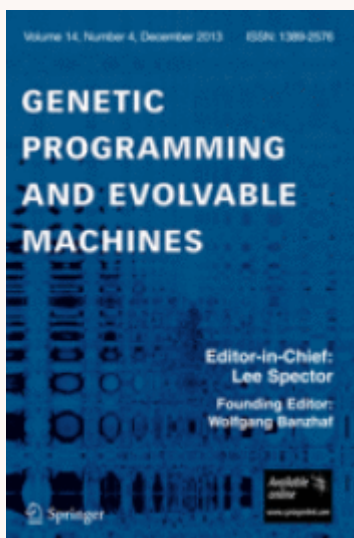
- Artefact of chosen programs?
  - May be, but using non-trivial examples.
  - Perhaps bigger programs are more robust?
- Artefact of chosen test suites?
  - Seem to have done good job
  - Better than usual industrial practice?
- Artefact of Java?
  - Seems unlikely. Reasons given (Section 5) would apply in other programming languages

# Conclusions

- Why
  - programs lose information as they execute.
    - e.g. calculate mean. Programs cannot be reversed. Many paths lead to same output.
  - Tendency to correctness is tendency to produce the same output as the unperturbed “correct” execution.
  - run time perturbations behaving like source code perturbations, c.f. genetic improvement
- What should we do
  - Continue to show software is not fragile
  - **Evolve increased levels of robustness**



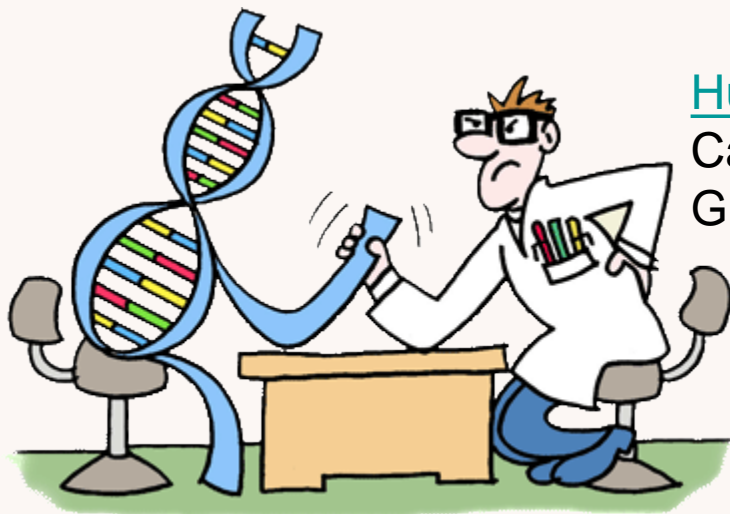
WIKIPEDIA  
Genetic Improvement



GI 2019

GI 2019, Montreal,  
25-31 May 2019  
ICSE workshop

Submissions due  
**1 Feb 2019**



Humies: Human-Competitive  
Cash prizes  
GECCO-2019

20<sup>th</sup> birthday special issue  
by 17 Oct 2018

W. B. Langdon, UCL <http://www.eprsrc.ac.uk/> **EPSRC**

END

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

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



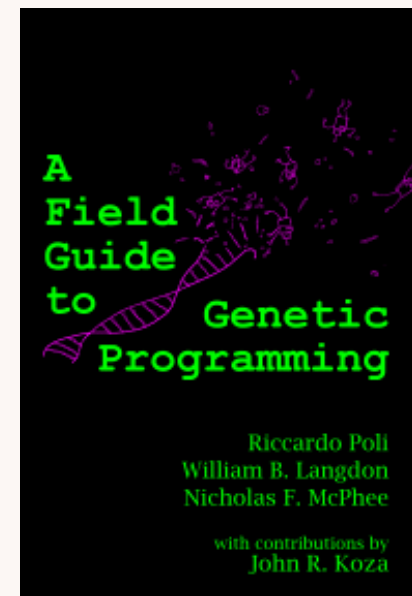
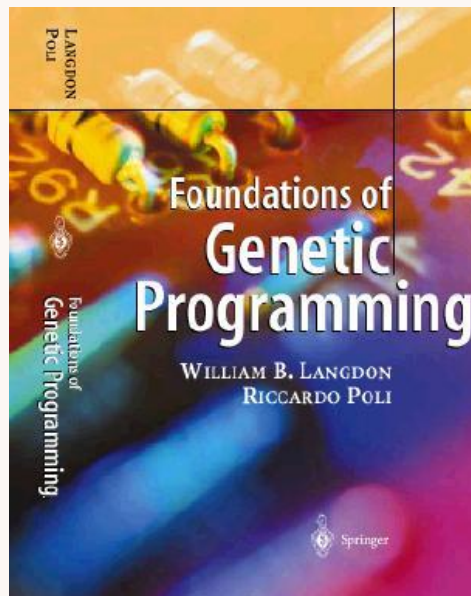
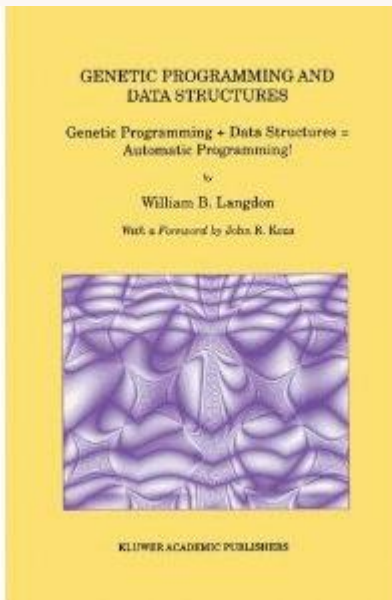
# Genetic Programming



W. B. Langdon

CREST

Department of Computer Science



**Table 4** Dataset of 10 subjects programs used in our experiments

Subject	LOC	Description	Oracle
quicksort	42	sort an array of integers	$\text{quicksort}(x) = \text{reference output}$
zip	56	compress a string with LZW	$\text{uncompress}(\text{compress}(x)) = x$
sudoku	87	solve a 9x9 sudoku grid	$\text{sudoku}(x) = \text{reference output}$
md5	91	compute the MD5 hash of a string	$\text{md5}(x) = \text{reference output}$
rsa	281	RSA encrypt/decrypt with public and private keys	$\text{decrypt}(\text{encrypt}(x)) = x$
rc4	146	RC4 encrypt/decrypt with symmetric key	$\text{decrypt}(\text{encrypt}(x)) = x$
canny	568	edge detector	$\text{canny}(x) = \text{reference output}$
lcs	43	compute the longest common sequence	$\text{lcs}(x) = \text{reference output}$
laguerre	440	find the roots of a polynomial functions	$ \text{poly}(\text{root})  < 10^{-6}, \forall \text{root} \in \text{laguerre}(\text{poly})$
linreg	188	compute the linear regression model for a set of points	$ \text{linreg}(x) - \text{reference coefficients}  < 10^{-6}$

**Table 6** PONE Results

Subject	$N_{pp}^{int}$	—Search space—	# Fragile exp.	#Robust exp.	# Antifragile exp.	$\Phi$ : Correctness ratio
quicksort	41	151444	6	10	19	————— 77%
zip	19	38840	5	2	5	————— 76%
sudoku	89	98211	12	27	8	————— 68%
md5	164	237680	102	24	7	— 29%
rsa	117	2576	55	8	32	————— 54%
rc4	115	165140	60	7	12	————— 38%
canny	450	616161	58	129	133	————— 94%
lcs	79	231786	10	47	13	————— 89%
laguerre	72	423454	15	24	15	————— 90%
linreg	75	543720	43	18	11	————— 47%
Total	1221	2509012	366	296	255	————— 66%

**Table 9** Dataset of 10 subjects programs used in our experiments

Subject	LOC	$N_{pp}^{int}$	$N_{pp}^{bool}$	$N_{pp}$	description
quicksort	42	41	6	47	sort an array of integer
zip	56	19	6	25	compress a string without loss
sudoku	87	89	26	115	solve 9x9 sudoku grid
md5	91	164	10	174	hash a message 5
rsa	281	117	20	137	crypt with public and private keys
rc4	146	115	7	122	crypt with symmetric key
canny	568	450	79	529	edge detector
lcs	43	79	9	88	compute the longest common sequence
laguerre	440	72	25	97	find roots for polynomial functions
linreg	188	75	15	90	compute the linear regression from set of points

# The Genetic Programming Bibliography

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

**12579** references, [11000 authors](#)

**Make sure it has all of your papers!**

E.g. email [W.Langdon@cs.ucl.ac.uk](mailto:W.Langdon@cs.ucl.ac.uk) or use | [Add to It](#) | web link

[XML](#) [RSS](#)

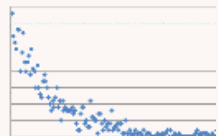
RSS Support available through the  
Collection of CS Bibliographies.



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

Co-authorship community.  
Downloads

Downloads by day



Your papers



A personalised list of every author's  
GP publications.

[blog](#)

Search the GP Bibliography at

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