# Work in Progress on Autoconstructive Evolution

**The 6th International Conference**
**on Metaheuristics and Nature Inspired Computing (META'16)**

**Marrakech, Morocco**

**October 27-31 2016**


# Lee Spector

**Hampshire College & The University of Massachusetts, Amherst**

**Amherst, Massachusetts, USA**

**http://hampshire.edu/lspector**

# Outline

- What is autoconstructive evolution?
- Recent developments
- Results
- Prospects

**Awards**

Search Awards
Recent Awards
Presidential and Honorary Awards
About Awards

**How to Manage Your Award**

Grant Policy Manual
Grant General Conditions
Cooperative Agreement Conditions
Special Conditions
Federal Demonstration Partnership
Policy Office Website

**Award Abstract #1617087**

# RI: Small: RUI: Synthesis of Robust Artificial Systems by Adaptive Genetic Programming

| | |
|---|---|
| **NSF Org:** | IIS<br>Div Of Information & Intelligent Systems |
| **Initial Amendment Date:** | June 20, 2016 |
| **Latest Amendment Date:** | June 20, 2016 |
| **Award Number:** | 1617087 |
| **Award Instrument:** | Standard Grant |
| **Program Manager:** | Hector Munoz-Avila<br>IIS Div Of Information & Intelligent Systems<br>CSE Direct For Computer & Info Scie & Enginr |
| **Start Date:** | September 1, 2016 |
| **End Date:** | August 31, 2019 (Estimated) |
| **Awarded Amount to Date:** | $418,897.00 |
| **Investigator(s):** | Lee Spector lspector@hampshire.edu (Principal Investigator) |
| **Sponsor:** | Hampshire College<br>893 West Street<br>Amherst, MA 01002-3372 (413)559-5378 |
| **NSF Program(s):** | ROBUST INTELLIGENCE |

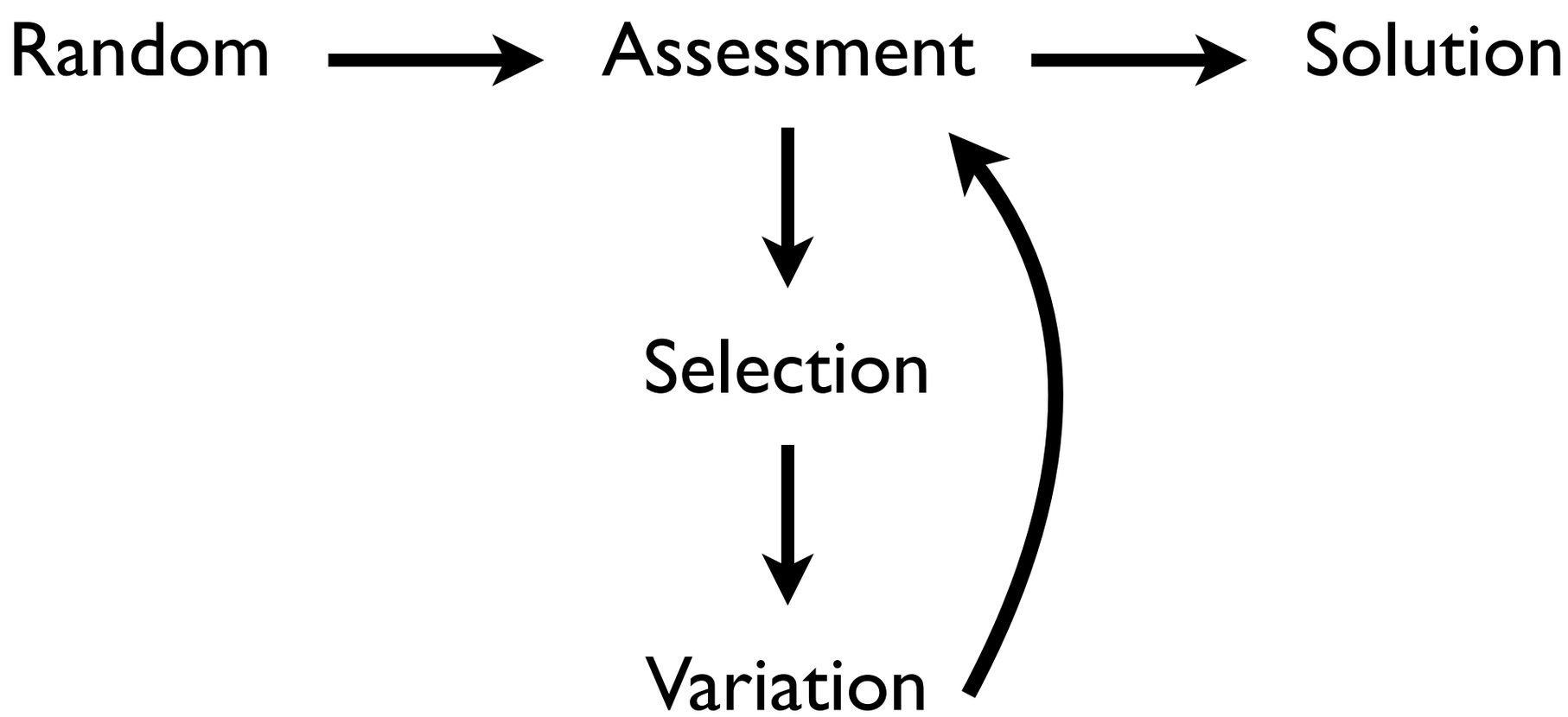
# Autoconstructive Evolution (1)

- Evolve evolution while evolving solutions
- How? Individuals produce and vary their own children, with methods that are subject to variation
- Requires understanding the evolution of variation
- Hope: May produce EC systems more powerful than we can write by hand

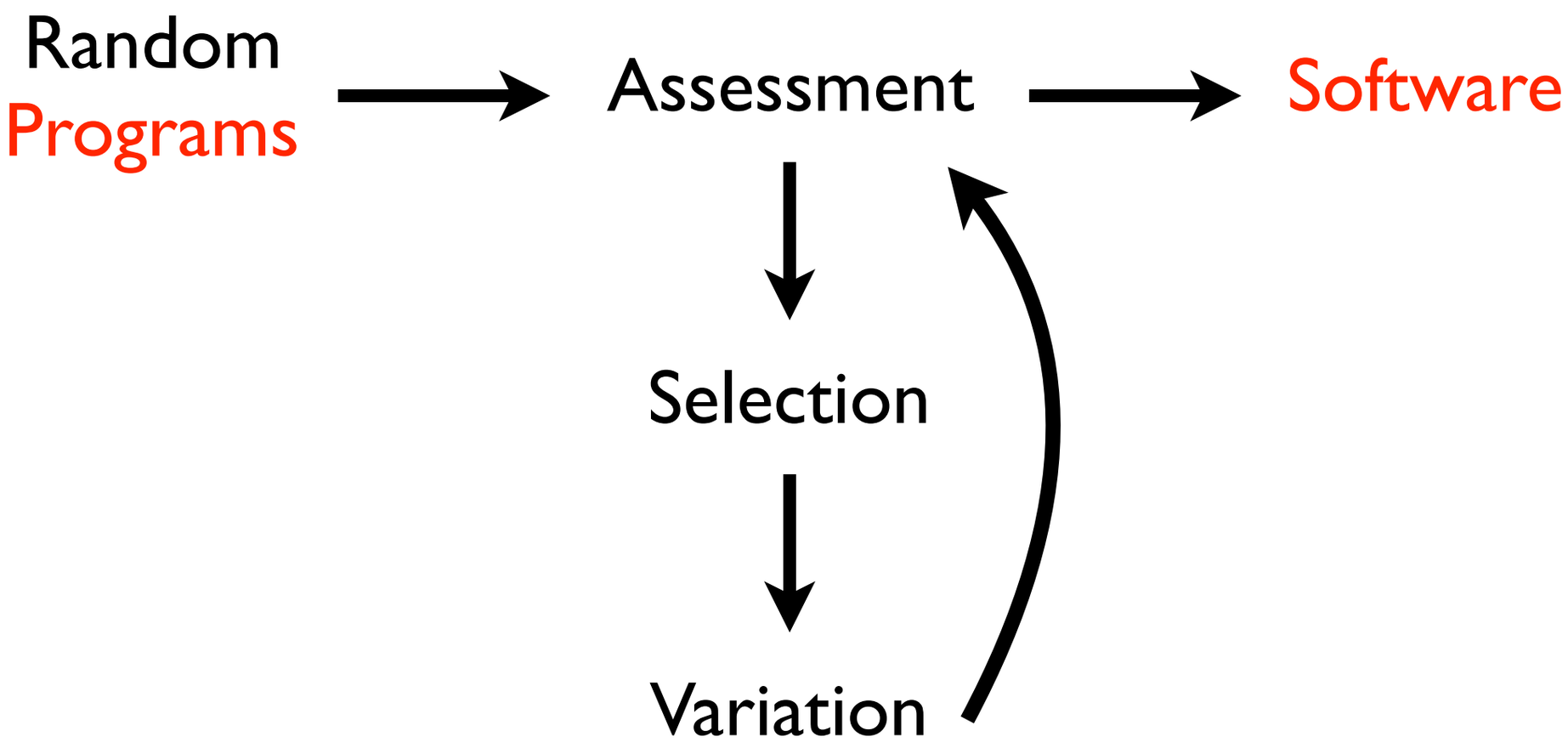
# Autoconstructive Evolution (2)

- A 15 year old project (building on older and broader-based ideas)
- Like genetic programming, but harder and less successful! But with greater potential?
- GECCO-2016: AutoDoG, sometimes solve significant problems, intriguing patterns of evolving evolution
- Recent: High-level DSL for genome manipulation

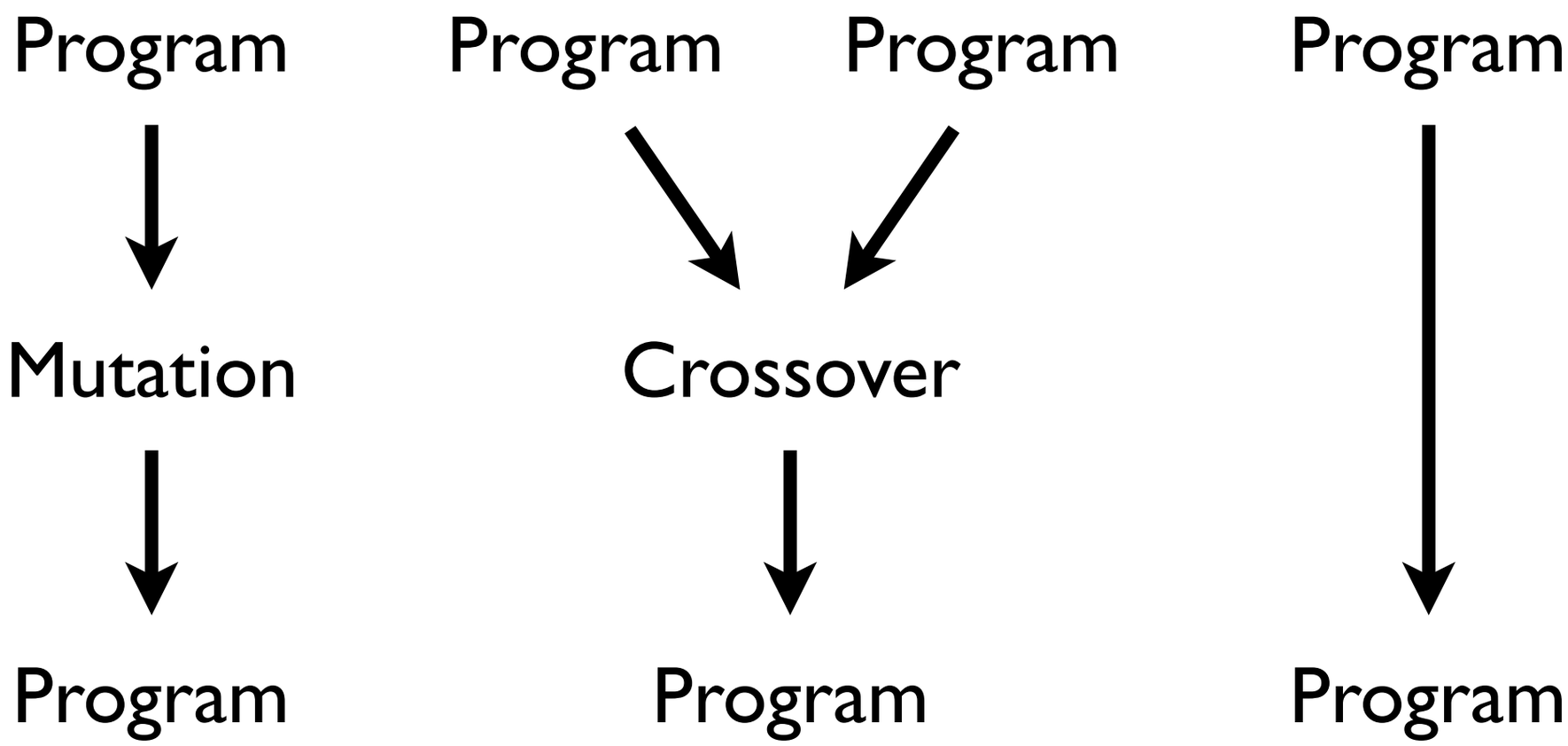
# Evolutionary Computing



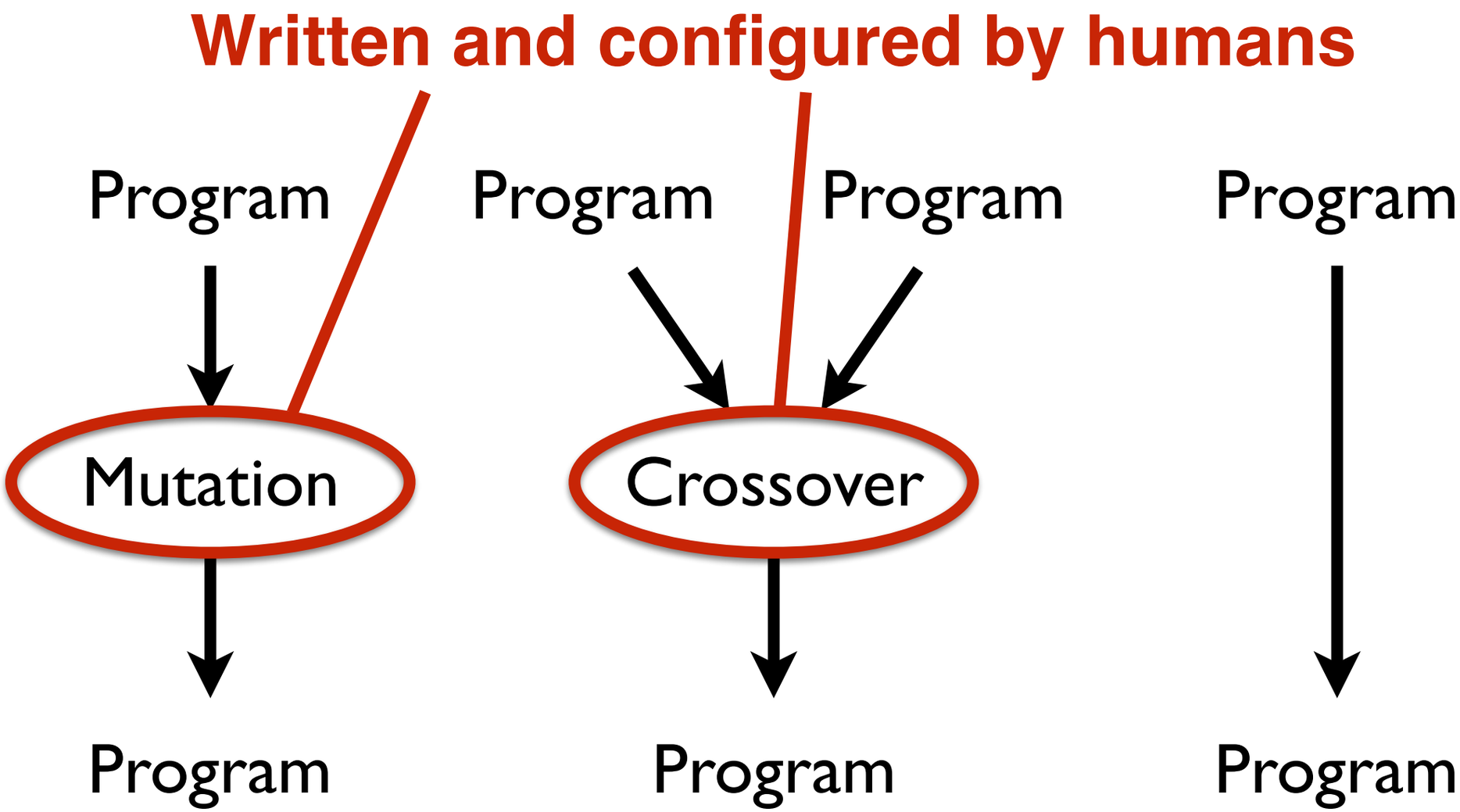
# Genetic Programming



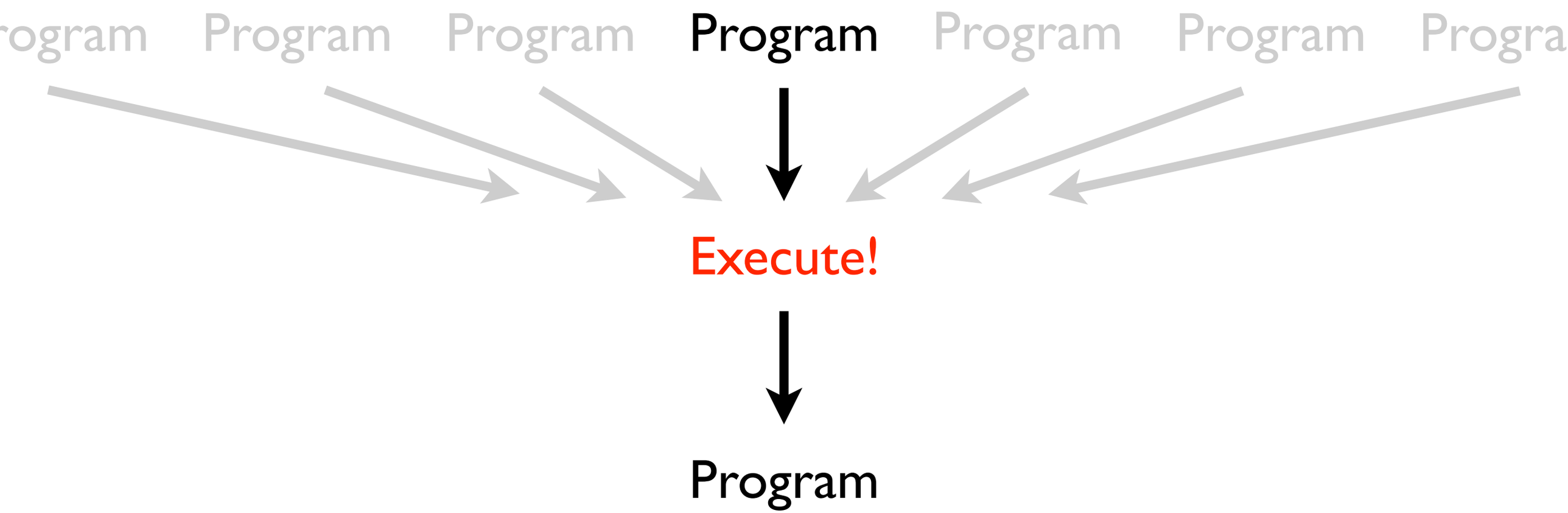
# Variation in Genetic Programming



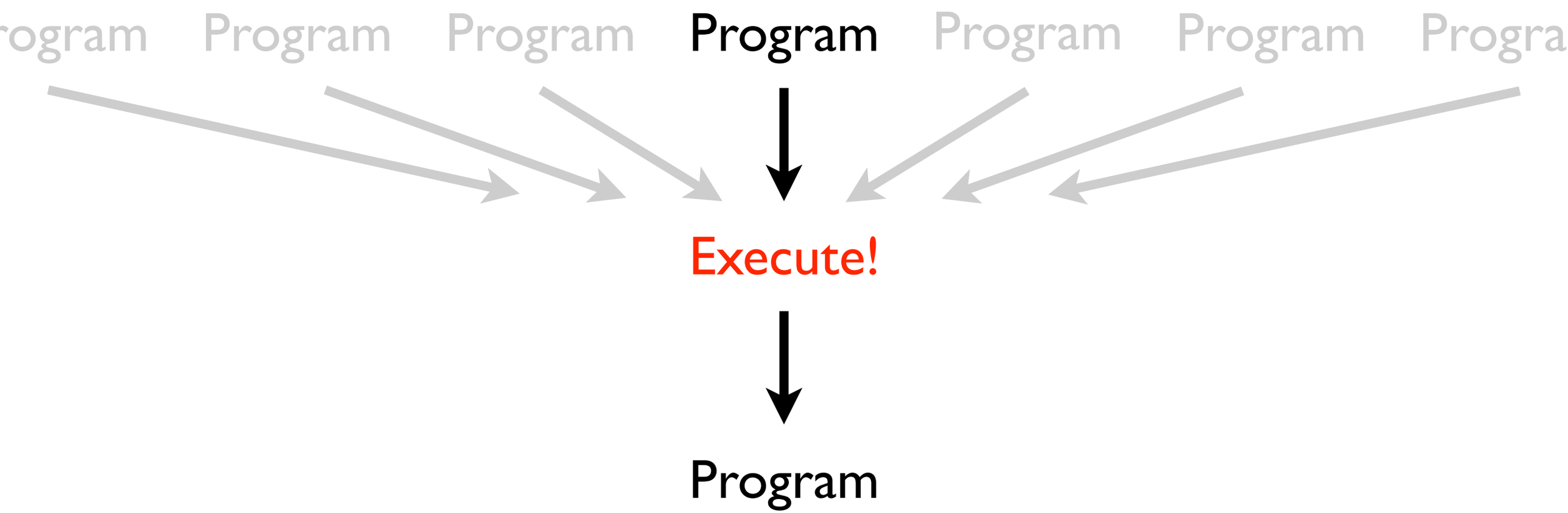
# Variation in Genetic Programming



# Autoconstruction



# Autoconstruction




*A bit more complicated when genomes distinguished from programs*

# Autoconstructive Evolution (3)

- Individual programs make their own children
- In doing so, they control their own mutation and recombination rates and methods, and in some cases mate selection, etc.
- The machinery of reproduction and diversification (i.e., the machinery of evolution) evolves
- In **Push**, experimentation with autoconstructive evolution is easy and natural

# Push

- A programming language for programs that evolve
- Data flows via per-type stacks, not syntax: integer, float, boolean, string, code, exec, vector, ...
- Trivial syntax, but rich data and control structures
- PushGP: GP system that evolves Push programs
- C++, **Clojure**, Common Lisp, Java, Javascript, Python, Racket, Ruby, Scala, Scheme, Swift
- http://pushlanguage.org

# Prior Work on Autoconstruction

- Demonstrated that selection can promote diversity
- Exhibited dynamics of diversification and adaptation
- Weak problem-solving power
- Difficult to analyze results, compare to ordinary genetic programming, or generalize

# AutoDoG (GECCO-2016)

**Auto**constructive **D**iversification **o**f **G**enomes

1. Construct genomes, not programs
2. Distinct mode/phase for construction of offspring
3. Select combinatorially, not on aggregate error
4. Enforce diversification constraints

# 1) What is Constructed?

- In prior work: Push programs, manipulated on code stacks using Lisp-inspired code-manipulation instructions
- In AutoDoG: Plush genomes, which are linear sequences of genes that specify instructions along with epigenetic markers that determine structure when Plush genomes are translated into Push programs, prior to running them

# Plush

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

- Linear genomes for Push programs
- Facilitates useful placement of code blocks
- Permits uniform linear genetic operators
- Allows for epigenetic hill-climbing

**Table 1: Genome instructions in AutoDoG**

| Instruction | Description |
|---|---|
| `close_dec` | Decrement close marker on a gene |
| `close_inc` | Increment close marker on a gene |
| `dup` | Duplicate top genome |
| `empty` | Boolean, is genome stack empty? |
| `eq` | Boolean, are top genomes equal? |
| `flush` | Empty genome stack |
| `gene_copy` | Copy gene from genome to genome |
| `gene_copy_range` | Copy genome segment |
| `gene_delete` | Remove gene |
| `gene_dup` | Duplicate gene |
| `gene_randomize` | Replace with random |
| `new` | Push empty genome |
| `parent1` | Push first parent's genome |
| `parent2` | Push second parent's genome |
| `pop` | Remove top genome |
| `rot` | Rotate top 3 genomes on stack |
| `rotate` | Rotate sequence of top genome |
| `shove` | Insert top genome deep in stack |
| `silence` | Add epigenetic silencing marker |
| `stackdepth` | Push integer depth of genome stack |
| `swap` | Exchange top two genomes |
| `toggle_silent` | Reverse silencing of a gene |
| `unsilence` | Remove epigenetic silencing marker |
| `yank` | Pull genome from deep in stack |
| `yankdup` | Copy genome from deep in stack |

# 2) When/how is it Constructed?

- In prior work: Various; sometimes during error testing, sometimes with problem inputs, sometimes with imposed but controllable variation

- In AutoDoG: Only within the autoconstruction genetic operator, entirely by the program itself

  - Construction: inputs are no-ops

  - Error testing: rand instructions produce constants

# 3) Who Constructs?

- In prior work: Parents selected using standard, error aggregating methods (tournament selection)

- In AutoDoG: Lexicase selection

# 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, and may or may not be particularly good on average, even though it may contribute to the evolution of generalists later

# Solving Uncompromising Problems with Lexicase Selection

Thomas Helmuth, Lee Spector *Member, IEEE,* James Matheson

***Abstract*—We describe a broad class of problems, called "uncompromising problems," characterized by the requirement that solutions must perform optimally on each of many test cases. Many of the problems that have long motivated genetic programming research, including the automation of many traditional programming tasks, are uncompromising. We describe and analyze the recently proposed "lexicase" parent selection algorition and show that it can facilitate the solution of uncompromising problems by genetic programming. Unlike most traditional parent selection techniques, lexicase selection does not base selection on a fitness value that is aggregated over all test cases; rather, it considers test cases one at a time in random order. We present results comparing lexicase selection to more traditional parent selection methods, including standard tournament selection and implicit fitness sharing, on four uncompromising problems: finding terms in finite algebras, designing digital multipliers, counting words in files, and performing symbolic regression of the factorial function. We provide evidence that lexicase selection maintains higher levels of population diversity than other selection methods, which may partially explain its utility as a parent selection algorithm in the context of uncompromising problems.**

***Index Terms*—parent selection, lexicase selection, tournament selection, genetic programming, PushGP.**

## I. Introduction

GENETIC programming problems generally involve test cases that are used to determine the performance of programs during evolution. While some classic genetic programming problems, such as the artificial ant problem and the

example, we can imagine
simulated wind turbine in
performance in low wind
performance in high wind c
optimize performance on al
and some sort of compro
Many common parent sele
ment selection, introduce c
aggregating the performan
cases into a single fitness
may be as simple as sum
squares, into a single error
as implicit fitness sharing
based on population statist

By contrast, we wish to
mising" problems: problem
must perform as well on
perform on that test case; t
is a problem for which i
to perform sub-optimally
for good performance on
problem defined by the se
programs in the search sp
produced by program $p_j$
lower error being better. T
program $p \in P$ would be c
if and only if $p(t_i) \leq p_i(t$

## Previous Results

| Problem | Lexicase | Tourney |
|---|---|---|
| Count Odds | 8 | 0 |
| Double Letters | 6 | 0 |
| Mirror Image | 78 | 46 |
| Negative To Zero | 45 | 10 |
| Replace Space with Newline | 51 | 8 |
| String Lengths Backwards | 66 | 7 |
| Syllables | 18 | 1 |
| Vector Average | 16 | 14 |
| X-Word Lines | 8 | 0 |

- 9 of 29 program synthesis benchmark problems
- Also higher levels of behavioral diversity

For floating-point problems, use Epsilon Lexicase Selection

***Consider for other applications involving multiple objectives***

# 4) Who Survives?

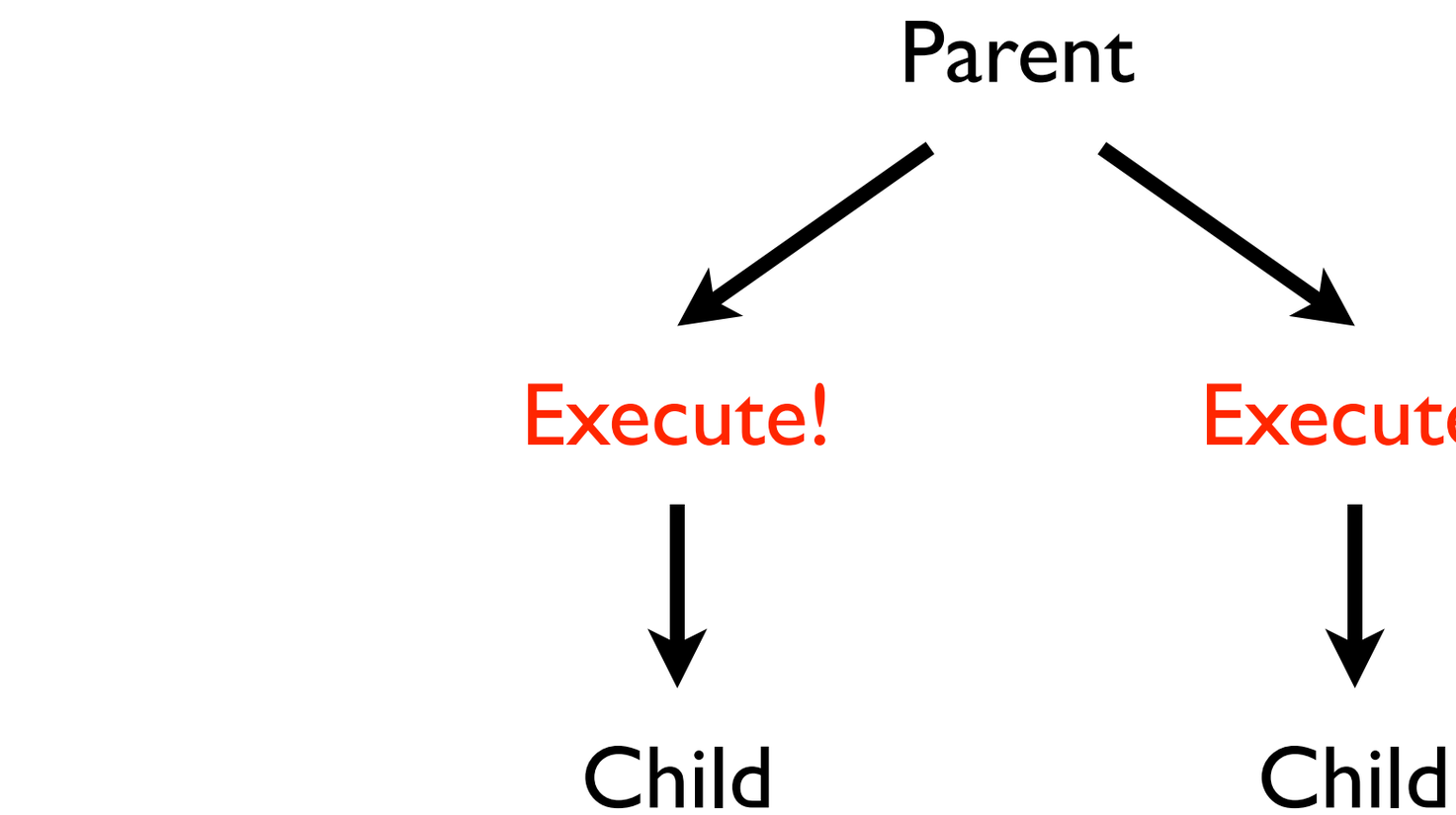
- In prior work: Sometimes everyone except clones, sometimes only those satisfying constraints on progress within lineages

- In AutoDoG: Only those satisfying diversification constraints on reproductive behavior, determined from a cascade of temporary descendants

# Diversification Constraint

- Applied to a cascade of temporary descendants
- Used here:
  - Children must differ from parent, differently
  - Applied to programs expressed by genomes
  - Enforced on a cascade with two children

# Diversification Constraint




Parent/child **program** differences positive; not the same

# Diversification Constraint

- Still under development
- How can you tell if an individual has the potential to produce diverse, adaptive descendants?
- Considering larger cascades, variation of:
  - genomes
  - reproductive behavior
  - problem solving behavior

# Needed for Evolution to Evolve

- Diversity: Individuals vary
- Diversification: Individuals produce descendants that vary, in various ways (used here)
- Recursive Variance: Individuals produce descendants that vary in the ways that they vary their offspring (under development)

# 29 Software 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

- PushGP has solved all of these except for the ones in blue

- Presented in a GECCO-2015 GP track paper

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.

# Replace Space With Newline

- Multiple types, looping, multiple tasks
- Simplified solution:

```
(\space char_dup exec_dup in1 \newline string_replacechar
print_string string_removechar string_length)
```

- PushGP can achieve success rates up to ~95%
- AutoDoG as described here succeeds 5-10%

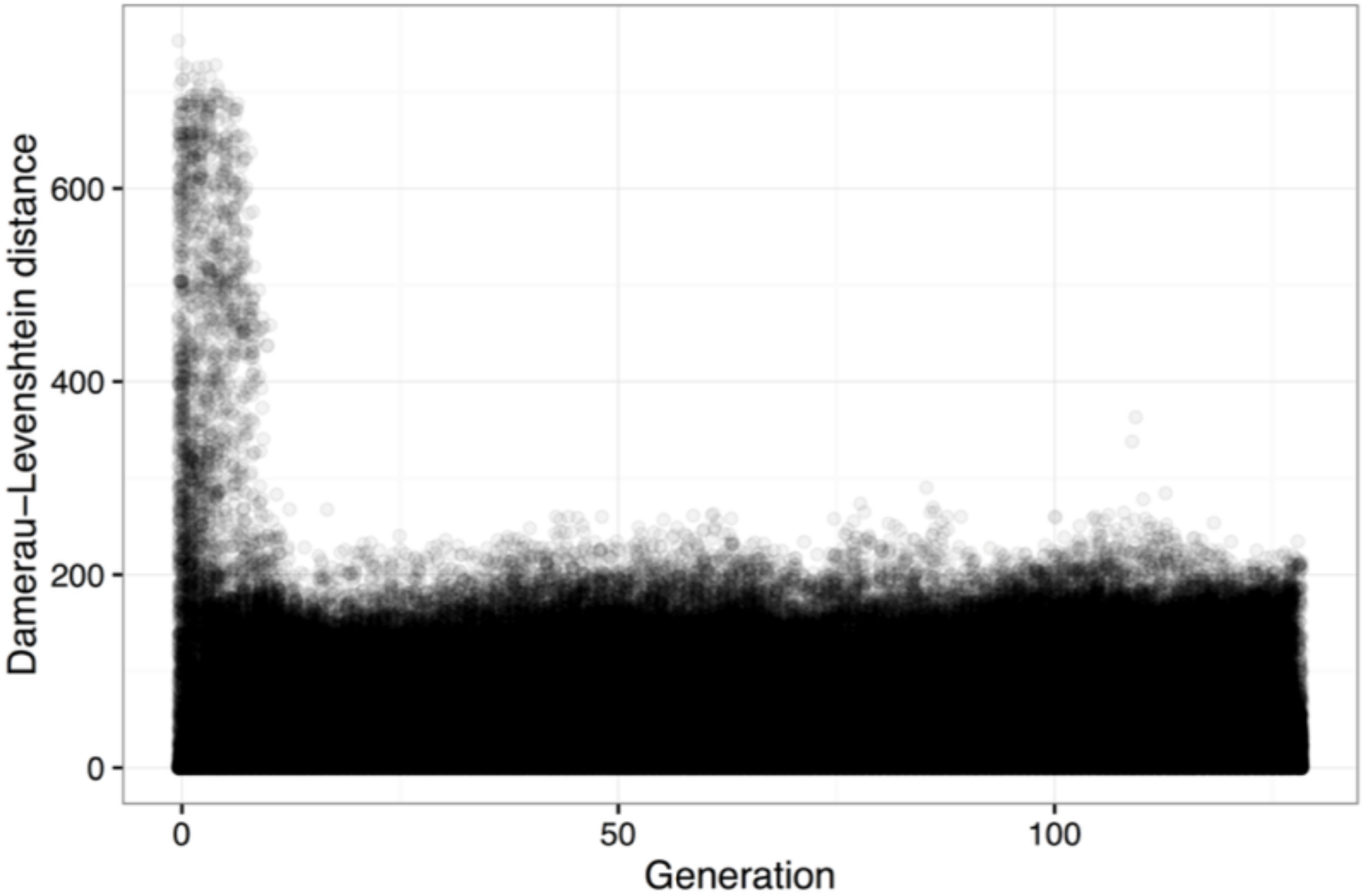



**Figure 1: DL-distances between parent and child during a single non-autoconstructive run of GP on the Replace Space With Newline problem**

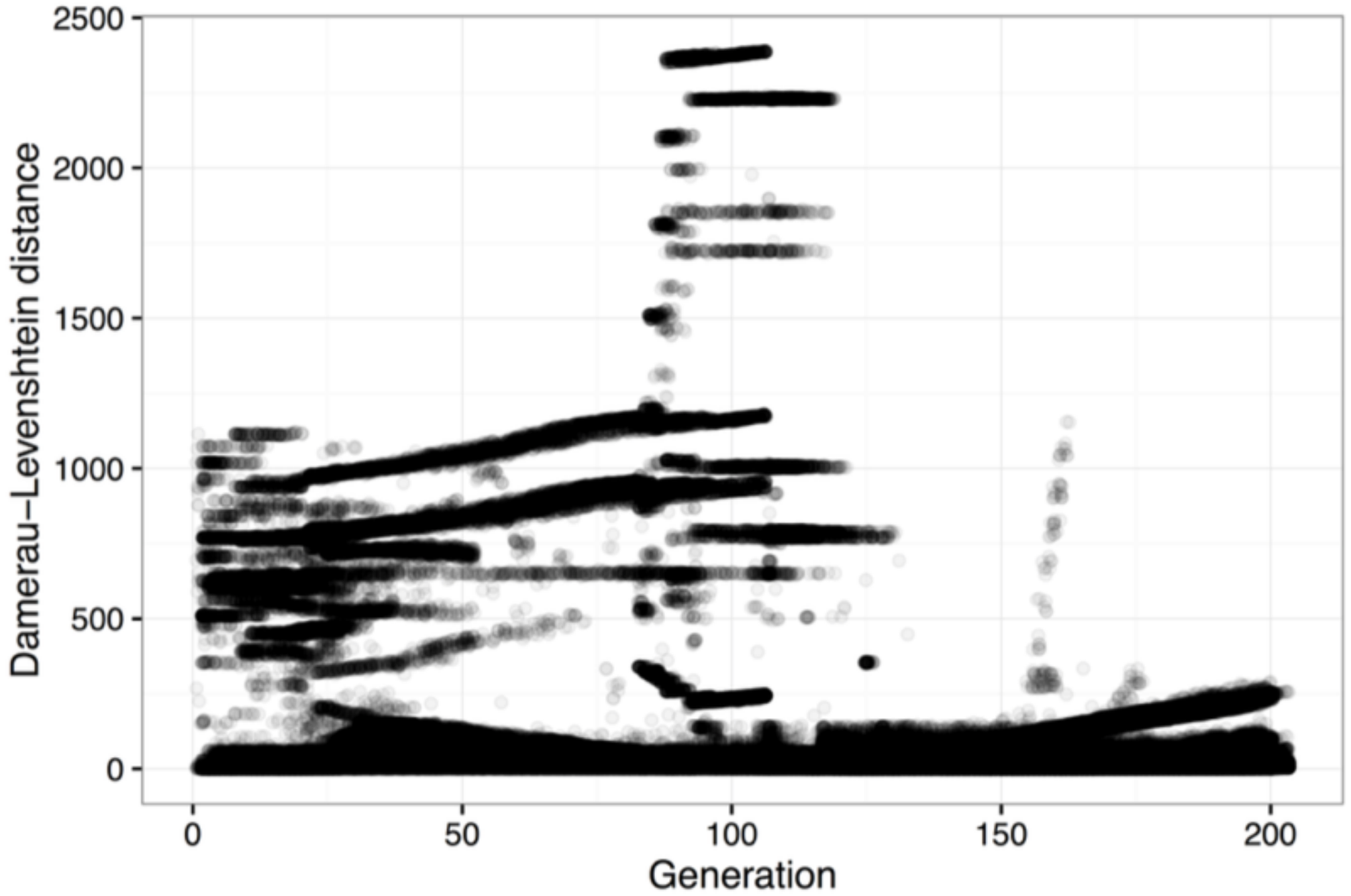



**Figure 3: DL-distances between parent and child during a single autoconstructive run of GP on the Replace Space With Newline problem**

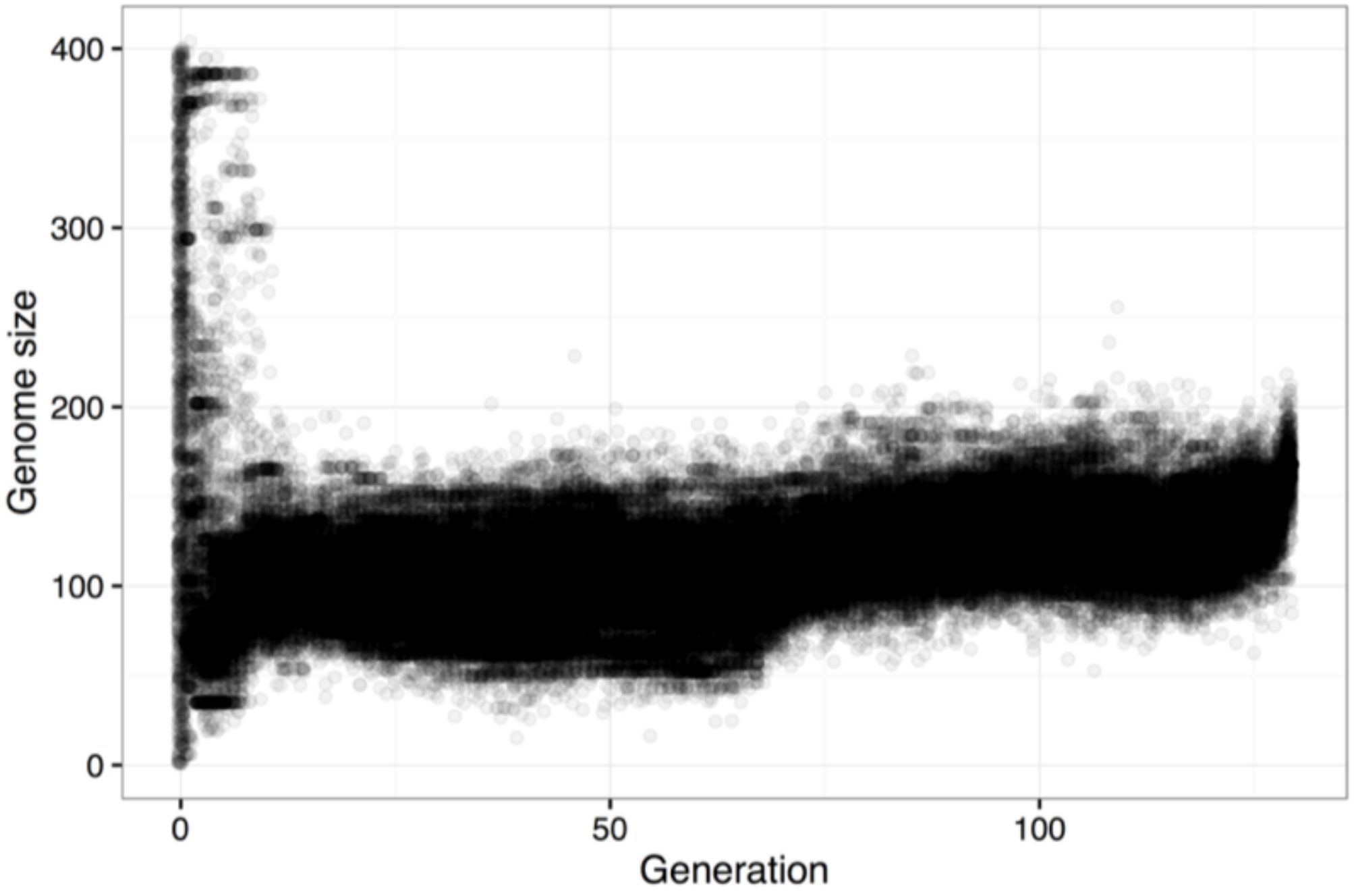



**Figure 2: Genome sizes during a single non-autoconstructive run of GP on the Replace Space With Newline problem**

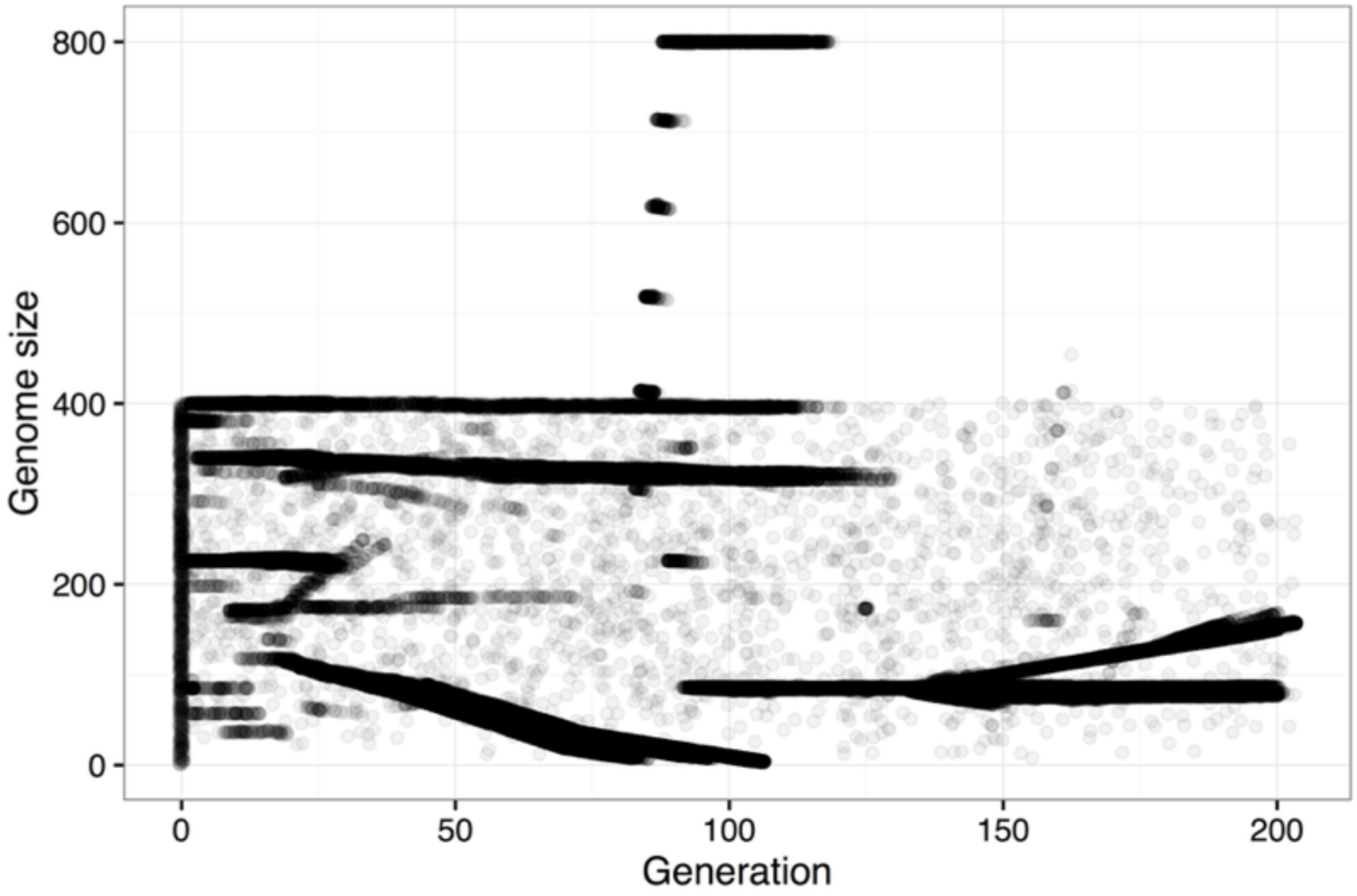
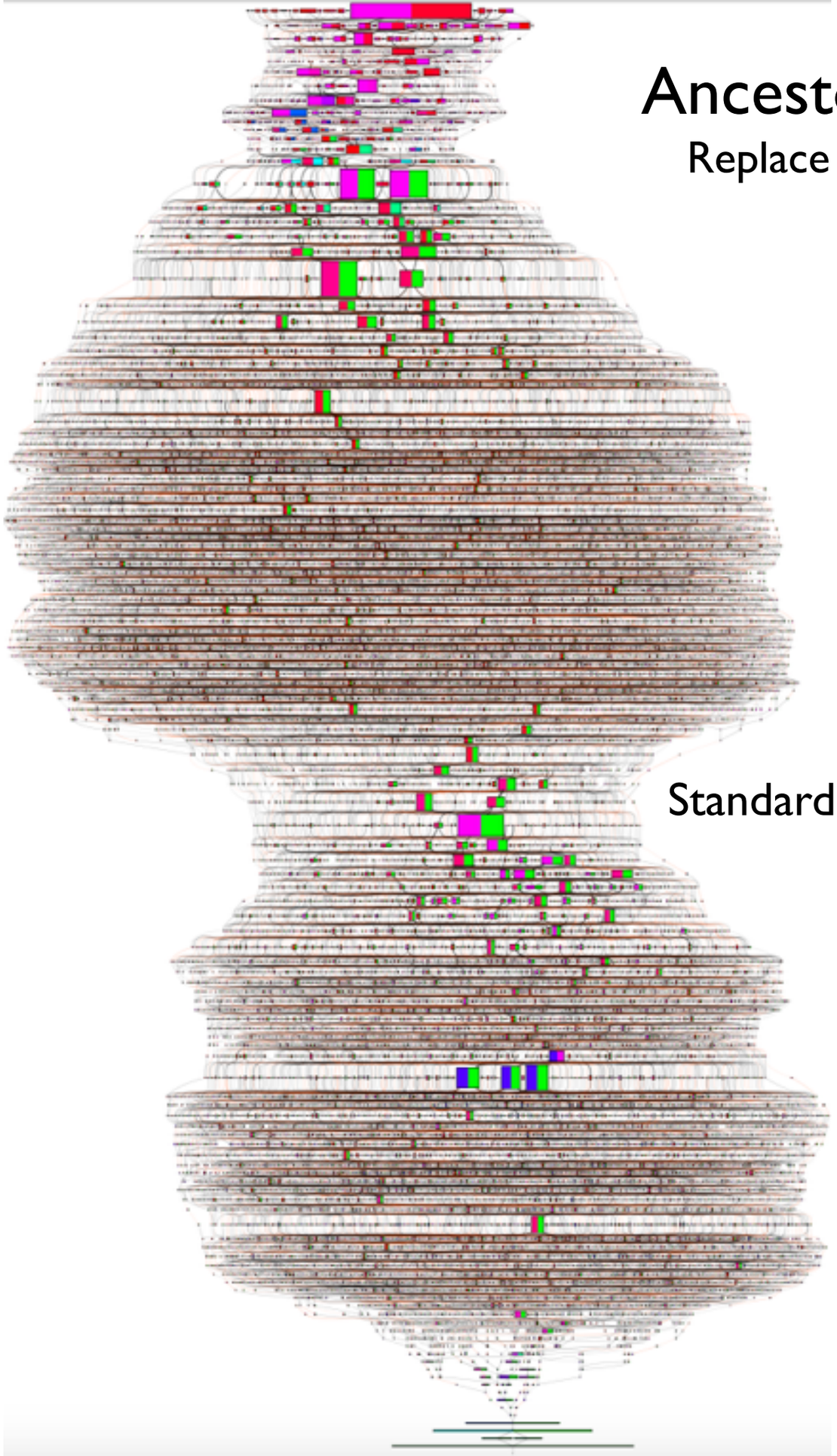


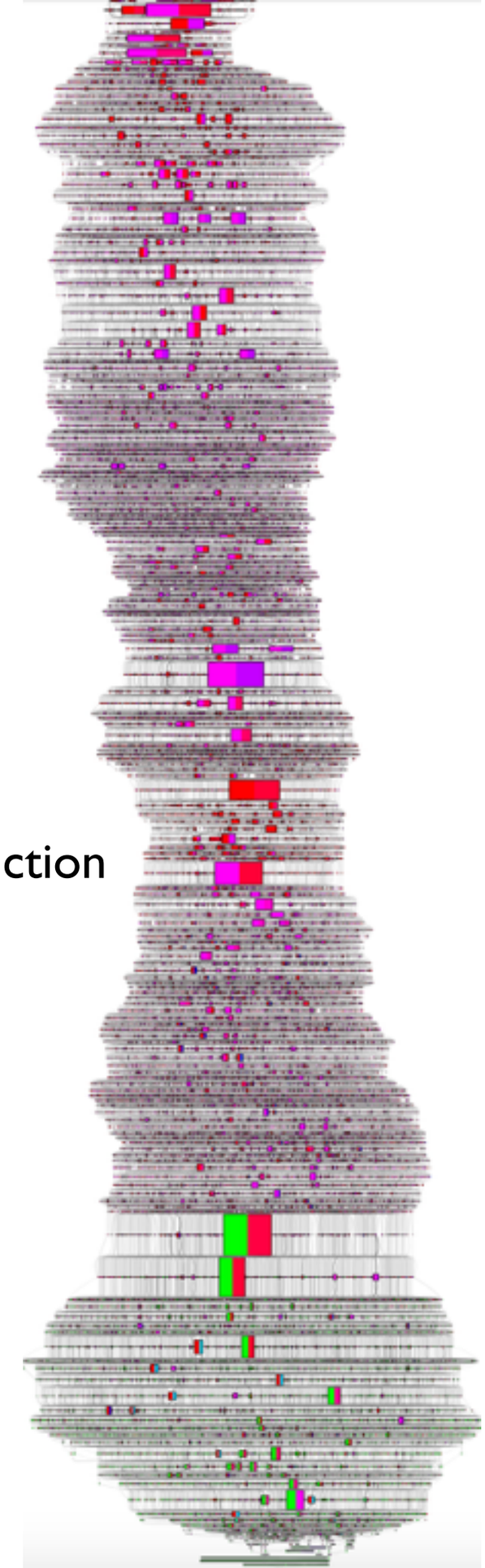

**Figure 4: Genome sizes during a single autoconstructive run of GP on the Replace Space With Newline problem**

# Ancestors of Solutions

## Replace Space with Newlines



Standard Operators



Autoconstruction

# High-level DSL for Genome Manipulation

- Parameterized calls to alternation (a generalization of uniform crossover) and several forms of uniform mutation

- Possibly: Evolve variation *method*, but adaptively control variation *amount*

# Conclusions and Prospects

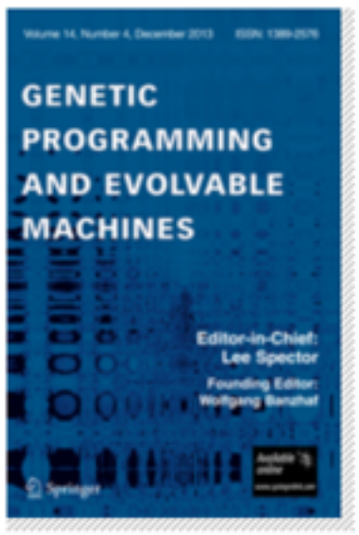
- Autoconstructive evolution can now solve reasonably hard problems, at least some of the time
- So far, it takes longer, because it must evolve evolution along with solutions
- Can it solve problems that can't be solved by ordinary genetic programming? Possibly, because it evolves
- Studying how/why it works may help us to improve it
- Studying how/why it works may help us to understand the evolution of biological evolution

# Thanks


- 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. 1129139, 1331283, and 1617087. 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.




# Genetic Programming and Evolvable Machines

Editor-in-Chief: Lee **Spector**

ISSN: 1389-2576 (print version)
ISSN: 1573-7632 (electronic version)

Journal no. 10710

Read Online

**63,02 €** Personal Rate e-only

Get Subscription

- Online subscription, valid from January through December of current calendar year
- Immediate access to this year's issues via SpringerLink
- 1 Volume(-s) with 4 issue(-s) per annual subscription
- Automatic annual renewal
- More information: >> FAQs // >> Policy





| Speed | Usage | Impact |
|---|---|---|