Genetic Programming with Push

Lee Spector
Hampshire College & UMass Amherst

This material is based upon work supported by the National Science Foundation under Grant No. 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.

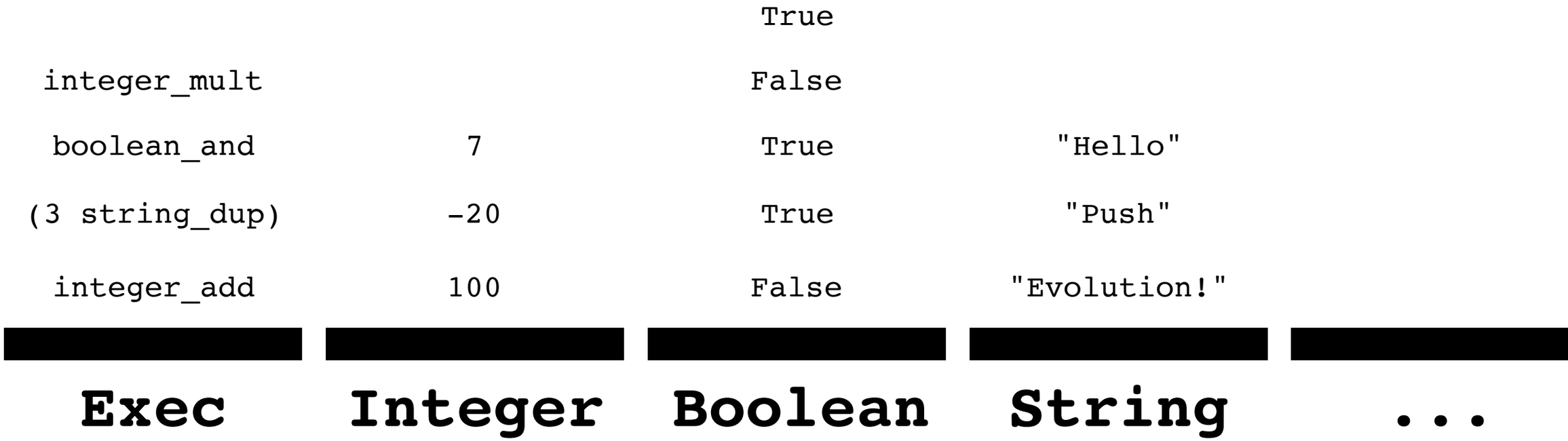
Outline

- The Push programming language
- Evolving Push programs
- Evolving Push program evolution

Push

- Programming language for programs that evolve
- Data flows via per-type stacks, not syntax
- Trivial syntax, rich data and control structures
- PushGP: GP system that evolves Push programs
- C++, Clojure, Common Lisp, Elixir, Java, Javascript, Python, Racket, Ruby, Scala, Scheme, Swift
- http://pushlanguage.org

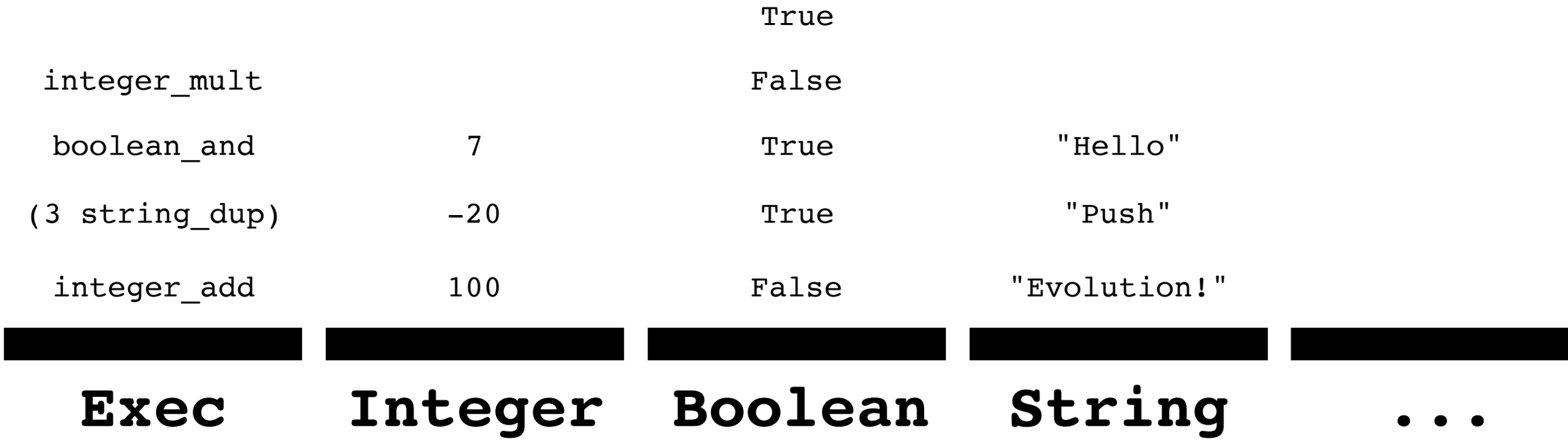
The Push VM


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

Push Execution

- Push the program onto the **exec** stack.
- While **exec** isn't empty and we haven't hit the step limit, pop and do the top:
  - If it's an instruction, execute it. (Insufficient arguments? Do nothing.)
  - If it's a literal, push it onto the appropriate stack.
  - If it's a list, push its elements back onto the **exec** stack one at a time.

The Push VM


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

```
(1 2 integer_add)
```

leaves 3 on the integer stack

```
(True False boolean_or boolean_not)
```

leaves False on the boolean stack

```
(3 5 integer_lte)
```

leaves True on the boolean stack

```
(3 5 integer_lte exec_if (1 "yes") (2 "no"))
```

leaves "yes" on string, 1 on integer

Clojush

Lee Spector (lspector@hampshire.edu), started 20100227 See version history. Older version history is in `old-version-history.txt`.

This is the README file accompanying Clojush, an implementation of the Push programming language and the PushGP genetic programming system in the Clojure programming language. Among other features this implementation takes advantage of Clojure's facilities for multi-core concurrency.

Availability

https://github.com/lspector/Clojush/

Requirements

To use this code you must have a Clojure programming environment; see http://clojure.org/. The current version of Clojush requires Clojure 1.7.0.

Clojure is available for most OS platforms. A good starting point for obtaining and using Clojure.

Quickstart

Using Leiningen you can run an example from the OS command line (in the Clojush directory) with a call like:

```
lein run clojush.problems.demos.simple-regression
```

```
;; https://github.com/lspector/Clojush/

=> (run-push '(1 2 integer_add) (make-push-state))

:exec ((1 2 integer_add))
:integer ()

:exec (1 2 integer_add)
:integer ()

:exec (2 integer_add)
:integer (1)

:exec (integer_add)
:integer (2 1)

:exec ()
:integer (3)
```

```
=> (run-push '(2 3 integer_mult 4.1 5.2 float_add
                 true false boolean_or)
               (make-push-state))

:exec ()
:integer (6)
:float (9.3)
:boolean (true)
```

In other words

- Put 2×3 on the integer stack
- Put $4.1 + 5.2$ on the float stack
- Put *true* $\vee$ *false* on the boolean stack

```
=> (run-push '(2 boolean_and 4.1 true integer_div
               false 3 5.2 boolean_or integer_mult
               float_add)
             (make-push-state))

:exec ()
:integer (6)
:float (9.3)
:boolean (true)
```

Same as before, but

- Several operations (e.g., `boolean_and`) become NOOPs
- Interleaved operations

```
=> (run-push
     '(4.0 exec_dup (3.13 float_mult) 10.0 float_div)
     (make-push-state))

:exec ((4.0 exec_dup (3.13 float_mult) 10.0 float_div))
:float ()

:exec (4.0 exec_dup (3.13 float_mult) 10.0 float_div)
:float ()

:exec (exec_dup (3.13 float_mult) 10.0 float_div)
:float (4.0)

:exec((3.13 float_mult) (3.13 float_mult) 10.0 float_div)
:float (4.0)

...

:exec ()
:float (3.91876)
```

Computes 4.0 **× 3.13 × 3.13** / 10.0

```
=> (run-push '(1 8 exec_do*range integer_mult)
             (make-push-state))

:integer (40320)
```

Computes 8! in a fairly “human” way

```
=> (run-push '(code_quote
                 (code_quote (integer_pop 1)
                             code_quote (code_dup integer_dup
                                         1 integer_sub code_do
                                         integer_mult)
                             integer_dup 2 integer_lt code_if)
                 code_dup
                 8
                 code_do)
               (make-push-state))

:code ((code_quote (integer_pop 1) code_quote (code_dup
       integer_dup 1 integer_sub code_do integer_mult)
       integer_dup 2 integer_lt code_if))
:integer (40320)
```

A less “obvious” recursive calculation of 8! achieved by code duplication

```
=> (run-push '(0 true exec_while
                  (1 integer_add true))
               (make-push-state))

:exec (1 integer_add true exec_while (1 integer_add
       true))
:integer (199)
:termination :abnormal
```

- An infinite loop
- Terminated by eval limit
- Result taken from appropriate stack(s) upon termination

```
=> (run-push '(in1 in1 float_mult 3.141592 float_mult)
             (push-item 2.5 :input (make-push-state)))

:float (19.63495)
:input (2.5)
```

Computes the area of a circle with the given radius: 3.141592 × in1 × in1

For Most Types

- <type>_dup
- <type>_empty
- <type>_eq
- <type>_flush
- <type>_pop
- <type>_rot
- <type>_shove
- <type>_stackdepth
- <type>_swap
- <type>_yank
- <type>_yankdup

Selected Integer Instructions

integer_add integer_dec integer_div integer_gt integer_fromstring integer_min integer_mult integer_rand

Selected Boolean Instructions

boolean_and boolean_xor boolean_frominteger

Selected String Instructions

string_concat string_contains string_length string_removechar string_replacechar

Exec (selected)

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

More

code_atom code_car print_newline integer_sub integer_inc boolean_stackdepth return_exec_pop vector_integer_eq autoconstructive_integer_rand boolean_pop genome_close_inc string_fromchar vector_string_shove zip_yankdup genome_new vector_float_yankdup exec_yankdup vector_integer_shove integer_yankdup string_flush boolean_swap zip_empty exec_shove vector_boolean_yank code_eq exec_y boolean_yank integer_eq genome_silence string_butlast code_contains string_conjchar code_do*count vector_float_last genome_pop string_substring integer_mult code_length vector_integer_dup boolean_or code_position boolean_empty zip_fromcode print_vector_string vector_boolean_swap return_frominteger vector_float_pushall char_iswhitespace code_cdr exec_do*vector_integer integer_rand vector_string_replacefirst string_first vector_boolean_first exec_do*while exec_string_iterate string_indexofchar vector_float_replace integer_fromstring code_list code_swap char_frominteger genome_gene_randomize vector_integer_emptyvector vector_string_eq vector_float_butlast exec_empty zip_end? exec_fromzipnode string_shove vector_boolean_pushall zip_insert_left_fromcode exec_rot vector_string_concat vector_float_indexof code_pop vector_string_subvec vector_integer_swap code_subst char_pop return_string_pop zip_yank exec_dup vector_integer_butlast vector_float_rest vector_string_flush boolean_fromfloat code_fromziprights float_sin boolean_flush char_isdigit float_lte exec_fromziproot vector_integer_empty print_code vector_string_stackdepth string_reverse exec_k vector_integer_yank float_frominteger char_rot print_char vector_integer_stackdepth vector_boolean_concat boolean_xor integer_gte genome_yankdup vector_float_shove vector_integer_take code_quote string_replacefirst return_fromstring exec_fromziplefts vector_integer_yankdup boolean_shove float_lt vector_string_dup vector_string_occurrencesof vector_integer_replace zip_branch? vector_float_reverse float_mod vector_float_subvec string_last print_boolean boolean_rot vector_string_rest integer_div vector_float_remove integer_fromfloat integer_lte code_fromzipchildren environment_end vector_integer_rot integer_mod string_concat vector_string_butlast genome_swap code_null exec_do*count vector_float_emptyvector vector_string_yankdup integer_rot float_yankdup vector_string_rot zip_replace_fromexec vector_string_take integer_add vector_integer_occurrencesof integer_shove genome_dup return_code_pop char_swap integer_max return_fromexec code_wrap return_float_pop code_flush genome_yank zip_shove vector_integer_flush vector_integer_subvec vector_boolean_indexof vector_float_pop vector_string_remove vector_integer_contains zip_remove code_append vector_float_eq vector_integer_conj string_eq zip_leftmost code_yankdup code_rot integer_stackdepth float_max vector_boolean_set zip_append_child_fromexec zip_next vector_float_conj zip_fromexec string_take zip_left zip_replace_fromcode char_stackdepth return_fromchar genome_eq vector_integer_replacefirst float_stackdepth code_fromziproot float_fromchar float_gt boolean_dup float_fromboolean code_fromzipnode genome_rot vector_float_replacefirst vector_boolean_conj vector_boolean_dup vector_integer_indexof vector_string_swap exec_eq string_emptystring string_swap integer_yank exec_while float_empty print_vector_boolean integer_min exec_swap genome_rotate integer_fromchar vector_string_yank string_stackdepth code_do*range string_replacechar char_allfromstring vector_integer_rest vector_boolean_length char_yank vector_float_empty code_fromfloat genome_parent2 return_fromcode string_pop float_eq vector_boolean_empty zip_insert_child_fromexec vector_string_last string_nth code_do* return_zip_pop vector_string_pop zip_rot vector_integer_nth exec_do*range exec_if char_shove zip_down zip_insert_left_fromexec code_frominteger vector_boolean_remove vector_integer_remove boolean_invert_first_then_and genome_flush print_string integer_fromboolean char_yankdup code_do vector_string_first boolean_frominteger string_setchar vector_integer_last char_isletter genome_gene_dup vector_integer_concat print_integer code_map boolean_eq float_gte return_fromfloat genome_gene_copy string_occurrencesofchar string_replacefirstchar print_float boolean_rand integer_flush float_shove string_replace char_dup float_pop char_eq vector_float_nth vector_string_conj integer_gt return_integer_pop float_sub vector_integer_length vector_float_set vector_string_indexof vector_boolean_rest code_dup vector_boolean_shove zip_eq float_min boolean_not float_mult float_fromstring genome_unsilence code_if vector_integer_pop vector_boolean_last exec_do*times zip_pop zip_rightmost float_dec vector_float_contains genome_gene_copy_range environment_new exec_do*vector_string code_nthcdr string_empty char_empty exec_pop vector_integer_set autoconstructive_boolean_rand vector_float_rot string_yankdup exec_do*vector_float string_removechar code_extract vector_string_replace vector_float_first genome_parent1 return_tagspace char_flush vector_float_occurrencesof vector_string_emptyvector float_add code_stackdepth exec_s zip_insert_right_fromexec float_dup vector_string_nth zip_stackdepth vector_integer_reverse print_vector_integer char_fromfloat code_do*times code_noop zip_swap code_yank integer_lt vector_boolean_eq genome_stackdepth code_fromziplefts noop_open_paren string_containschar string_yank char_rand zip_flush vector_boolean_rot float_swap exec_fromziprights vector_string_pushall vector_string_set vector_boolean_flush exec_noop code_size vector_boolean_stackdepth vector_integer_pushall vector_boolean_reverse integer_swap string_split vector_boolean_contains string_fromboolean return_boolean_pop vector_float_dup vector_boolean_replace integer_dup vector_boolean_nth vector_string_length string_rest zip_insert_child_fromcode float_tan string_rot string_rand exec_yank string_parse_to_chars integer_pop integer_empty vector_float_flush vector_float_yank noop_delete_prev_paren_pair print_exec zip_append_child_fromcode genome_gene_delete code_empty float_inc zip_right vector_float_length float_rand integer_dec string_contains return_fromboolean vector_float_concat vector_float_stackdepth exec_do*vector_boolean vector_integer_first genome_shove code_rand print_vector_float float_rot return_char_pop vector_string_contains vector_boolean_occurrencesof genome_empty zip_prev genome_toggle_silent vector_string_reverse zip_dup code_cons code_member exec_stackdepth float_flush boolean_and vector_boolean_butlast string_length float_cos string_frominteger exec_flush vector_string_empty exec_when vector_float_swap genome_close_dec code_insert vector_boolean_pop float_div zip_insert_right_fromcode code_fromboolean vector_boolean_take code_shove environment_begin vector_float_take boolean_invert_second_then_and code_container code_nth vector_boolean_subvec float_yank zip_up vector_boolean_emptyvector vector_boolean_replacefirst string_fromfloat vector_boolean_yankdup string_dup boolean_yankdup exec_fromzipchildren

Auto-Simplification

- Loop:
  - Make it randomly simpler
  - Keep simpler if as good or better; otherwise revert

- GECCO-2014 poster: efficiently and reliably reduces the size of the evolved programs

- GECCO-2014 student paper: used as genetic operator

- GECCO-2017 GP best paper nominee: 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Total error: 0.0

Size: 231

Auto-simplifying with starting size: 231

...

step: 5000

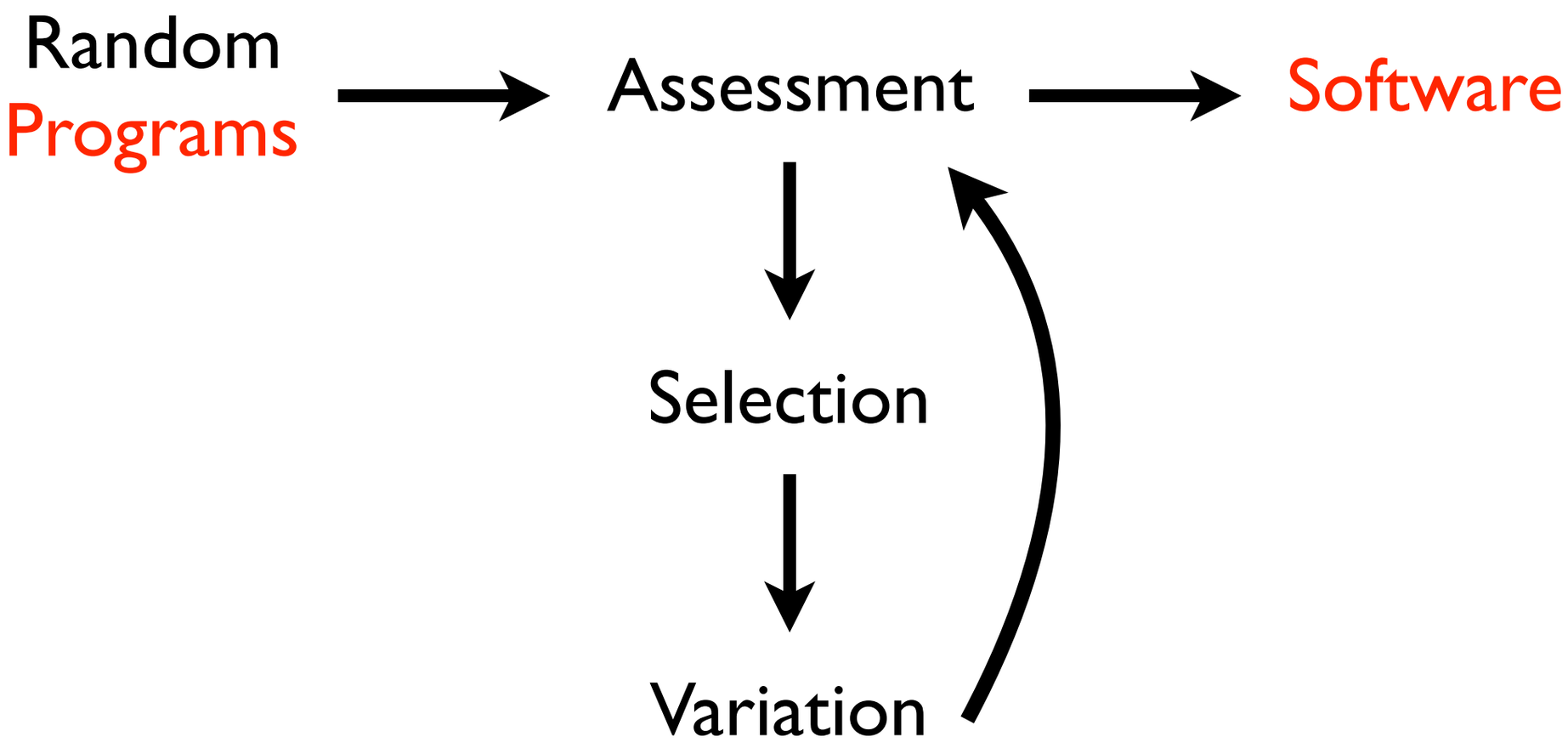
program: (\space \newline in1 string_replacechar print_string "Wx{ " string_last in1 string_removechar string_length)

errors: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Total error: 0.0

Size: 11

Genetic Programming



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

```
(pushgp
  {:error-function
   (fn [{:keys [program] :as individual}]
     (assoc individual
       :errors
       (vec
         (for [input (mapv float (range 10))]
           (let [output (->> (make-push-state)
                             (push-item input :input)
                             (run-push program)
                             (top-item :float))]
             (if (number? output)
               (Math/abs (float (- output
                                   (- (* input
                                         input
                                         input)
                                      (* 2 input input)
                                      input))))
               1000000))))))
   :atom-generators (list 'in1
                          'float_div
                          'float_mult
                          'float_add
                          'float_sub)})
```

This repository Search Pull requests Issues Marketplace Explore

lspector / propel
forked from thelmuth/propel

Unwatch 1 Star 0 Fork 1

Code Pull requests 0 Projects 0 Wiki Insights Settings

Lee Spector's Plushy fork of Tom Helmuth's little PushGP implementation in Clojure. Edit

Add topics

24 commits 1 branch 0 releases 2 contributors EPL-1.0

Branch: master New pull request Create new file Upload files Find file Clone or download

This branch is 18 commits ahead of thelmuth:master. Pull request Compare

lspector Tweak text Latest commit 8d81266 6 days ago

doc	First commit. propel-gp is working	9 months ago
src/propel	Add exec_if and handle boolean literals	6 days ago
test/propel	First commit. propel-gp is working	9 months ago
.gitignore	Add worksheet; make interpreter handle non-flat programs	7 days ago
CHANGELOG.md	First commit. propel-gp is working	9 months ago
LICENSE	First commit. propel-gp is working	9 months ago
README.md	Tweak text	6 days ago
project.clj	Update version	7 days ago
project.clj~	Add Gorilla REPL dependency	16 days ago
worksheet.clj	Reformat worksheet	6 days ago

README.md

propel

Lee Spector's Plushy fork of Tom Helmuth's little PushGP implementation in Clojure.

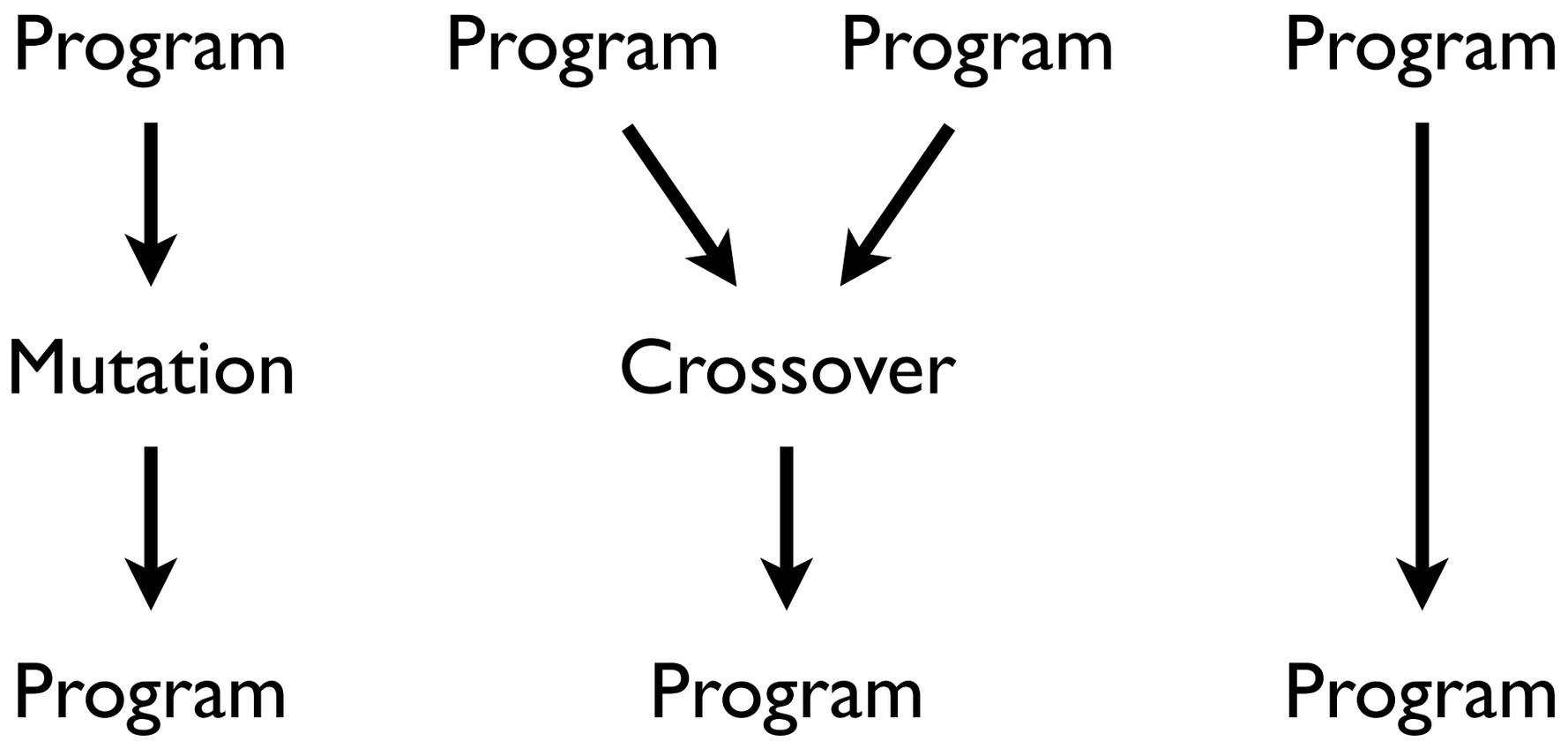
Usage

Propel

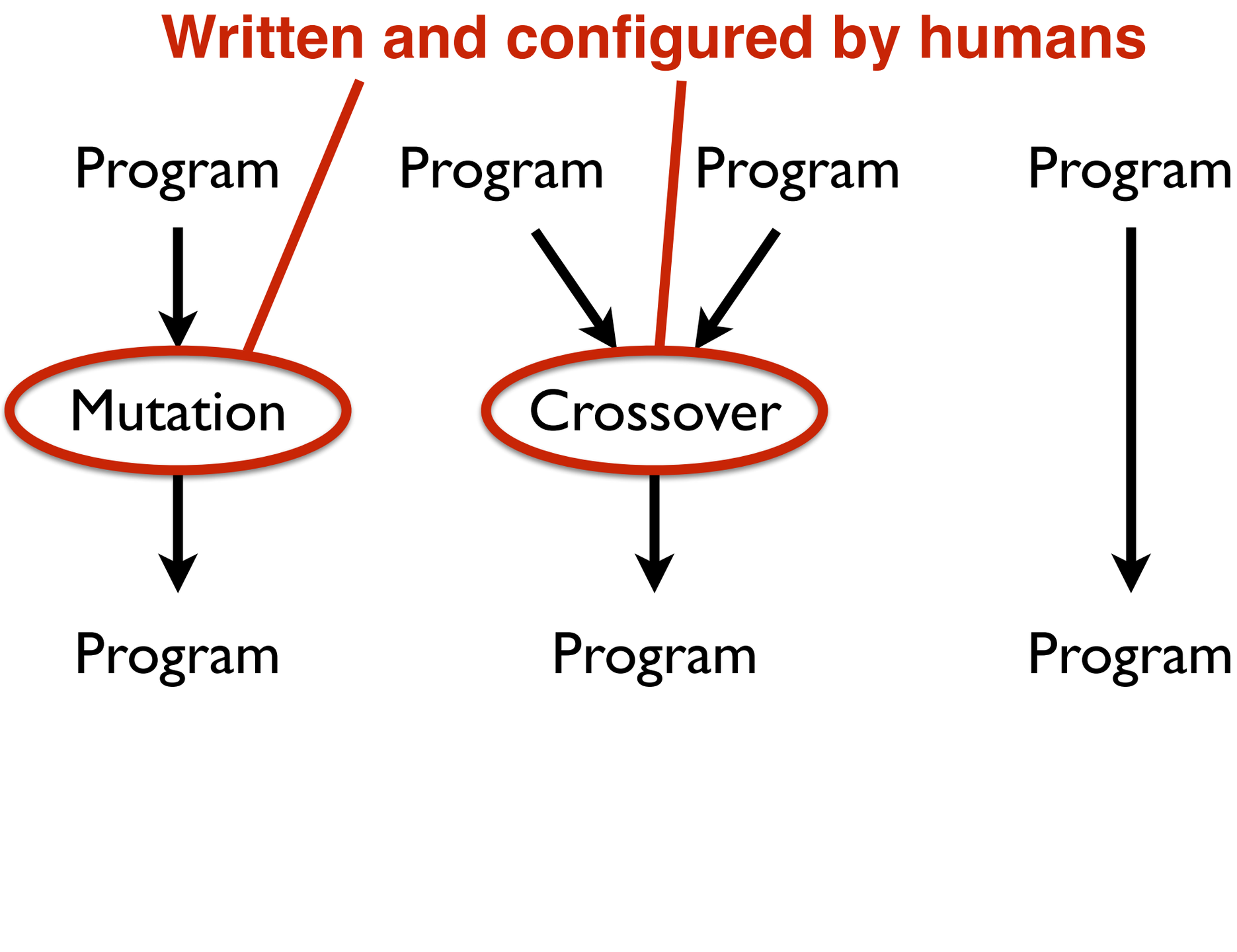
- Use "close" instructions instead of epigenetic markers: an individual is represented as a "Plushy" rather than as a Plush genome

- <code & demo>

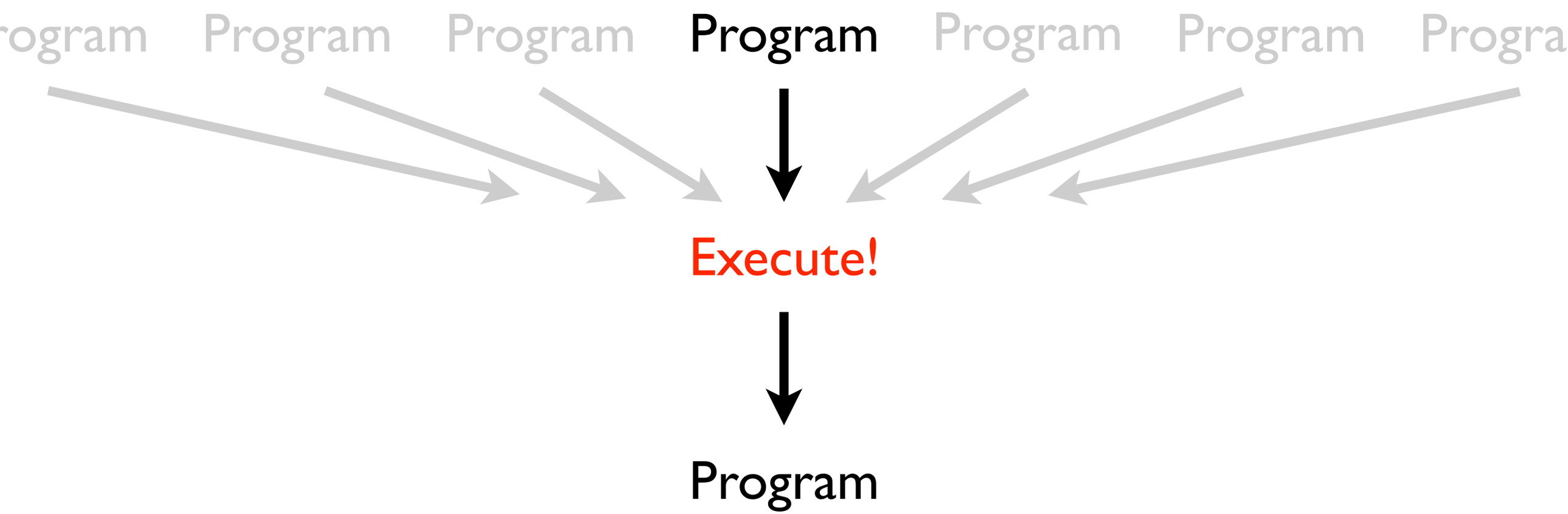
Variation in GP



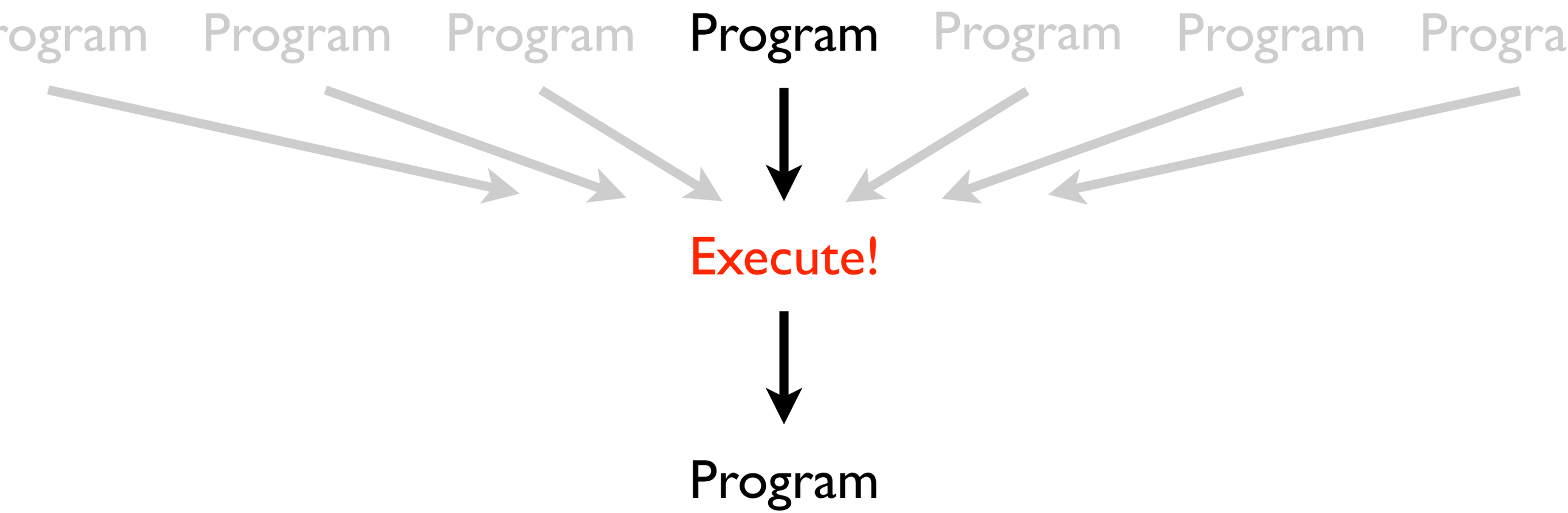
Variation in GP



Autoconstruction



Autoconstruction



A bit more complicated when genomes distinguished from programs

Autoconstruction

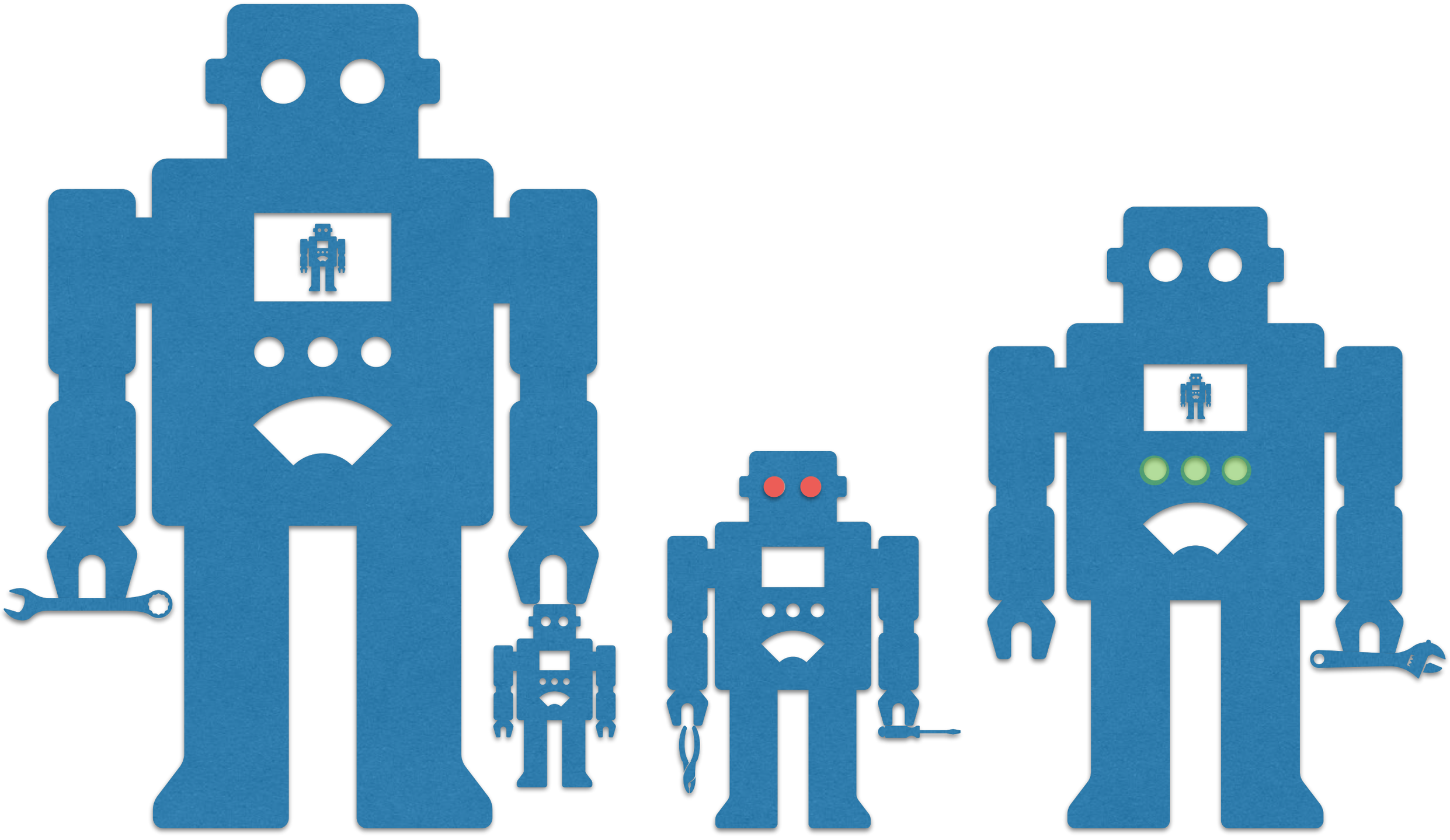
- Evolve evolution while evolving solutions
- How? Individuals produce and vary their own children, with methods that are subject to variation
- Requires understanding the evolution of variation
- Hope: May produce EC systems more powerful than we can write by hand

Autoconstruction

- A 15 year old project (building on older and broader-based ideas)

- Like genetic programming, but harder and less successful! But with greater potential?

- Recent versions sometimes solve significant problems, intriguing patterns of evolving evolution



For Evolution^2

- Diversity: Individuals vary
- Diversification: Individuals produce descendants that vary, in various ways
- Recursive Variance: Individuals produce descendants that vary in the ways that they vary their offspring

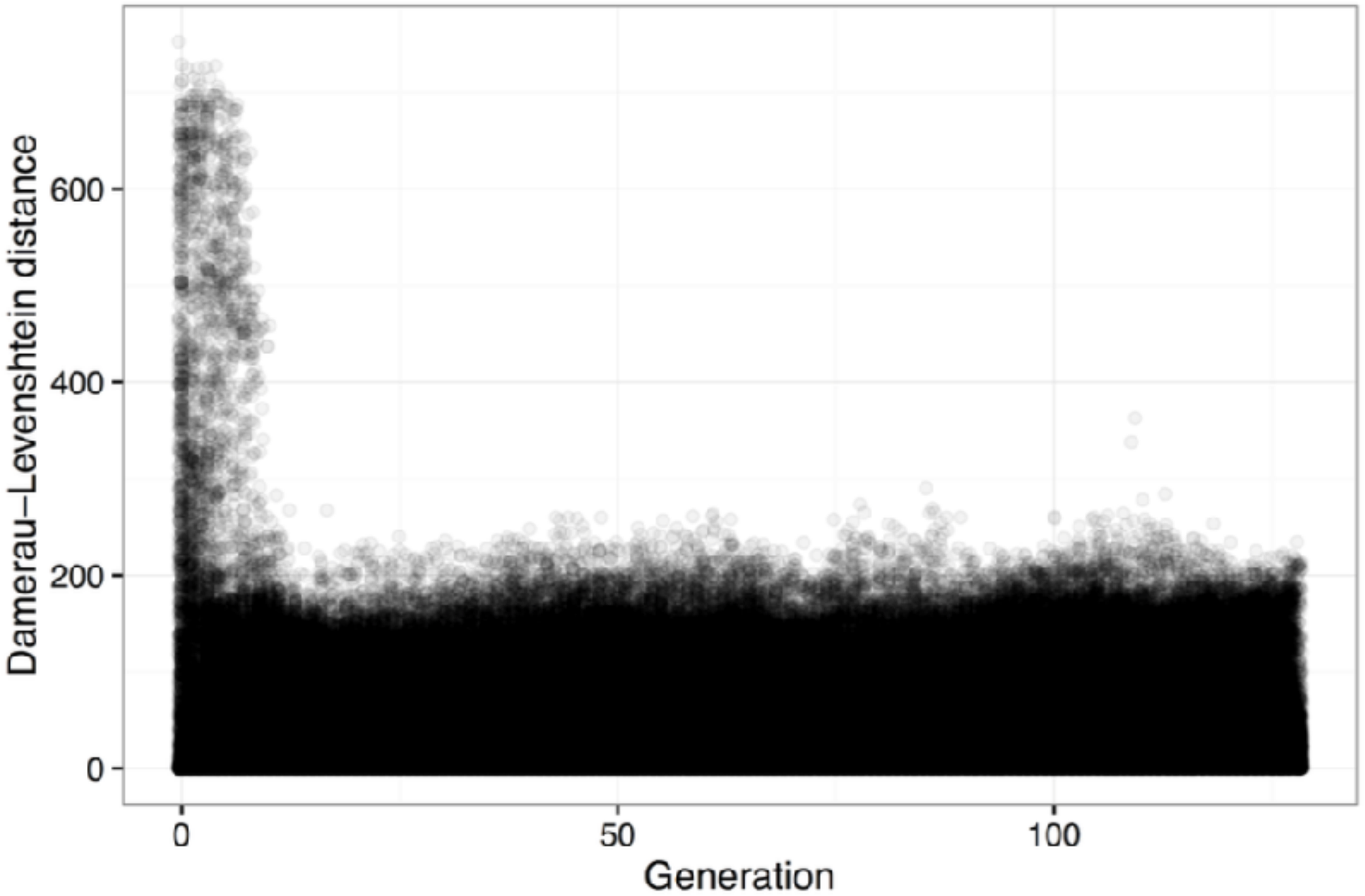


Figure 1: DL-distances between parent and child during a single non-autoconstructive run of GP on the Replace Space With Newline problem

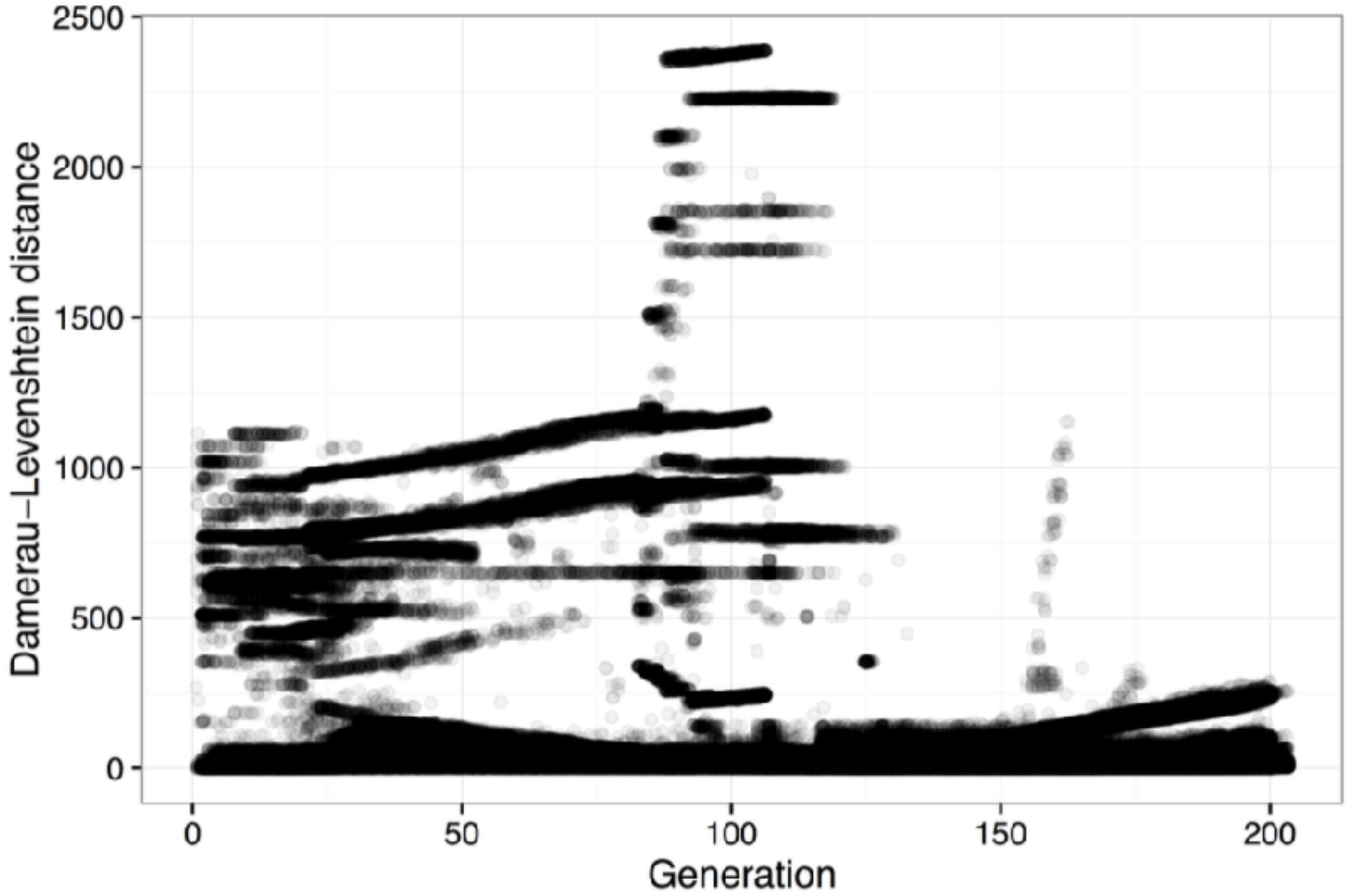


Figure 3: DL-distances between parent and child during a single autoconstructive run of GP on the Replace Space With Newline problem

Rivaling GP

- Autoconstructive evolution can succeed as much and as fast as non-autoconstructive evolution
- In 20 runs in one configuration, 75% success within 300 generations on Replace Space With Newline (100% by generation 628)
- Surprising!

Extending GP's reach

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
```

- Autoconstruction found solutions before ordinary GP

Prospects

- Genetic programming is already solving important, hard problems
- If it can be applied to itself, to evolve as it runs, then it will be able to solve harder problems
- We are beginning to see how this might work
- Help would be appreciated!