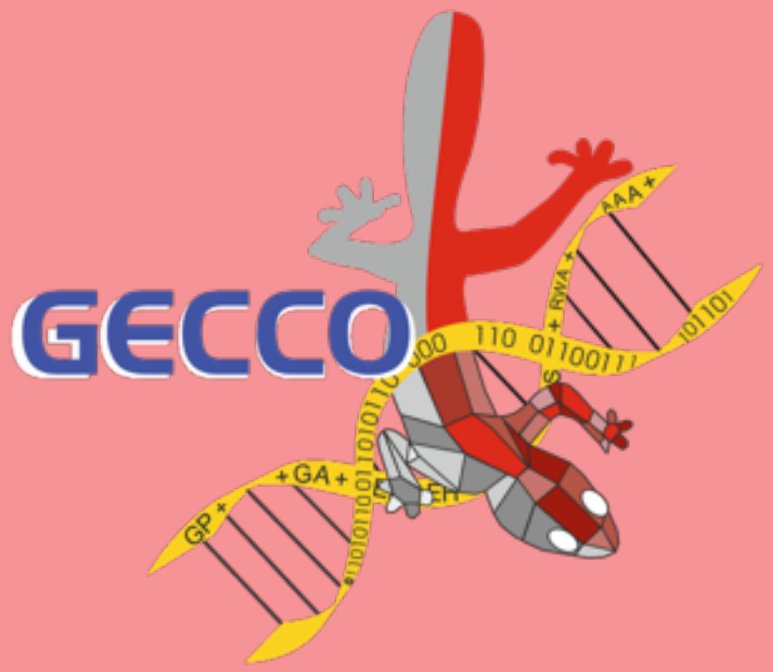
General Program Synthesis Benchmark Suite

Thomas Helmuth
Lee Spector
Hampshire College & University of Massachusetts, Amherst



Madrid, Spain
July 11-15, 2015

Genetic and Evolutionary Computation Conference

Outline

- Motivation
- Software synthesis benchmark suite
- Illustrative experiment
- Conclusions

Motivation

- Demand for benchmarks in GP more generally
- General program synthesis (automatic programming) is a long-standing goal of the field
- Few existing benchmarks for general program synthesis
- Purpose: help researchers assess the ability of a system to automate human programming

Tests

Software

Desiderata

- A program synthesis benchmark suite should require:
  - Multiple data types and data structures
  - Control flow
  - Large instruction sets
  - Larger programs than can be found by brute force

Sources

- ***iJava***: an interactive introductory computer science text-book with automatically graded programming problems [Moll]
- ***IntroClass***: a dataset designed for benchmarking automatic software defect repair systems [Le Goues, Holtschulte, Smith, Brun, Devanbu, Forrest, Weimer]

Criteria

- A range of inputs that have known correct outputs
- Present challenges typical of real programming tasks
- Agnostic with respect to programming language and synthesis technique

29 Synthesis Benchmarks

- **From *iJava***: Number IO, Small or Large, For Loop Index, Compare String Lengths, Double Letters, Collatz Numbers, Replace Space with Newline, String Differences, Even Squares, Wallis Pi, String Lengths Backwards, Last Index of Zero, Vector Average, Count Odds, Mirror Image, Super Anagrams, Sum of Squares, Vectors Summed, X-Word Lines, Pig Latin, Negative to Zero, Scrabble Score, Word Stats
- **From *IntroClass***: Checksum, Digits, Grade, Median, Smallest, Syllables
- PushGP has solved all of these except for the ones in blue

1. **Number IO (Q 3.5.1)** Given an integer and a float, print their sum.

2. **Small or Large (Q 4.6.3)** Given an integer n, print "small" if $n < 1000$ and "large" if $n \geq 2000$ (and nothing if $1000 \leq n < 2000$).

3. **For Loop Index (Q 4.11.7)** Given 3 integer inputs $start, end$, and $step$, print the integers in the sequence

$$n_0 = start$$
$$n_i = n_{i-1} + step$$

   for each $n_i < end$, each on their own line.

4. **Compare String Lengths (Q 4.11.13)** Given three strings $n1$, $n2$, and $n3$, return true if $length(n1) < length(n2) < length(n3)$, and false otherwise.

5. **Double Letters (P 4.1)** Given a string, print the string, doubling every letter character, and tripling every exclamation point. All other non-alphabetic and non-exclamation characters should be printed a single time each.

6. **Collatz Numbers (P 4.2)** Given an integer, find the number of terms in the Collatz (hailstone) sequence starting from that integer.

7. **Replace Space with Newline (P 4.3)** Given a string input, print the string, replacing spaces with newlines. Also, return the integer count of the non-whitespace characters. The input string will not have tabs or newlines.

8. **String Differences (P 4.4)** Given 2 strings (without whitespace) as input, find the indices at which the strings have different characters, stopping at the end of the shorter one. For each such index, print a line containing the index as well as the character in each string. For example, if the strings are "dealer" and "dollars", the program should print:

```
1 e o
2 a l
4 e a
```

9. **Even Squares (Q 5.4.1)** Given an integer n, print all of the positive even perfect squares less than n on separate lines.

10. **Wallis Pi (P 6.4))** John Wallis gave the following infinite product that converges to $\pi/4$:

$$\frac{2}{3} \times \frac{4}{3} \times \frac{4}{5} \times \frac{6}{5} \times \frac{6}{7} \times \frac{8}{7} \times \frac{8}{9} \times \frac{10}{9} \times ...$$

Given an integer input n, compute an approximation of this product out to n terms. Results are rounded to 5 decimal places.

11. **String Lengths Backwards (Q 7.2.5)** Given a vector of strings, print the length of each string in the vector starting with the last and ending with the first.

12. **Last Index of Zero (Q 7.7.8)** Given a vector of integers, at least one of which is 0, return the index of the last occurrence of 0 in the vector.

13. **Vector Average (Q 7.7.11)** Given a vector of floats, return the average of those floats. Results are rounded to 4 decimal places.

14. **Count Odds (Q 7.7.12)** Given a vector of integers, return the number of integers that are odd, without use of a specific `even` or `odd` instruction (but allowing instructions such as `mod` and `quotient`).

15. **Mirror Image (Q 7.7.15)** Given two vectors of integers, return `true` if one vector is the reverse of the other, and `false` otherwise.

16. **Super Anagrams (P 7.3)** Given strings x and y of lowercase letters, return true if y is a super anagram of x, which is the case if every character in x is in y. To be true, y may contain extra characters, but must have at least as many copies of each character as x does.

17. **Sum of Squares (Q 8.5.4)** Given integer n, return the sum of squaring each integer in the range $[1, n]$.

18. **Vectors Summed (Q 8.7.6)** Given two equal-sized vectors of integers, return a vector of integers that contains the sum of the input vectors at each index.

19. **X-Word Lines (P 8.1)** Given an integer X and a string that can contain spaces and newlines, print the string with exactly X words per line. The last line may have fewer than X words.

20. **Pig Latin (P 8.2)** Given a string containing lowercase words separated by single spaces, print the string with each word translated to pig Latin. Specifically, if a word starts with a vowel, it should have "ay" added to its end; otherwise, the first letter is moved to the end of the word, followed by "ay".

21. **Negative To Zero (Q 9.6.8)** Given a vector of integers, return the vector where all negative integers have been replaced by 0.

22. **Scrabble Score (P 10.1)** Given a string of visible ASCII characters, return the Scrabble score for that string. Each letter has a corresponding value according to normal Scrabble rules, and non-letter characters are worth zero.

23. **Word Stats (P 10.5)** Given a file, print the number of words containing n characters for n from 1 to the length of the longest word, in the format:

```
words of length 1:  12
words of length 2:  3
words of length 3:  0
words of length 4:  5
...
```

At the end of the output, print a line that gives the number of sentences and line that gives the average sentence length using the form:

```
number of sentences:  4
average sentence length:  7.452423455
```

A word is any string of consecutive non-whitespace characters (including sentence terminators). Every file will contain at least one sentence terminator (period, exclamation point, or question mark). The average sentence length is the number of words in the file divided by the number of sentence terminator characters.

24. **Checksum** Given a string, convert each character in the string into its integer ASCII value, sum them, take the sum modulo 64, add the integer value of the space character, and then convert that integer back into its corresponding character (the checksum character). The program must print `Check sum is X`, where X is replaced by the correct checksum character.

25. **Digits** Given an integer, print that integer's digits each on their own line starting with the least significant digit. A negative integer should have the negative sign printed before the most significant digit.

26. **Grade** Given 5 integers, the first four represent the lower numeric thresholds for achieving an A, B, C, and D, and will be distinct and in descending order. The fifth represents the student's numeric grade. The program must print `Student has a X grade.`, where X is A, B, C, D, or F depending on the thresholds and the numeric grade.

27. **Median** Given 3 integers, print their median.

28. **Smallest** Given 4 integers, print the smallest of them.

29. **Syllables** Given a string containing symbols, spaces, digits, and lowercase letters, count the number of occurrences of vowels (a, e, i, o, u, y) in the string and print that number as X in `The number of syllables is X`.

Using the Suite

- Seek ***success*** (passing all tests in training set)
- Seek generalization (passing all tests in test set)
- Seek high rates of success
- Use program evaluation limits
- Be reasonable about language feature and synthesis technique differences; it will not be possible to make comparisons that are "fair" in all ways

Push

- Designed for program evolution
- Data flows via stacks, not syntax
- One stack per type:
  integer, float, boolean, string, code, exec, vector, ...
- Rich data and control structures
- Minimal syntax:
  program → instruction | literal | (program*)
- Uniform variation, meta-evolution

Plush

Instruction	integer_eq	exec_dup	char_swap	integer_add	exec_if	
Close?	2	0	0	0	1	
Silence?	1	0	0	1	0	

Problem	# Instructions	exec	integer	float	boolean	char	string	vector of integers	vector of floats	vector of strings	print	file input	Terminals (besides inputs)
Number IO	50		x	x							x		integer ERC, float ERC
Small Or Large	103	x	x		x		x				x		"small", "large", integer ERC
For Loop Index	74	x	x		x						x		
Compare String Lengths	98	x	x		x		x						boolean ERC
Double Letters	132	x	x		x	x	x				x		\!
Collatz Numbers	102	x	x	x	x								0, 1, integer ERC
Replace Space with Newline	135	x	x		x	x	x				x		\space, \newline, string ERC, char ERC
String Differences	135	x	x		x	x	x				x		\space, \newline, integer ERC
Even Squares	72	x	x		x						x		
Wallis Pi	103	x	x	x	x								2 integer ERCs, 2 float ERCs
String Lengths Backwards	134	x	x		x		x			x	x		integer ERC
Last Index of Zero	101	x	x		x			x					0
Vector Average	88	x	x	x					x				
Count Odds	104	x	x		x			x					0, 1, 2, integer ERC
Mirror Image	102	x	x		x			x					boolean ERC
Super Anagrams	129	x	x		x	x	x						boolean ERC, char ERC, integer ERC
Sum of Squares	71	x	x		x								0, 1, integer ERC
Vectors Summed	68	x	x					x					[], integer ERC
X-Word Lines	134	x	x		x	x	x				x		\newline, \space
Pig Latin	141	x	x		x	x	x				x		"ay", \space, \a, \e, \i, \o, \u, "aeiou", string ERC, char ERC
Negative To Zero	102	x	x		x			x					0, []
Scrabble Score	158	x	x		x	x	x	x					vector containing Scrabble values (indexed by ASCII values)
Word Stats	281	x	x	x	x	x	x	x	x	x	x	x	\., \?, \!, \space, \tab, \newline, [], "words of length ", ": ", "number of sentences: ", "average sentence length: ", integer ERC
Checksum	136	x	x		x	x	x				x		"Check sum is ", \space, 64, integer ERC, char ERC
Digits	133	x	x		x	x	x				x		\newline, integer ERC [-10, 10]
Grade	112	x	x		x		x				x		"Student has a ", " grade.", "A", "B", "C", "D", "F", integer ERC
Median	75	x	x		x						x		integer ERC
Smallest	76	x	x		x						x		integer ERC
Syllables	141	x	x		x	x	x				x		"The number of syllables is ", "aeiouy", \a, \e, \i, \o, \u, \y, char ERC, string ERC

Selection

- In genetic programming, selection is typically based on average performance across all test cases (sometimes weighted, e.g. with "implicit fitness sharing")
- In nature, selection is typically based on sequences of interactions with the environment

Lexicase Selection

- Emphasizes individual test cases and combinations of test cases; not aggregated fitness across test cases
- Random ordering of test cases for each selection event

Lexicase Selection

To select single parent:

1. Shuffle test cases
2. First test case – keep best individuals
3. Repeat with next test case, etc.

Until one individual remains

The selected parent may be a specialist in the tests that happen to have come first, and may or may not be particularly good on average

Implicit Fitness Sharing

- Scale errors per case based on population-wide error
- Non-binary version

$$f_{NBIFS}(i) = \sum_{t \in T} \frac{f(i,t)}{\sum_{i' \in P} f(i',t)}$$

- All successes shown here generalize across the testing set
- Many non-generalizing "solutions" were also found

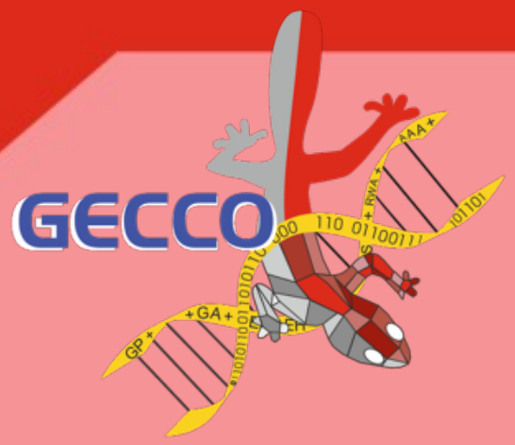
Problem	Tourn	IFS	Lex	Size
Number IO	68	72	98	5
Small Or Large	3	3	5	27
For Loop Index	0	0	1	21
Compare String Lengths	3	6	7	11
Double Letters	0	0	6	20
Collatz Numbers	0	0	0	
Replace Space with Newline	8	16	51	9
String Differences	0	0	0	
Even Squares	0	0	2	37
Wallis Pi	0	0	0	
String Lengths Backwards	7	10	66	9
Last Index of Zero	8	4	21	5
Vector Average	14	13	16	7
Count Odds	0	0	8	7
Mirror Image	46	64	78	4
Super Anagrams	0	0	0	
Sum of Squares	2	0	6	7
Vectors Summed	0	0	1	11
X-Word Lines	0	0	8	15
Pig Latin	0	0	0	
Negative To Zero	10	8	45	8
Scrabble Score	0	0	2	14
Word Stats	0	0	0	
Checksum	0	0	0	
Digits	0	1	7	20
Grade	0	0	4	52
Median	7	43	45	10
Smallest	75	98	81	8
Syllables	1	7	18	14
Problems Solved	13	13	22	

Results and Metaresults

- Benchmarks representative of novice programming tasks
- Benchmarks range in difficulty
- PushGP can solve many of them
- Lexicase selection often helps substantially

Conclusions

- GP can now automate some human programming
- Proposed benchmarks can guide and assess progress
- Full details in technical report: https://web.cs.umass.edu/publication/details.php?id=2387
- Data: https://github.com/thelmuth/Program-Synthesis-Benchmark-Data
- Coming soon: Tom Helmuth's dissertation!



Thanks

- Members of the Hampshire College Computational Intelligence Lab.
- This material is based upon work supported by the National Science Foundation under Grants No. 1017817, 1129139, and 1331283. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.