Evolving Code

Lee Spector
Hampshire College & UMass Amherst

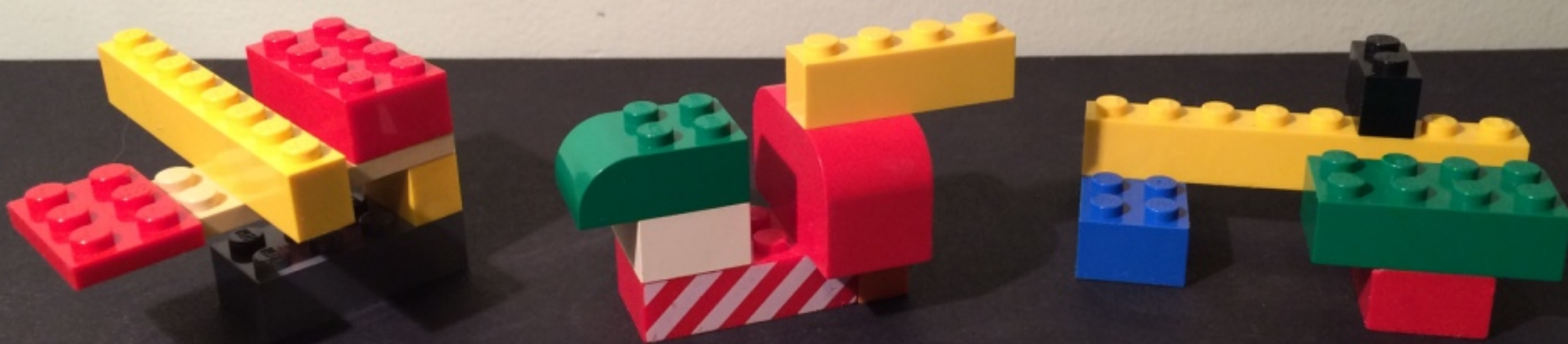
Outline

- Evolving code
- Language, variation, selection
- Evolving evolution
- Connections

Outline

- Evolving code
- Language, variation, selection
- Evolving evolution
- Connections

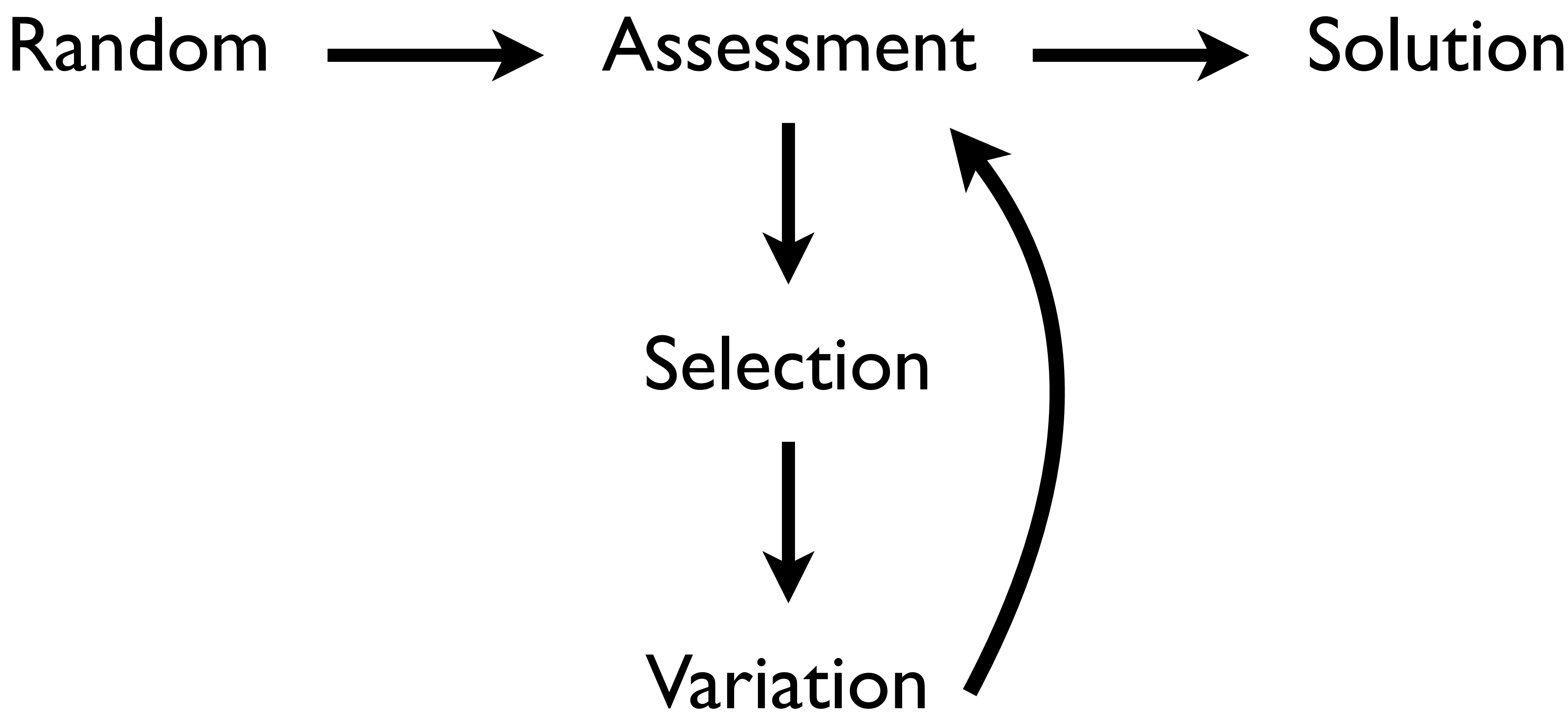
Evolving LEGO bridges



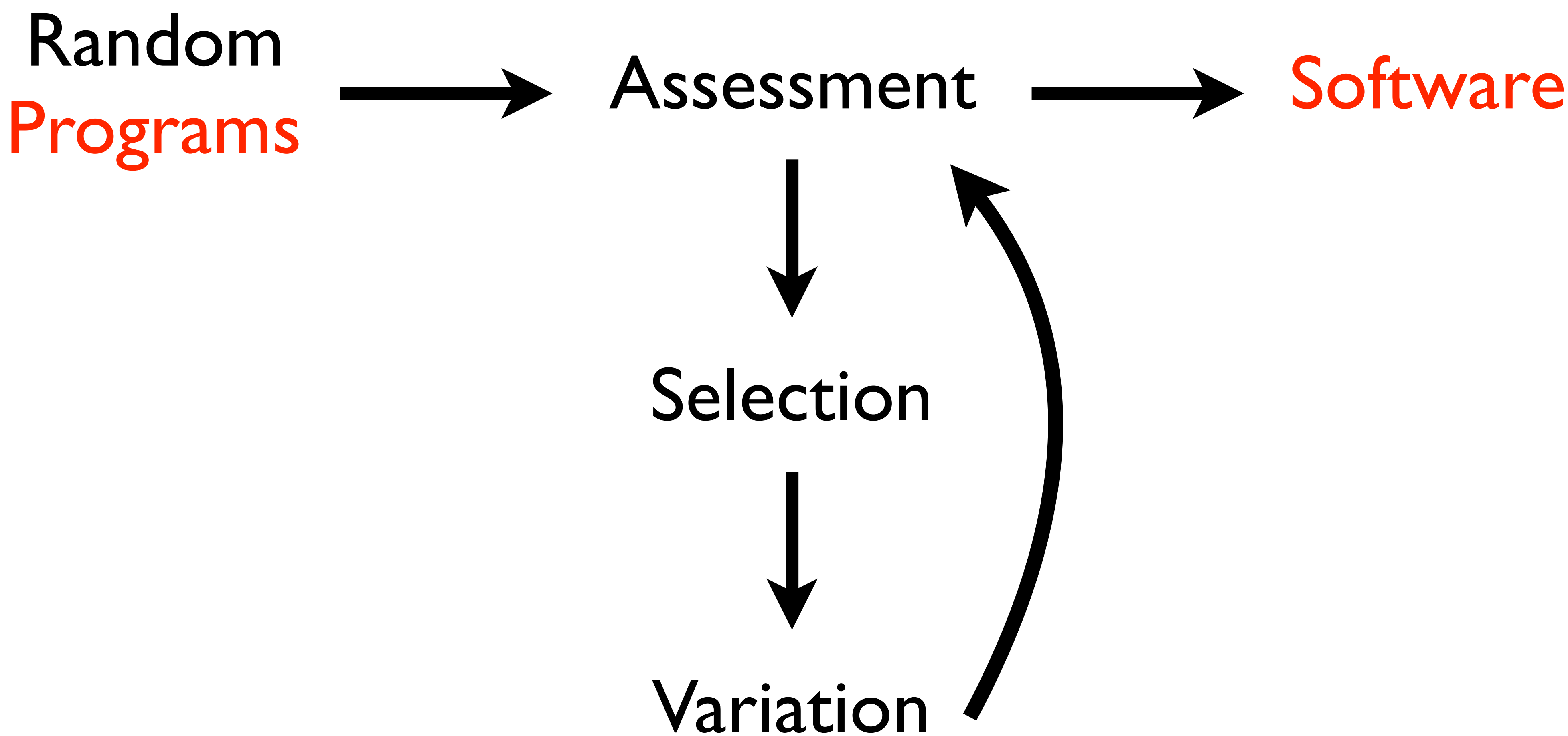
Evolving Code

- LEGO -> Code
- Bridge ->
  - Meets specification
  - Solves problem
  - Provides insight

Evolutionary Computing



Genetic Programming



Annual "Humies" Awards For Human-Competitive Results

Produced By Genetic And Evolutionary Computation

The result was **patented as an invention** in the past is an improvement over a patented invention or would qualify today as a patentable new invention.

The result is equal to or better than a result that was accepted as a **new scientific result** at the time when it was published in a peer-reviewed scientific journal.

The result is equal to or better than a result that was placed into a database or archive of results maintained by an **internationally recognized panel of scientific experts**.

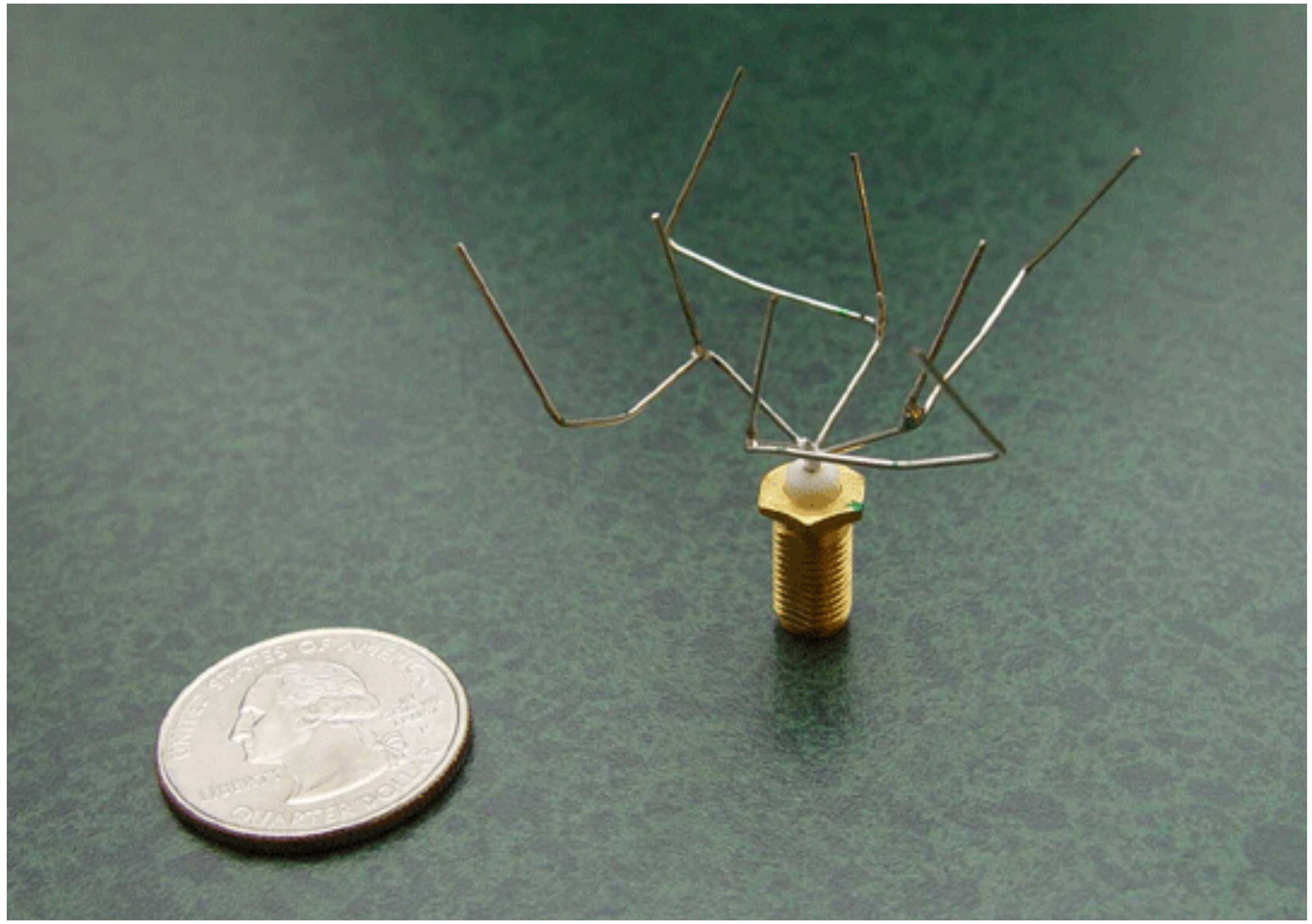
The result is **publishable in its own right** as a new scientific result independent of the fact that the result was mechanically created.

The result is **equal to or better than the most recent human-created solution** to a long-standing problem for which there has been a succession of increasingly better human-created solutions.

The result is equal to or better than a result that was considered an **achievement in its field** at the time it was first discovered.

The result **solves a problem of indisputable difficulty** in its field.

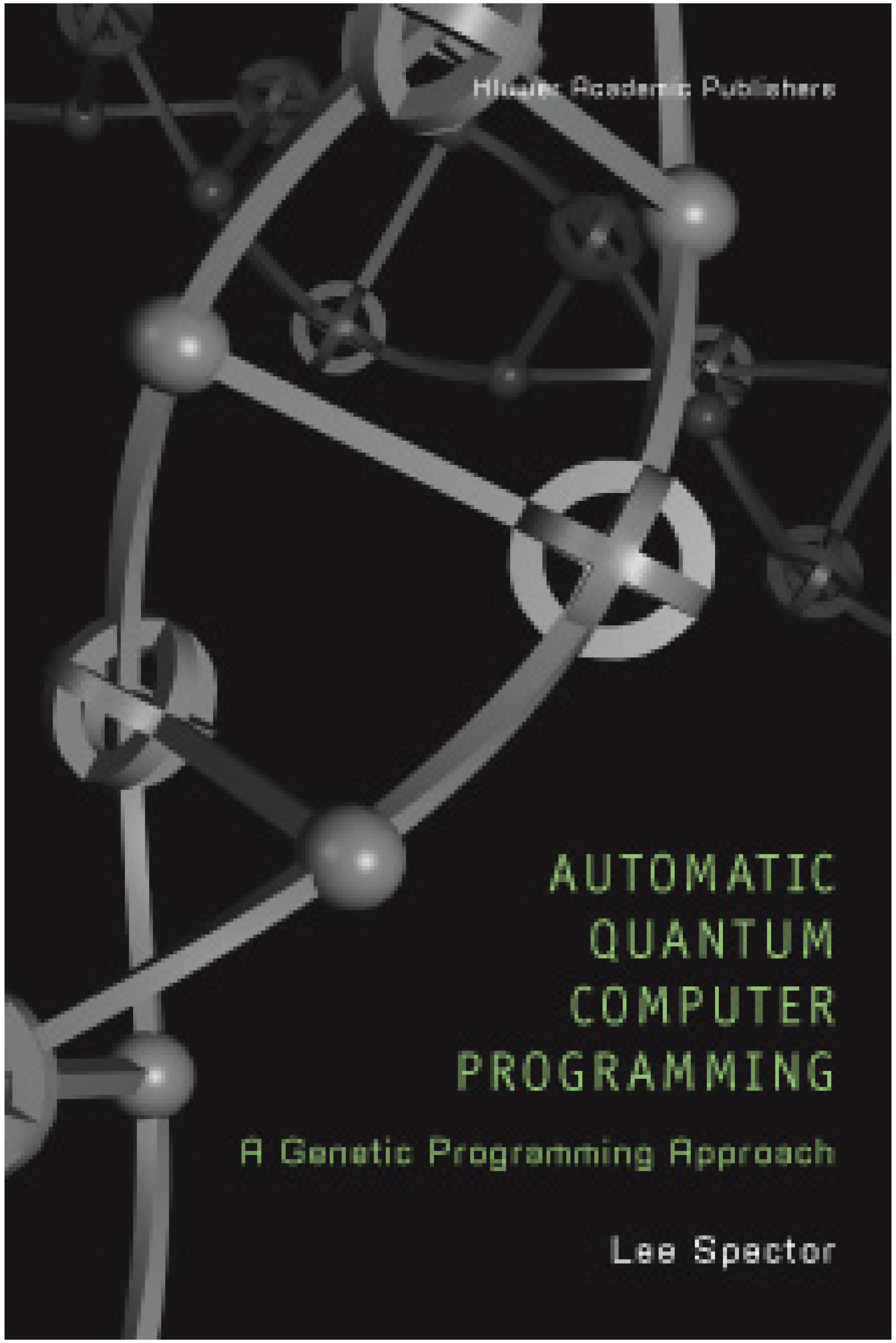
The result holds its own or **wins a regulated competition involving human contestants** (in the form of either live human players or human-written computer programs).



An Evolved Antenna for Deployment on NASA's Space Technology 5 Mission

Jason D. Lohn, Gregory S. Hornby, Derek S. Linden
NASA Ames Research Center

Humies Gold Medal, 2004



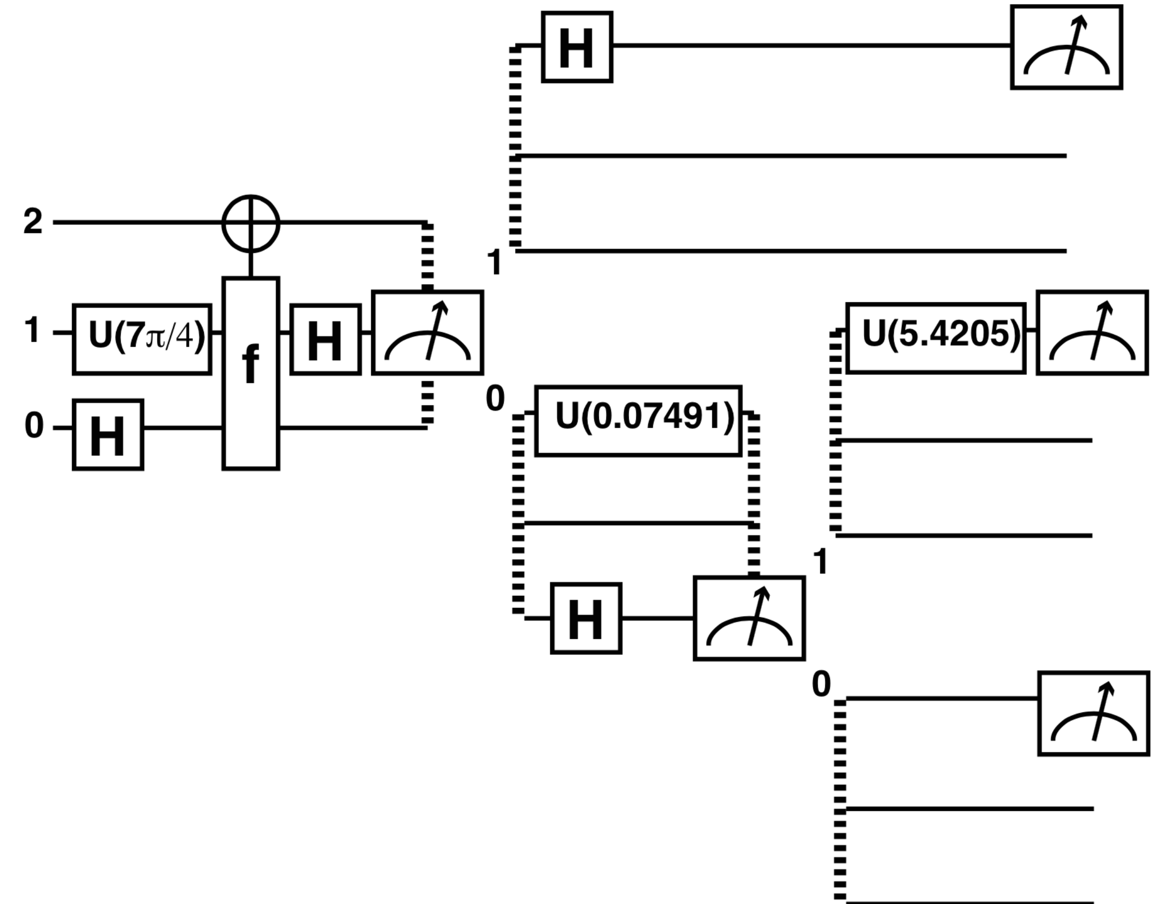


Figure 8.11. A gate array diagram for an evolved solution to the AND/OR oracle problem. The gate marked "f" is the oracle. The sub-diagrams on the right represent the possible execution paths following the intermediate measurements.

Lee Spector
Hampshire College

Humies Gold Medal, 2004

Genetic Programming for Finite Algebras

Lee Spector
Cognitive Science
Hampshire College
Amherst, MA 01002
lspector@hampshire.edu

David M. Clark
Mathematics
SUNY New Paltz
New Paltz, NY 12561
clarkd@newpaltz.edu

Ian Lindsay
Hampshire College
Amherst, MA 01002
iml04@hampshire.edu

Bradford Barr
Hampshire College
Amherst, MA 01002
bradford.barr@gmail.com

Jon Klein
Hampshire College
Amherst, MA 01002
jk@artificial.com

$$((((((((x*(y*x))*x)*z)*(z*x))*((x*(z*(x*(z*y))))*z))*z)*z)*(z*((((x*(((z*z)*x)*(z*x)))*x)*y)*(((y*(z*(z*y)))*(((y*y)*x)*z))*(x*(((z*z)*x)*(z*(x*(z*y)))))))))$$

Humies Gold Medal, 2008

International Journal of Algebra and Computation | Vol. 28, No. 05, pp. 759-790 (2018)

Evolution of algebraic terms 3: Term continuity and beam algorithms

David M. Clark and Lee Spector

Yavalath

Yavalath is an abstract board game for two or three players, invented by a computer program called Ludi. It has an easy rule set that any player can pick up immediately, but which produces surprisingly tricky emergent play.

Yavalath is available from nestorgames, making it the first — and still only — computer-generated game to be commercially published, together with its sister game Pentalath.

In October 2011, Yavalath was ranked in the top #100 abstract board games ever invented on the BoardGameGeek database. This helped it win the GECCO "Humies" gold medal for human-competitive results in evolutionary computation for 2012.

Here is a Yavalath article in the November 2013 issue of Bitcoin magazine.

Rules

The board starts empty.

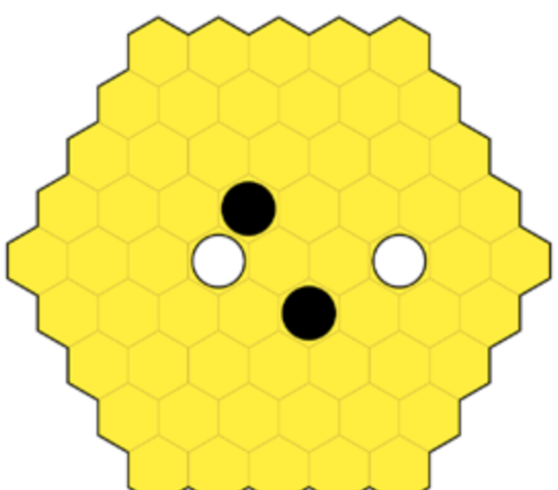
Two players take turns adding a piece of their colour to an empty cell.

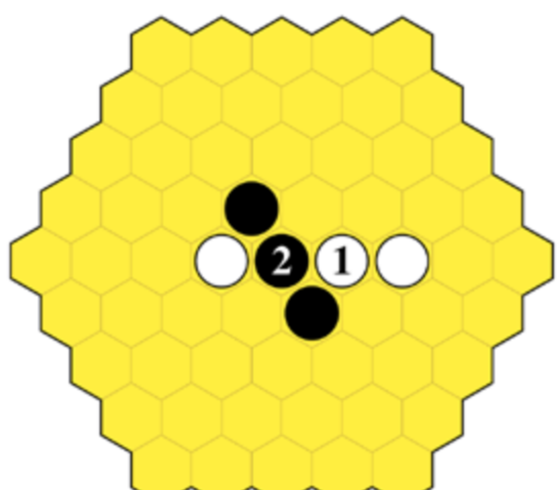
Win by making a line-of-4 (or more) pieces of your colour.
Lose by making a line-of-3 pieces of your colour beforehand.
Draw if the board otherwise fills up.

No, players are not allowed to pass.

Tactics and Strategy

The key tactical play in Yavalath is the *forcing move*, as shown below. White move **1** forces Black to lose with the blocking move **2**.





Cameron Browne
Imperial College London

Humies Gold Medal, 2012

Fixing software bugs in 10 minutes or less using evolutionary computation

University of New Mexico

Stephanie Forrest

ThanhVu Nguyen

University of Virginia

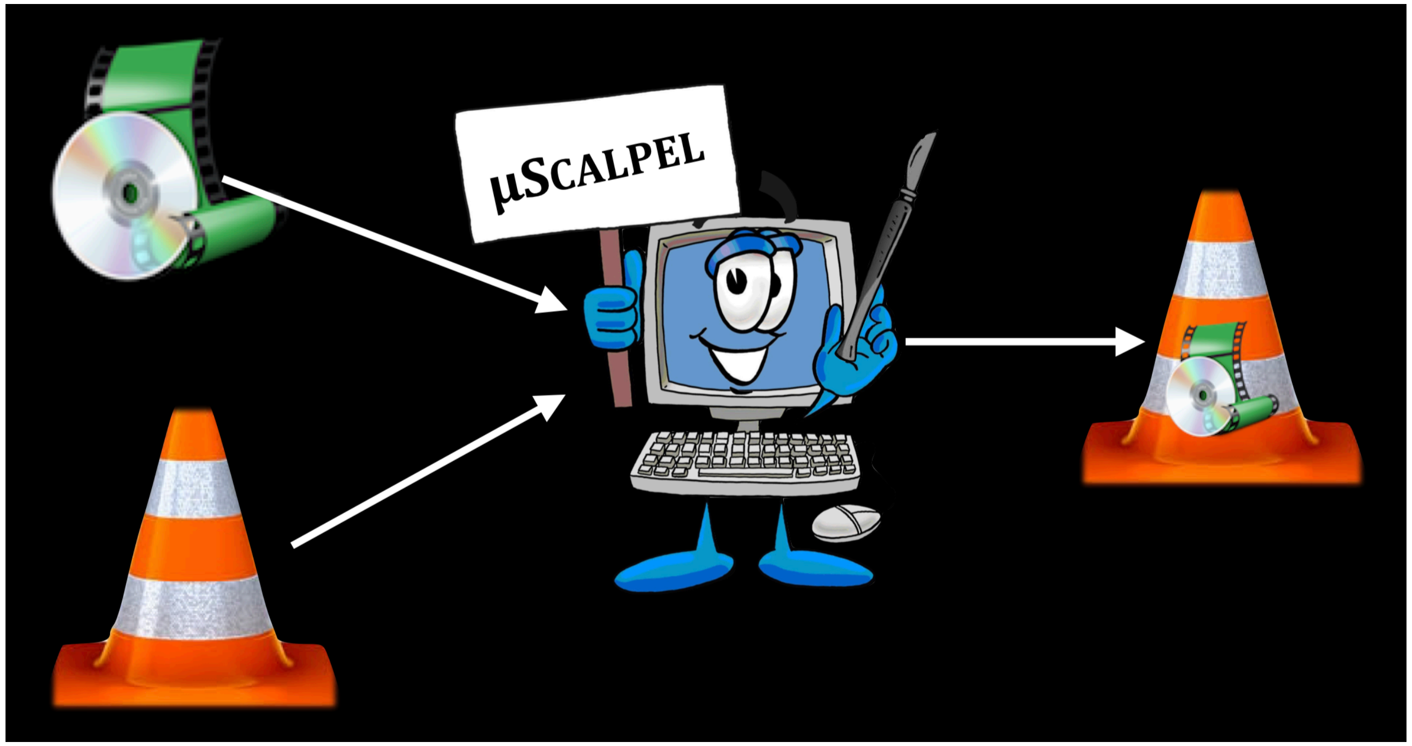
Claire Le Goues

Westley Weimer

THE UNIVERSITY *of* NEW MEXICO

Humies Gold Medal, 2009

Automated Software Transplantation



Earl T. Barr, Mark Harman, Yue Jia, Alexandru Marginean, Justyna Petke
University College London

Humies Gold Medal, 2016

Application	Count	Application Category
Antennas	1	Engineering (19)
Biology	2	Science (7)
Chemistry	1	Science (7)
Computer vision	2	Computer science (7)
Electrical engineering	1	Engineering (19)
Electronics	5	Engineering (19)
Games	6	Games (6)
Image processing	3	Computer science (7)
Mathematics	2	Mathematics (3)
Mechanical engineering	4	Engineering (19)
Medicine	2	Medicine (2)
Operations research	1	Engineering (19)
Optics	2	Engineering (19)
Optimization	1	Mathematics (3)
Photonics	1	Engineering (19)
Physics	1	Science (7)
Planning	1	Computer science (7)
Polymers	1	Engineering (19)
Quantum	3	Science (7)
Security	1	Computer science (7)
Software engineering	3	Engineering (19)

Problem Type	Count
Classification	5
Clustering	1
Design	20
Optimization	8
Planning	1
Programming	4
Regression	3

Kannappan, K., L. Spector, M. Sipper, T. Helmuth, W. La Cava, J. Wisdom, and O. Bernstein. 2015. Analyzing a decade of Human-competitive ("HUMIE") winners -- what can we learn? In *Genetic Programming Theory and Practice XII*. New York: Springer.

Evolution, the Designer

WHAT WOULD DARWIN SAY? | LEE SPECTOR

The Boston Globe

And now, digital evolution

By Lee Spector | August 29, 2005

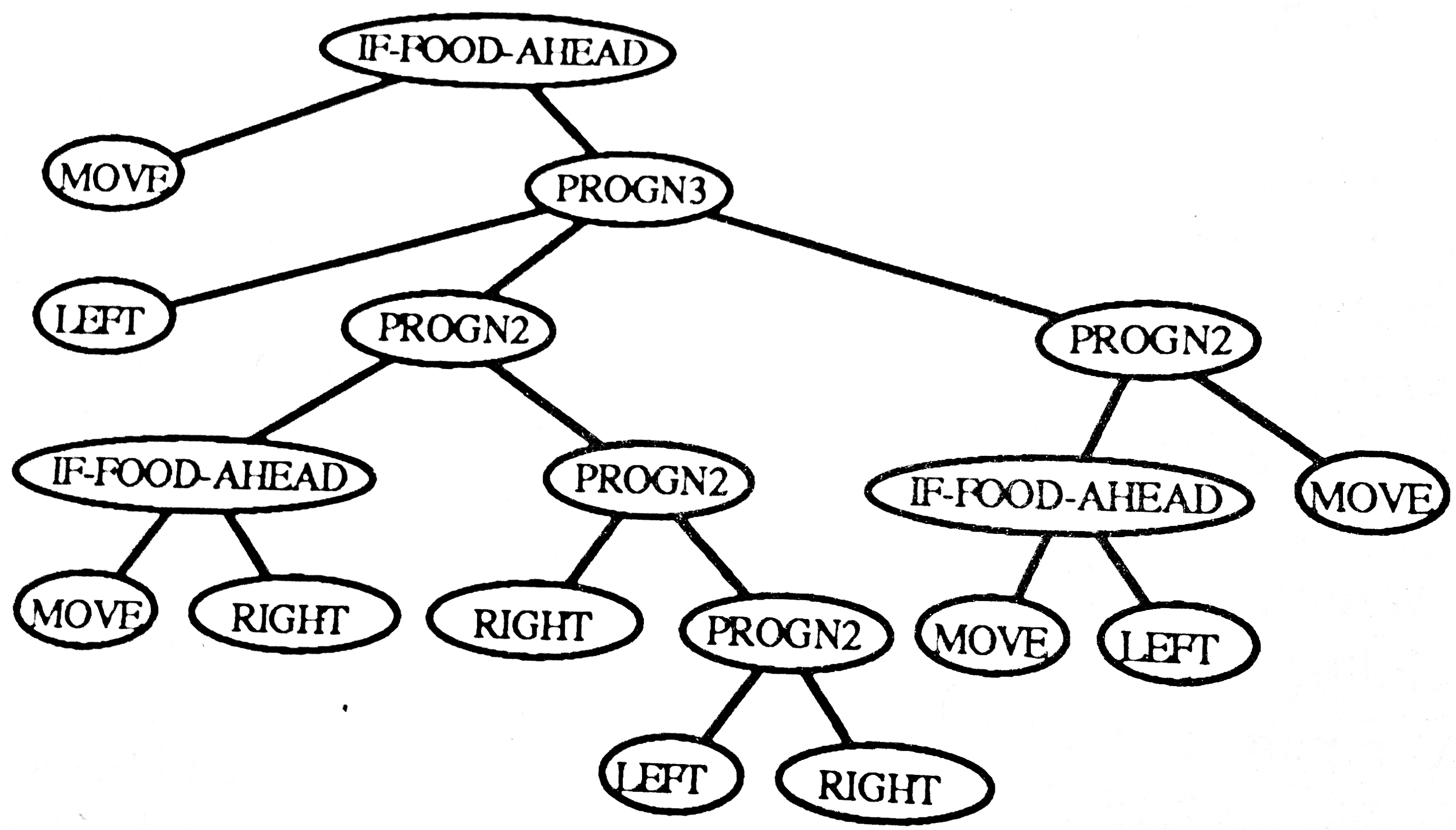
RECENT developments in computer science provide new perspective on "intelligent design," the view that life's complexity could only have arisen through the hand of an intelligent designer. These developments show that complex and useful designs can indeed emerge from random Darwinian processes.

“Darwinian evolution is itself a designer worthy of significant respect, if not religious devotion.”

Outline

- Evolving code
- Language, variation, selection
- Evolving evolution
- Connections

```
(IF-FOOD-AHEAD (MOVE)
               (PROGN3 (LEFT)
                       (PROGN2 (IF-FOOD-AHEAD (MOVE)
                                              (RIGHT))
                               (PROGN2 (RIGHT)
                                       (PROGN2 (LEFT)
                                               (RIGHT))))
                       (PROGN2 (IF-FOOD-AHEAD (MOVE)
                                              (LEFT))
                               (MOVE)))).
```



Koza, 1992

Languages

- Lisp-style symbolic expressions (Koza, ...)
- Purely functional/lambda expressions (Walsh, Yu, ...)
- Linear sequences of machine/byte code (Nordin et al., ...)
- Artificial assembly-like languages (Ray, Adami, ...)
- Stack-based languages (Perkis, Spector, Stoffel, Tchernev, ...)
- Graph-structured programs (Teller, Globus, ...)
- Object hierarchies (Bruce, Abbott, Schmutter, Lucas, ...)
- Fuzzy rule systems (Tunstel, Jamshidi, ...)
- Logic programs (Osborn, Charif, Lamas, Dubossarsky, ...)
- Strings, grammar-mapped to arbitrary languages (O'Neill, Ryan, ...)

Push

- Programming language for programs that evolve
- Simple syntax, but rich data and control structures:
  program → instruction | literal | (program*)
- Data flows via typed stacks, not syntax
- C++, Clojure, Common Lisp, Elixir, Java, Javascript, Python, Racket, Ruby, Scala, Scheme, Swift, ...
- http://pushlanguage.org

Push Execution

- Push the program onto the `exec` stack.
- While `exec` isn't empty and and we haven't hit the step limit, pop and **do** the top:
    - If it's an instruction, execute it.
    - If it's a literal, push it onto the appropriate stack.
    - If it's a block of code, push its elements back onto the `exec` stack one at a time.

`(1 2 integer_add)`

Exec **Integer** **Boolean** **String** **...**

`(1 2 integer_add)`

Exec **Integer** **Boolean** **String** **...**

1

2

integer_add

Exec **Integer** **Boolean** **String** **...**

1

2

integer_add

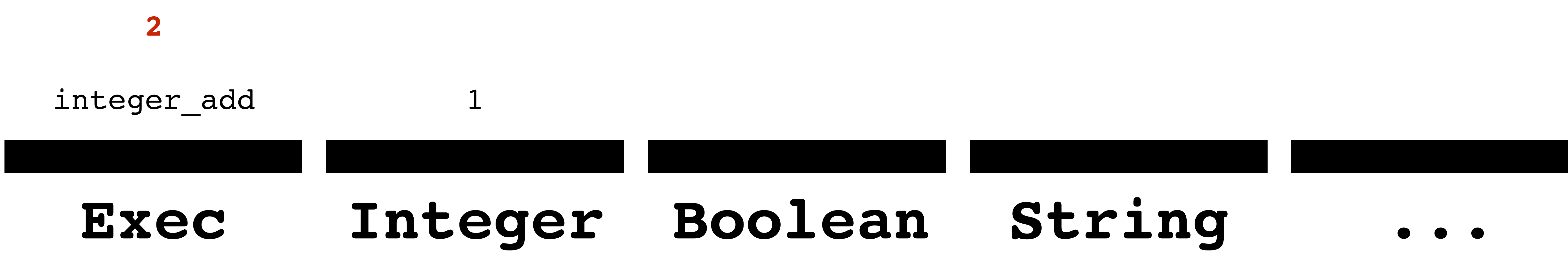
Exec **Integer** **Boolean** **String** **...**

2

integer_add

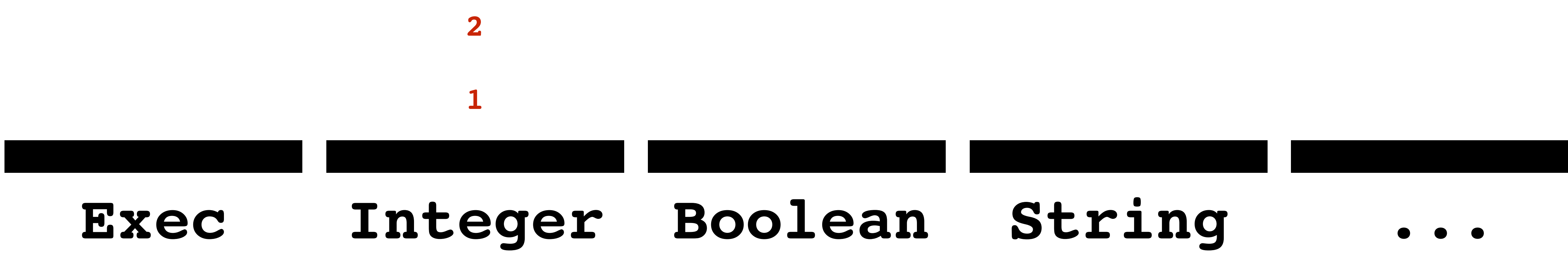
1

Exec **Integer** **Boolean** **String** **...**


2
integer_add
1
Exec
Integer
Boolean
String
...

Exec	Integer	Boolean	String	...
	2			
integer_add	1			

Exec	Integer	Boolean	String	...
	2			
integer_add	1			


2
1
Exec
Integer
Boolean
String
...

3

Exec Integer Boolean String ...

		true		
integer_mult		false		
boolean_and	7	true	"Hello"	
(3 string_dup)	-20	true	"Push"	
integer_add	100	false	"Evolution!"	
Exec	**Integer**	**Boolean**	**String**	**...**

		true		
		false		
boolean_and	**7**	true	"Hello"	
(3 string_dup)	**-20**	true	"Push"	
integer_add	100	false	"Evolution!"	
Exec	**Integer**	**Boolean**	**String**	**...**

Exec	Integer	Boolean	String	...
		true		
		false		
boolean_and		true	"Hello"	
(3 string_dup)	***-140***	true	"Push"	
integer_add	100	false	"Evolution!"	

Exec	Integer	Boolean	String	...
		true		
		false		
boolean_and		true	"Hello"	
(3 string_dup)	-140	true	"Push"	
integer_add	100	false	"Evolution!"	

		true		
		false		
		true	"Hello"	
(3 string_dup)	-140	true	"Push"	
integer_add	100	false	"Evolution!"	
Exec	**Integer**	**Boolean**	**String**	**...**

		false		
		true	"Hello"	
(3 string_dup)	-140	true	"Push"	
integer_add	100	false	"Evolution!"	
Exec	**Integer**	**Boolean**	**String**	**...**

Exec	Integer	Boolean	String	...
		false		
		true	"Hello"	
(3 string_dup)	-140	true	"Push"	
integer_add	100	false	"Evolution!"	

Exec	Integer	Boolean	String	...
		false		
		true	"Hello"	
(3 string_dup)	-140	true	"Push"	
integer_add	100	false	"Evolution!"	

Exec	Integer	Boolean	String	...
		false		
3		true	"Hello"	
string_dup	-140	true	"Push"	
integer_add	100	false	"Evolution!"	

Exec	Integer	Boolean	String	...
		false		
3		true	"Hello"	
string_dup	-140	true	"Push"	
integer_add	100	false	"Evolution!"	

Exec	Integer	Boolean	String	...
		false		
3		true	"Hello"	
string_dup	-140	true	"Push"	
integer_add	100	false	"Evolution!"	

		false		
	3	true	"Hello"	
string_dup	-140	true	"Push"	
integer_add	100	false	"Evolution!"	
Exec	**Integer**	**Boolean**	**String**	**...**

Exec	Integer	Boolean	String	...
		false		
	3	true	"Hello"	
string_dup	-140	true	"Push"	
integer_add	100	false	"Evolution!"	

Exec	Integer	Boolean	String	...
		false		
	3	true	**"Hello"**	
	-140	true	"Push"	
integer_add	100	false	"Evolution!"	

Exec	Integer	Boolean	String	...
		false	***"Hello"***	
	3	true	***"Hello"***	
	-140	true	"Push"	
integer_add	100	false	"Evolution!"	

		false	"Hello"	
	3	true	"Hello"	
	-140	true	"Push"	
integer_add	100	false	"Evolution!"	
Exec	**Integer**	**Boolean**	**String**	**...**

Exec	Integer	Boolean	String	...
		false	"Hello"	
	3	true	"Hello"	
	-140	true	"Push"	
	100	false	"Evolution!"	

Exec	Integer	Boolean	String	...
		false	"Hello"	
		true	"Hello"	
	-137	true	"Push"	
	100	false	"Evolution!"	

Example `exec` Instructions

Conditionals:

```
exec_if exec_when
```

General loops:

```
exec_do*while
```

“For” loops:

```
exec_do*range exec_do*times
```

Looping over structures:

```
exec_do*vector_integer exec_string_iterate
```

Combinators:

```
exec_k exec_y exec_s
```

Auto-Simplification

- Loop:
  - Make it randomly simpler
  - Keep simpler if as good or better; otherwise revert
- Efficiently and reliably reduces the size of evolved programs
- Often improves generalization

SUCCESS at generation 20

Successful program: (boolean_and boolean_shove exec_do*count (exec_swap (integer_empty char_yank boolean_or integer_fromboolean \space \newline) (exec_dup (char_yank char_iswhitespace string_butlast in1) string_empty boolean_frominteger tagged_275 string_substring) exec_do*times (integer_empty string_dup) string_replacechar print_string string_rot print_char integer_fromboolean string_length integer_eq string_last boolean_swap integer_yankdup) string_swap string_containschar "Wx{ " exec_stackdepth char_empty integer_swap integer_rot string_last boolean_swap integer_yankdup string_swap string_containschar "Wx{ " exec_stackdepth char_empty integer_swap integer_rot integer_fromstring string_pop string_shove char_eq char_empty integer_swap integer_rot integer_fromstring string_pop string_shove char_rot integer_stackdepth integer_min char_yankdup char_eq char_empty tagged_349 exec_yank string_rot exec_dup (boolean_eq string_removechar exec_s (exec_dup (boolean_eq exec_rot (exec_s (string_eq string_fromboolean exec_noop char_eq) () (string_butlast) integer_pop) (char_eq char_empty) (integer_swap integer_rot string_emptystring boolean_stackdepth integer_inc in1 boolean_shove boolean_swap char_isletter integer_gt integer_yankdup) exec_when (string_emptystring string_nth exec_do*range (\space integer_yankdup string_dup exec_shove (integer_swap string_removechar exec_yank string_dup exec_empty) char_eq exec_do*times (tagged_349 boolean_pop exec_when (string_removechar integer_mult integer_inc in1 boolean_shove boolean_swap char_isletter integer_gt string_butlast) integer_mult string_last string_parse_to_chars boolean_frominteger boolean_yank exec_when (string_nth exec_do*range (\space integer_yankdup string_dup exec_shove (integer_swap string_removechar exec_yank integer_yank exec_while (boolean_or)) char_isdigit boolean_swap char_isletter) integer_gt integer_yankdup integer_mult string_last string_parse_to_chars boolean_frominteger char_isletter exec_when (string_nth exec_do*range (\space integer_yankdup string_dup exec_shove (integer_swap string_removechar exec_yank integer_yank integer_mult integer_inc in1 boolean_shove boolean_swap char_isletter integer_gt string_butlast) boolean_invert_second_then_and exec_empty string_rot)) boolean_rot char_iswhitespace integer_yank string_conjchar boolean_dup) integer_add char_dup string_length integer_fromchar string_split char_isdigit boolean_swap boolean_eq char_isdigit exec_shove (boolean_invert_second_then_and string_empty string_conjchar string_shove) string_fromchar boolean_not string_stackdepth exec_y () integer_empty exec_do*range (in1 string_replace)))))) () ()))

Errors: [0 0]

Total error: 0.0

Size: 231

Auto-simplifying with starting size: 231

...

step: 5000

program: (\space \newline in1 string_replacechar print_string "Wx{ " string_last in1 string_removechar string_length)

errors: [0 0]

Total error: 0.0

Size: 11

Outline

- Evolving code
- Language, variation, selection
- Evolving evolution
- Connections

Variation

- Replacement mutations
- Crossover / alternation
- UMAD: Uniform Mutation by Addition and Deletion
- 2018 GECCO best paper, GP track

UMAD

- Two passes:
  - For each gene, maybe add gene before or after
  - Then, for each gene, maybe delete
- Size neutral if $\boldsymbol{d = a/(1+a)}$
- Capable of replacement, and so much more

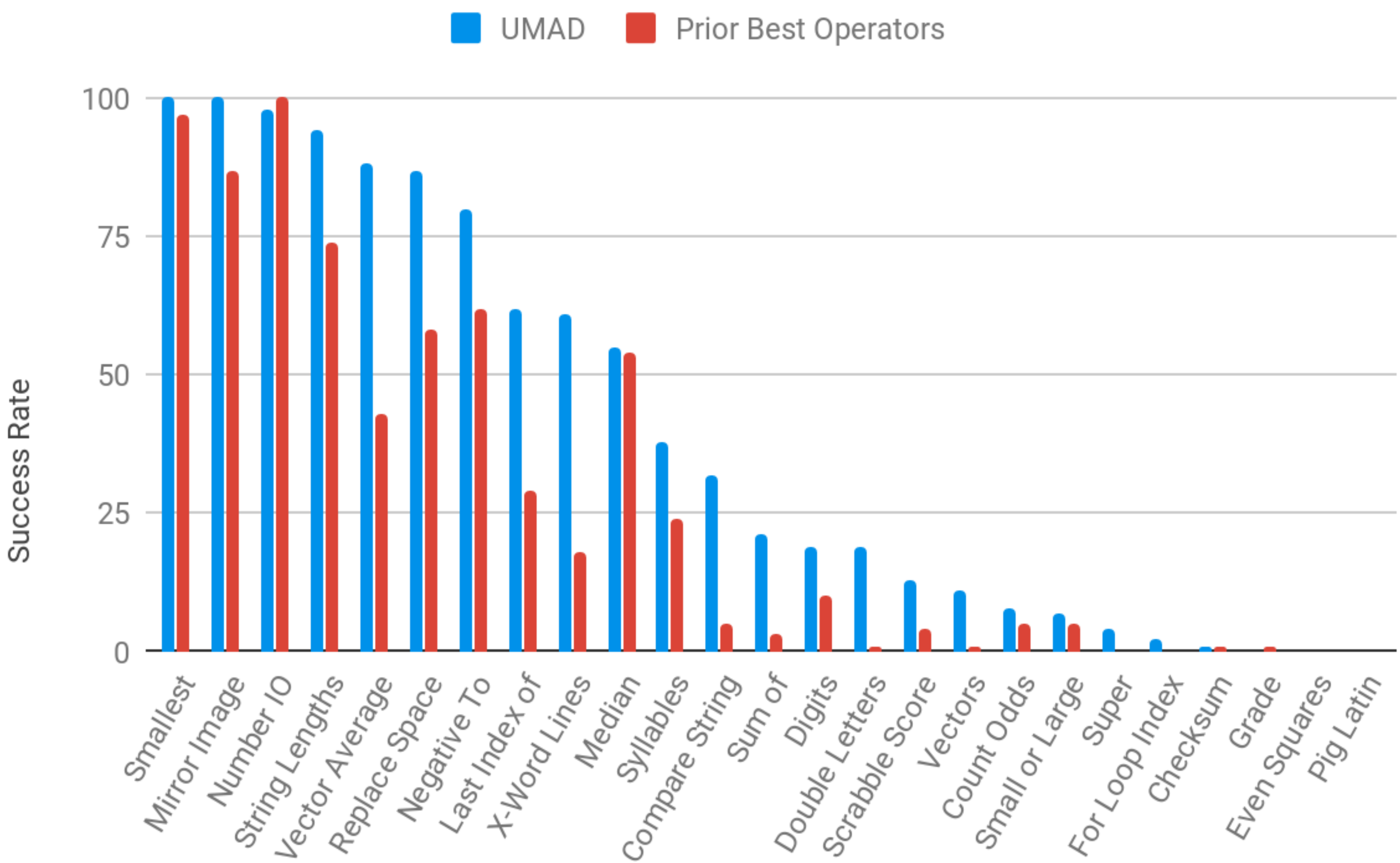
Software Synthesis

- 29 benchmark problems taken from intro CS textbooks
- Require multiple data types and control structures
- Driven by software tests, input/output pairs
- Used for studies of program synthesis, by us and by others

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
   ```



UMAD
Prior Best Operators
Success Rate
100
75
50
25
0
Smallest
Mirror Image
Number IO
String Lengths
Vector Average
Replace Space
Negative To
Last Index of
X-Word Lines
Median
Syllables
Compare String
Sum of
Digits
Double Letters
Scrabble Score
Vectors
Count Odds
Small or Large
Super
For Loop Index
Checksum
Grade
Even Squares
Pig Latin

Paths

- Suppose we have ABC, ADC is a solution, and the possible genes are just A, B, C, and D
- How many paths are there, of various lengths?
- Count as 1 step:
  - Replacement: replace 1 gene
  - UMAD: add 1 gene and/or delete 1 gene
- For replacement or UMAD, there is a single 1-step path, ABC→ADC

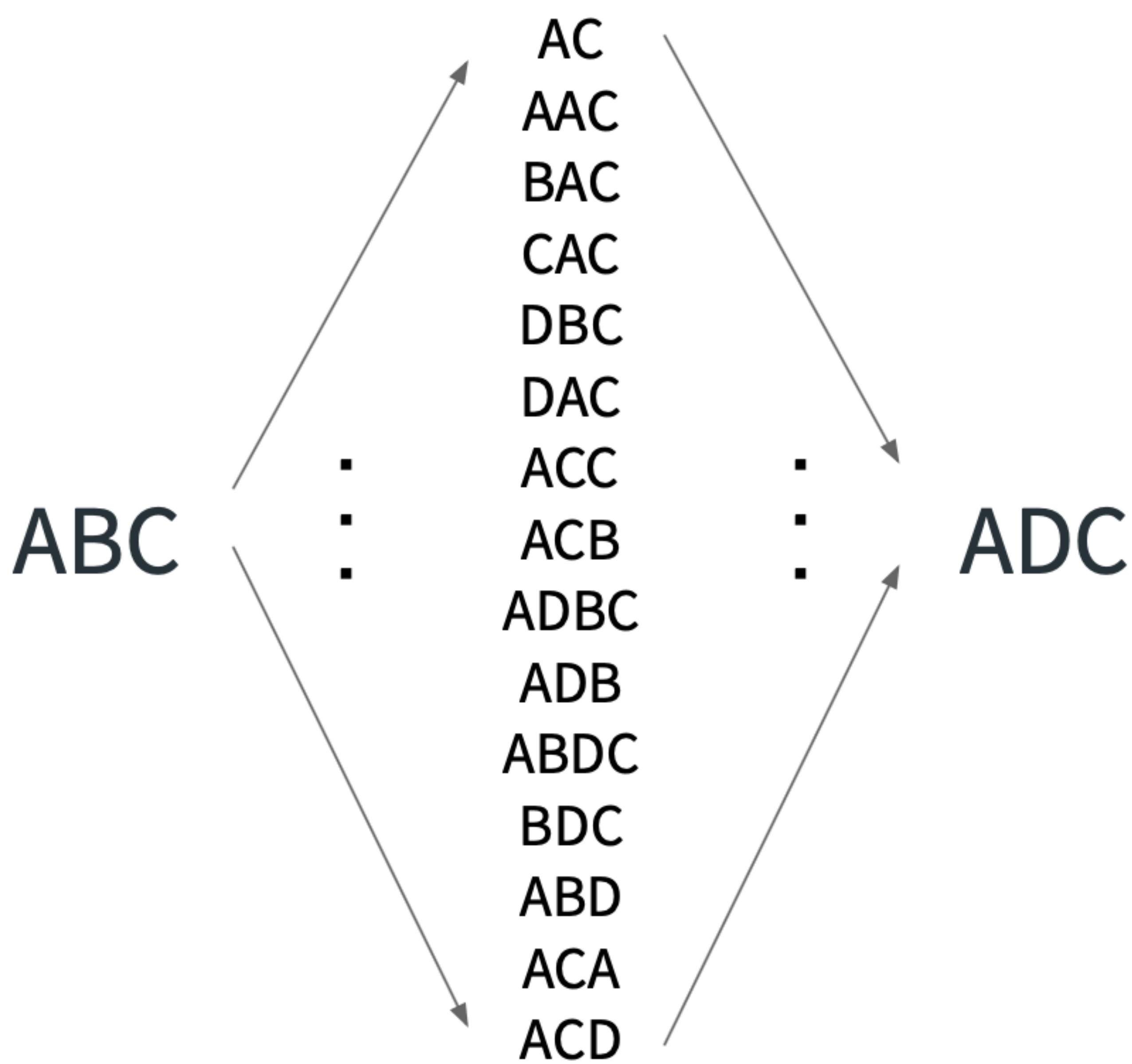
2-step paths, replacement

- 2 paths

- If AAC and ACC are unviable, neither path works

2-step paths, UMAD

- 15 paths



ABCDA → ADCDA

Number of Steps	Replacement	UMAD
1	1	1
2	2	25
3	14	974

Outline

- Evolving code
- Language, variation, selection
- Evolving evolution
- Connections

Parent selection

- Traditionally based on overall scores
- Roulette wheels or tournaments
- Unbalanced, qualitatively diverse test sets

Lexicase Selection

- Don't reduce to overall scores

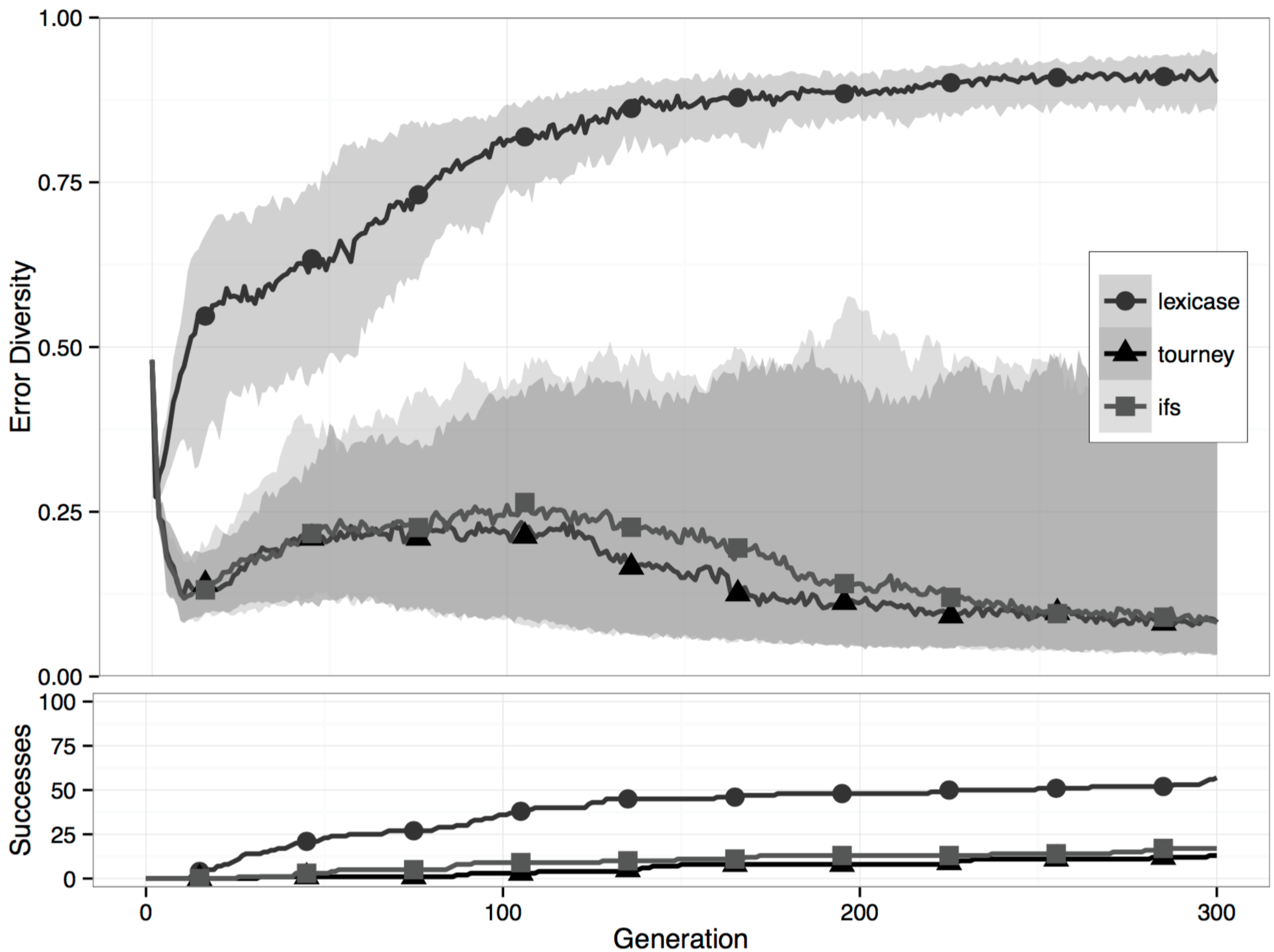
- To select single parent:
  1. Shuffle test cases
  2. First test case – keep best* individuals
  3. Repeat with next test case, etc.

  Until one individual remains

- Selected parent may be specialist, not great on average, but lead to generalists later

Problem name	Lexicase	Tournament
Replace Space With Newline	57	13
Syllables	24	1
String Lengths Backwards	75	18
Negative To Zero	72	15
Double Letters	5	0
Scrabble Score	0	0
Checksum	0	0
Count Odds	4	0

Diversity



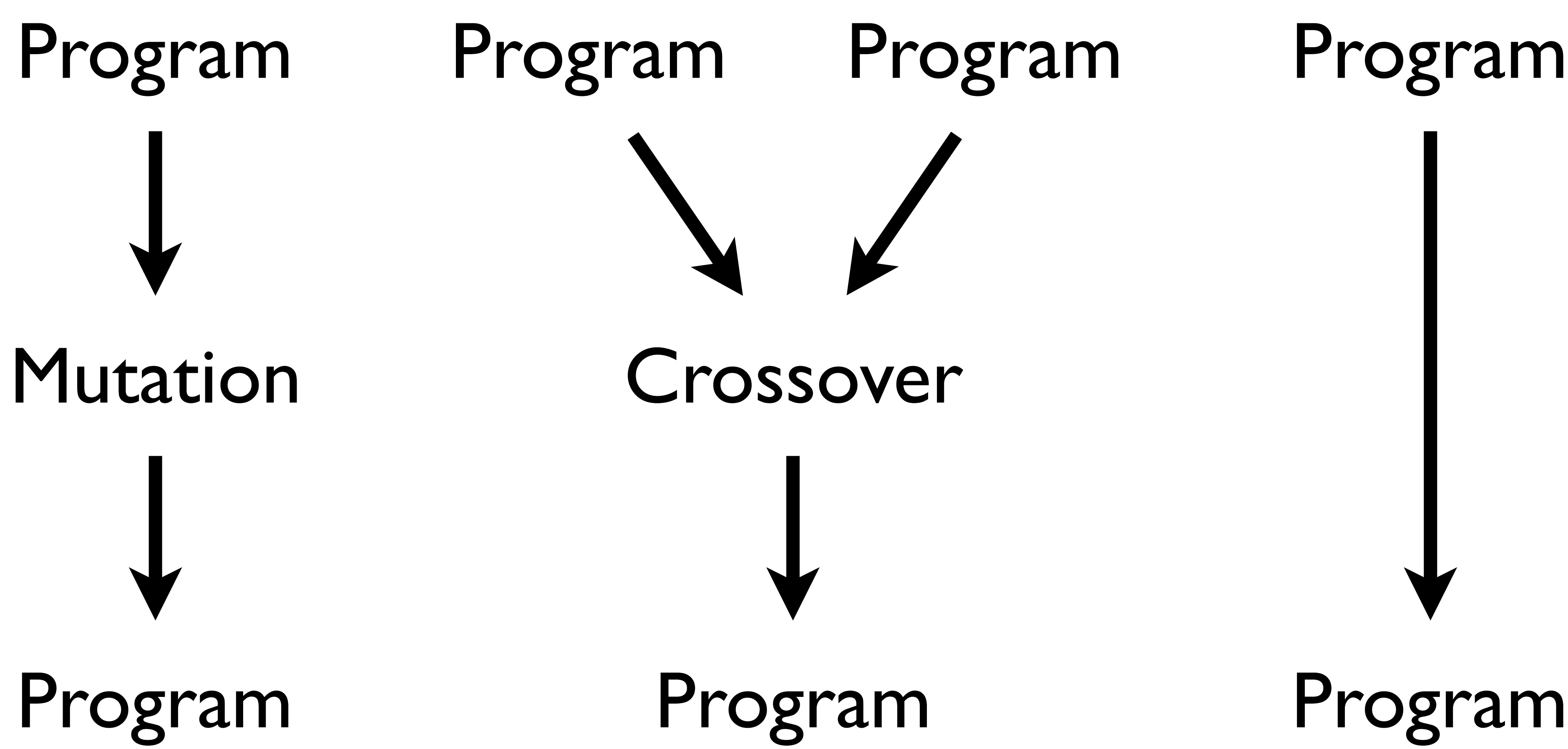
GPTP-2015

Fig. 1 Replace Space With Newline – error diversity

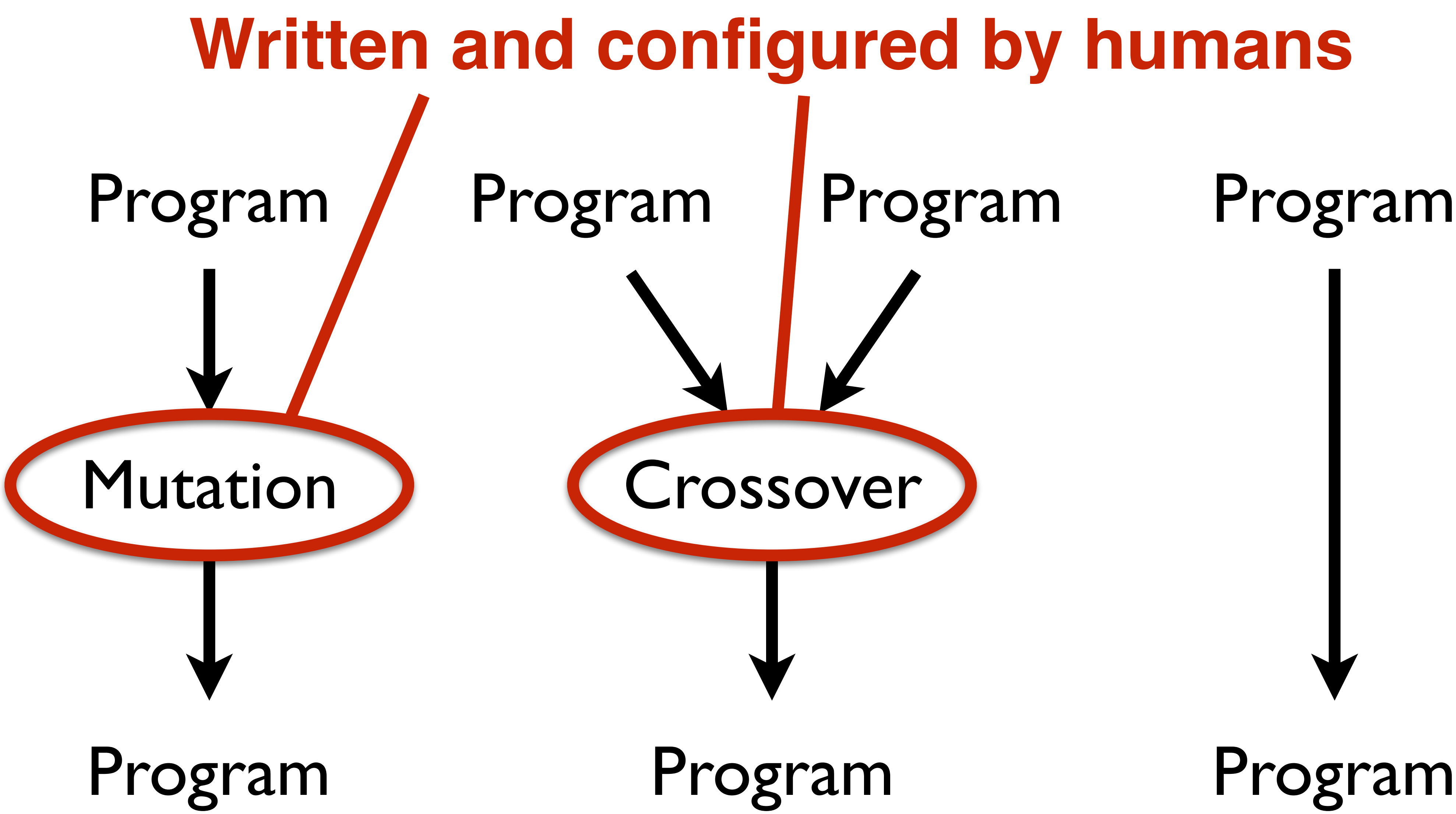
Outline

- Evolving code
- Language, variation, selection
- Evolving evolution
- Connections

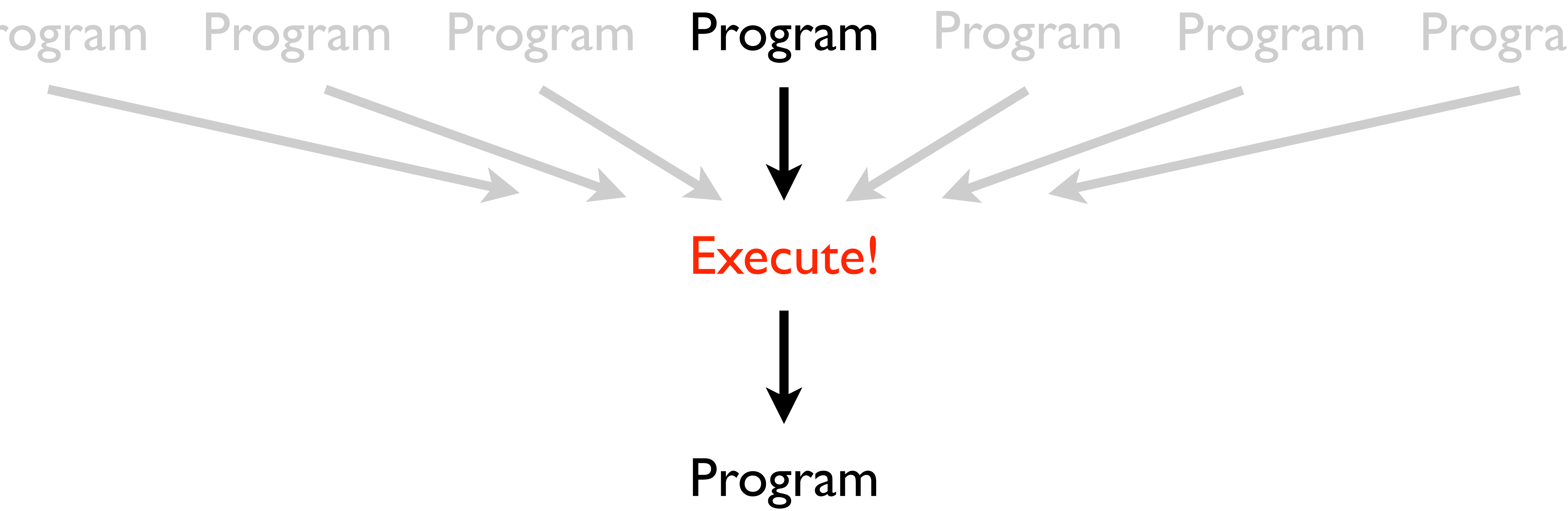
Variation


Program
Program
Program
Program
Mutation
Crossover
Program
Program
Program

Variation


Written and configured by humans
Program
Program
Program
Program
Mutation
Crossover
Program
Program
Program

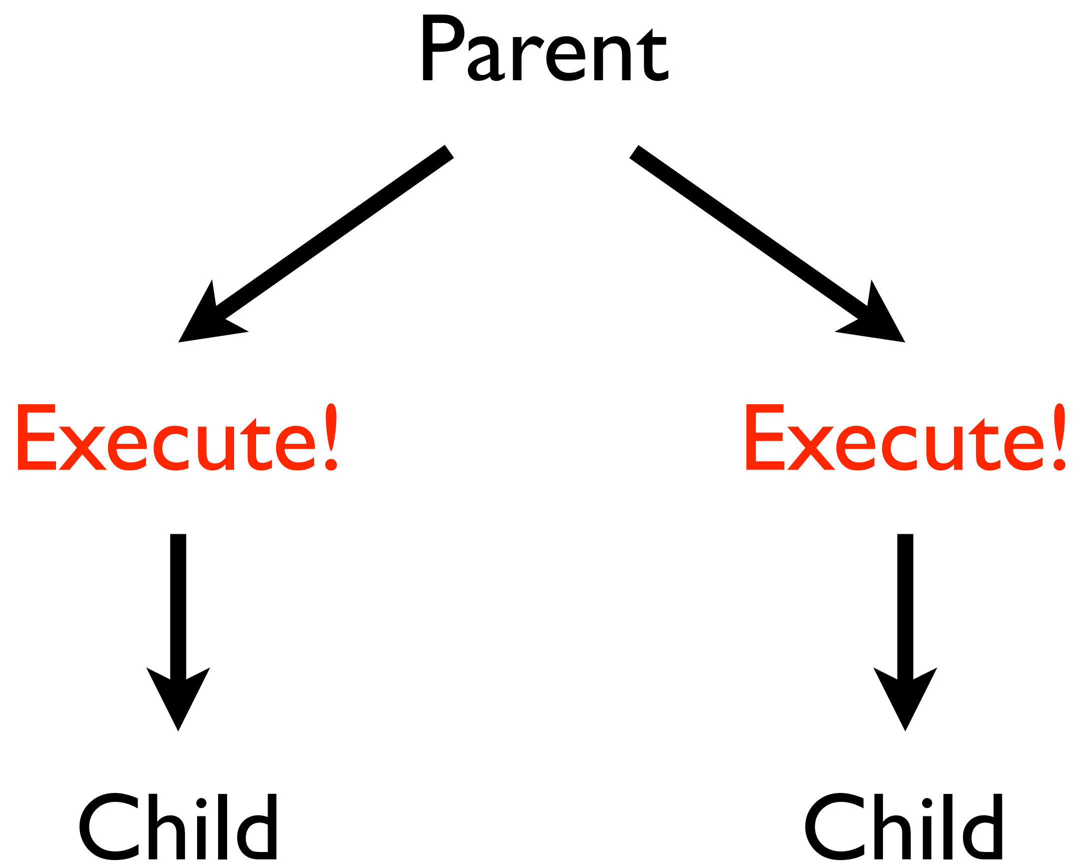
Autoconstruction



Autoconstructive Evolution

- Evolve evolution while evolving solutions
- Individuals produce and vary their own children, with methods that are subject to variation
- May produce EC systems more powerful than those we can write by hand

Diversification Constraints



Parent differs from both children, by different amounts

Synthesis Benchmarks

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, Checksum, Digits, Grade, Median, Smallest, Syllables

Solved with PushGP; first with autoconstruction

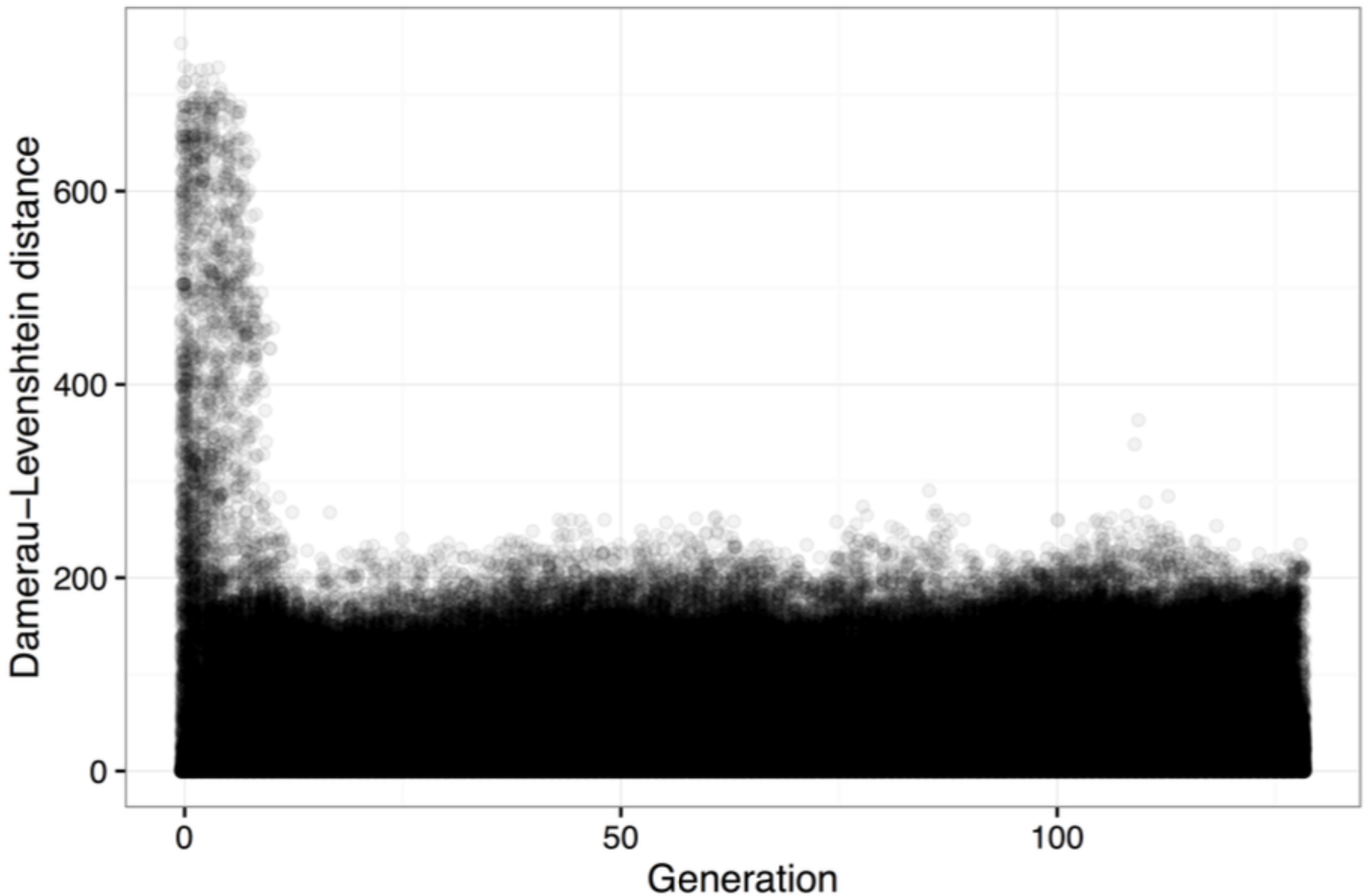


Figure 1: DL-distances between parent and child during a single non-autoconstructive run of GP on the Replace Space With Newline problem

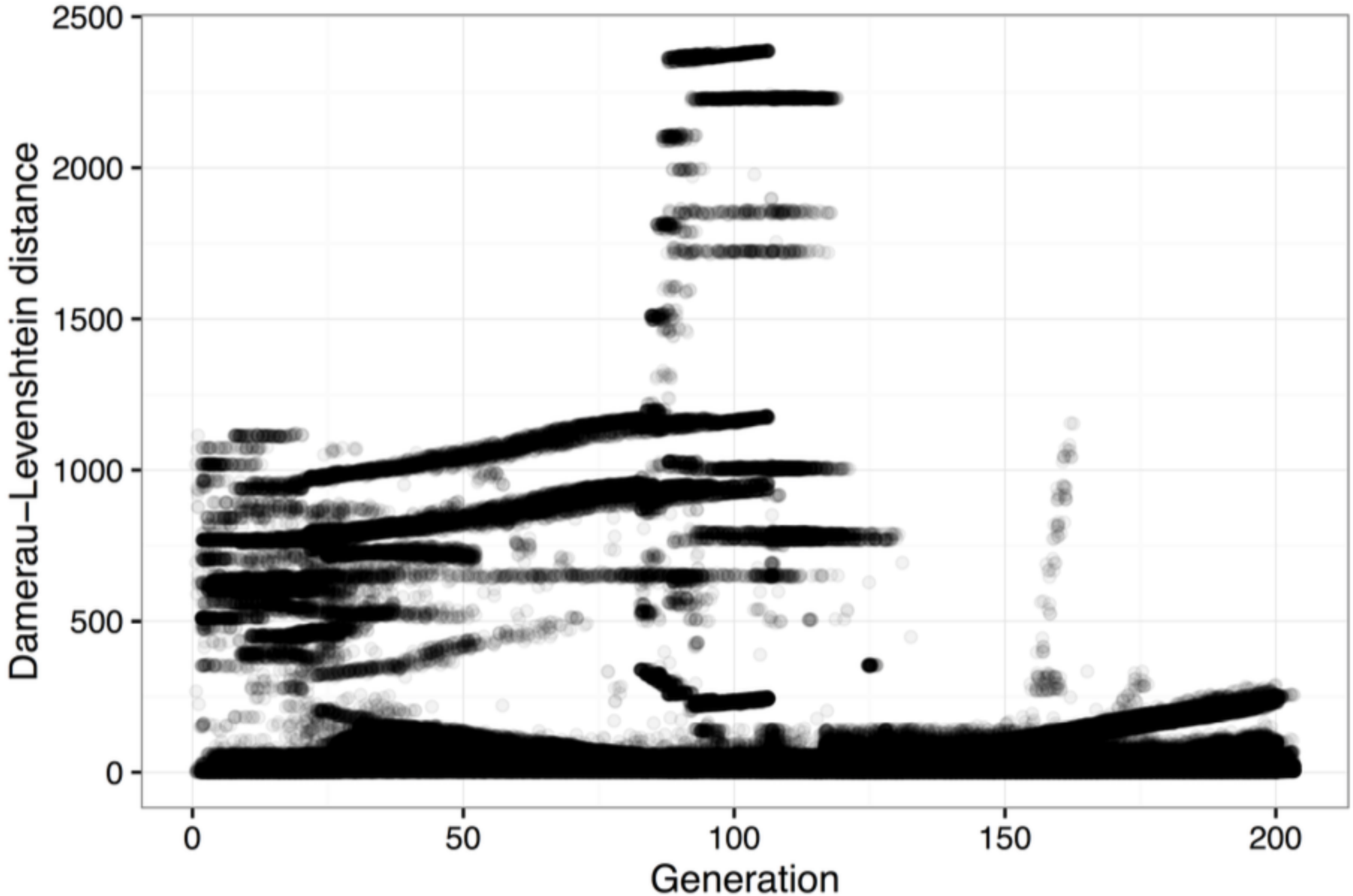


Figure 3: DL-distances between parent and child during a single autoconstructive run of GP on the Replace Space With Newline problem

Future

- Use autoconstruction to solve other previously unsolved problems
- Study how autoconstruction works, to improve it
- Consider implications for study of evolution of biological evolution

Outline

- Evolving code
- Language, variation, selection
- Evolving evolution
- Connections

Connections

- Machine learning
- Software engineering
- Programming languages
- Theory
- Evolutionary biology
- Applications

Takeaways

- Evolving code is fun and useful
- Push is a flexible and powerful representation for programs that evolve
- UMAD maximizes paths for evolution
- Lexicase selection: don't score; randomly sequence
- Evolving evolution is fun; may someday be useful

Thanks

- Nic McPhee, Tom Helmuth, Maggie M. Casale, and Julian Oks
- Members of the Hampshire College Computational Intelligence Lab
- Hampshire College for support for the Hampshire College Institute for Computational Intelligence
- This material is based upon work supported by the National Science Foundation under Grants No. 1617087, 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.