Evolution of Expressive Programs

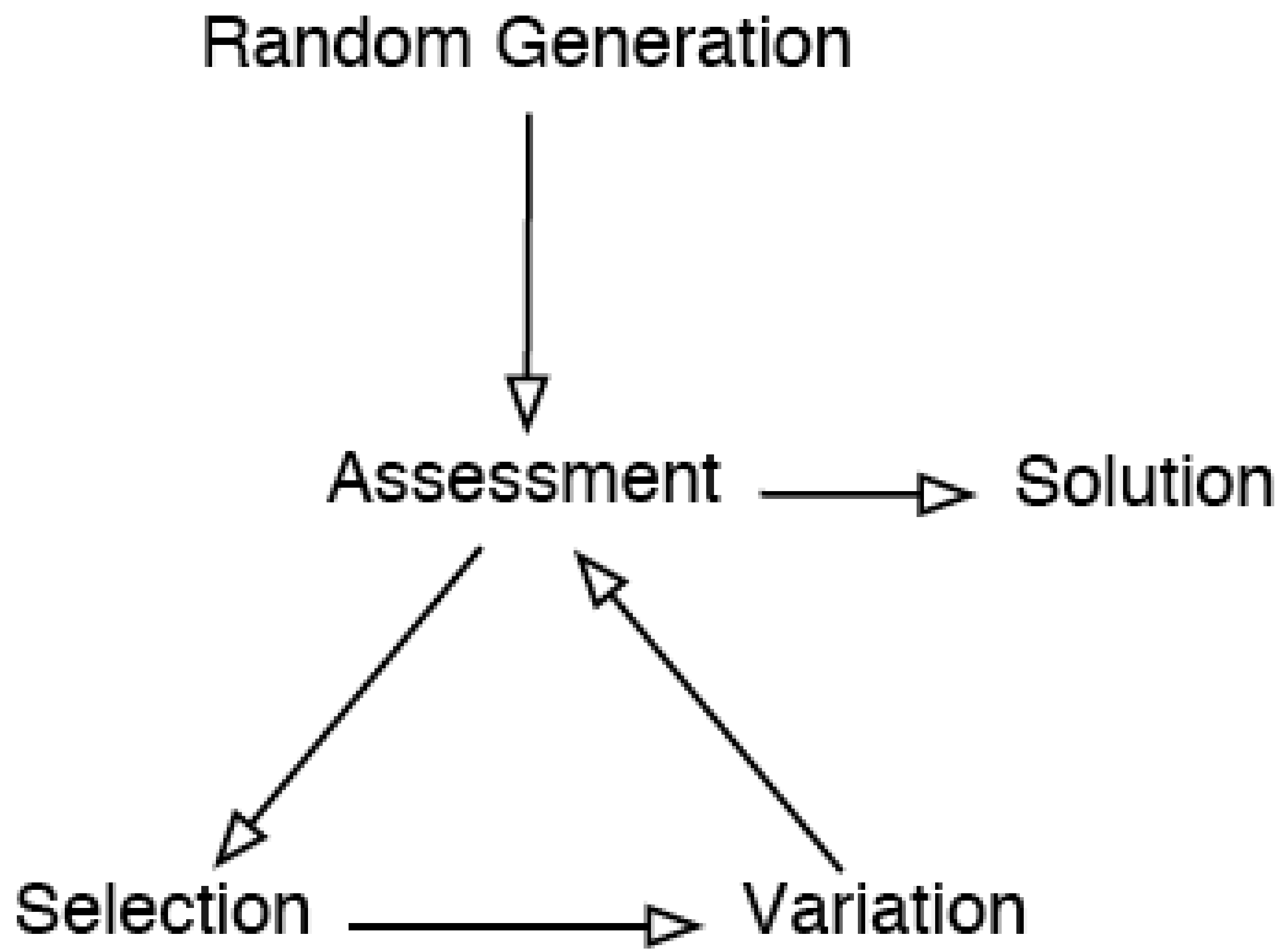
Principles, Products, and Prospects

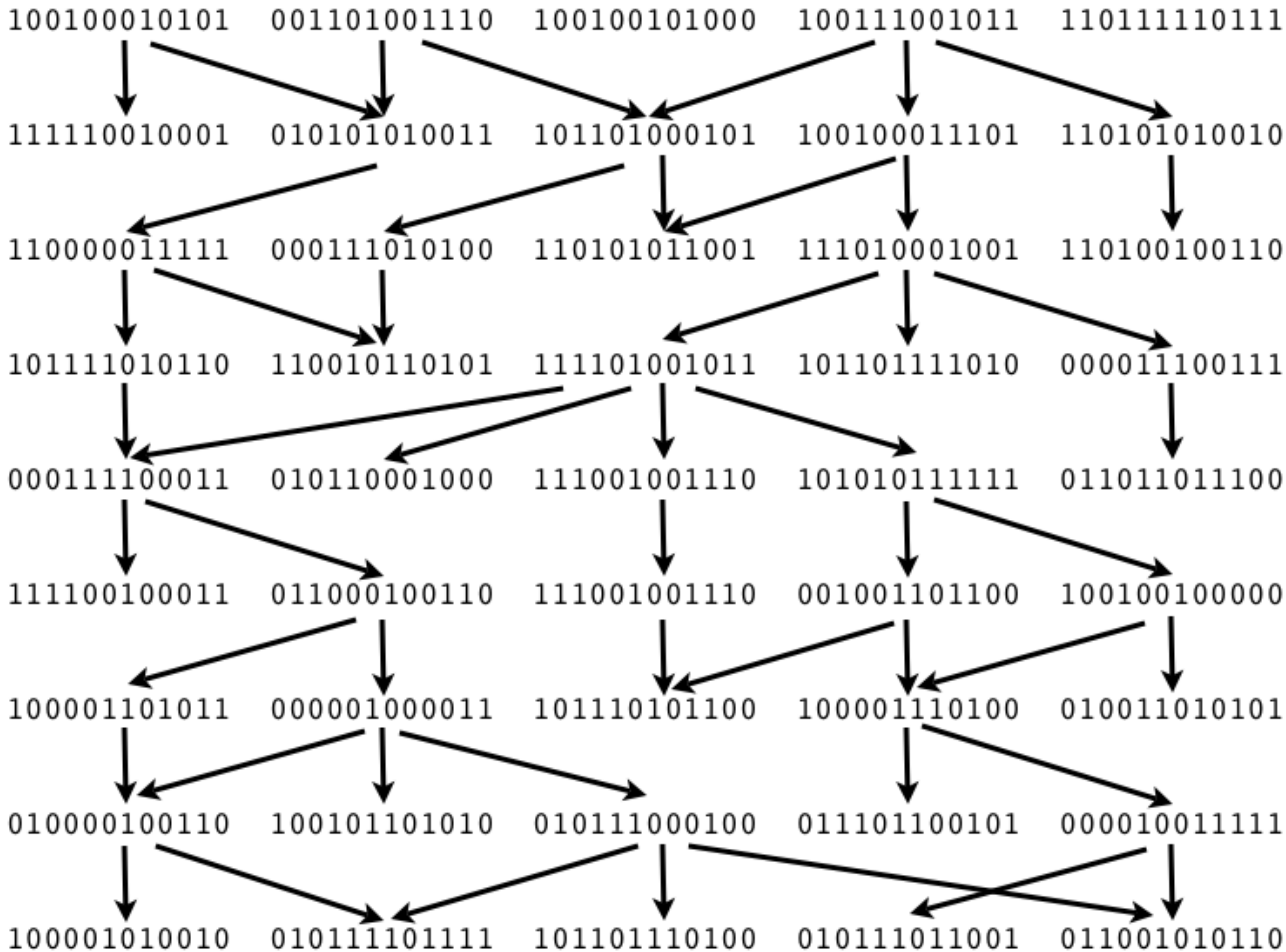
Lee Spector
Cognitive Science, Hampshire College
Computer Science, UMass Amherst
http://hampshire.edu/lspector

Outline

- Program evolution
  - Genetic programming
  - Digital organisms
- Expressive representations (Push)
- Hints from nature (lexicase selection)
- The future

Genetic Algorithms





100100010101
001101001110
100100101000
100111001011
110111110111
111110010001
010101010011
101101000101
100100011101
110101010010
110000011111
000111010100
110101011001
111010001001
110100100110
101111010110
110010110101
111101001011
101101111010
000011100111
000111100011
010110001000
111001001110
101010111111
011011011100
111100100011
011000100110
111001001110
001001101100
100100100000
100001101011
000001000011
101110101100
100001110100
010011010101
010000100110
100101101010
010111000100
011101100101
000010011111
100001010010
010111101111
101101110100
010111011001
011001010110

Genetic Programming

- Genetic algorithms that produce executable computer programs
- Programs are assessed by executing them
- Automatic programming by evolution

Program Representations

- 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, ...).

Mutating Lisp

```
(+ (* X Y)
   (+ 4 (- Z 23)))

(+ (* X Y)
   (+ 4 (- Z 23)))

(+ (- (+ 2 2) Z)
   (+ 4 (- Z 23)))
```

Recombining Lisp

```
Parent 1:  (+  (* X Y)
               (+ 4 (- Z 23)))
Parent 2:  (-  (* 17 (+ 2 X))
              (* (- (* 2 Z) 1)
                 (+ 14 (/ Y X))))

Child 1:   (+  (- (* 2 Z) 1)
              (+ 4 (- Z 23)))
Child 2:   (-  (* 17 (+ 2 X))
              (* (* X Y)
                 (+ 14 (/ Y X))))
```

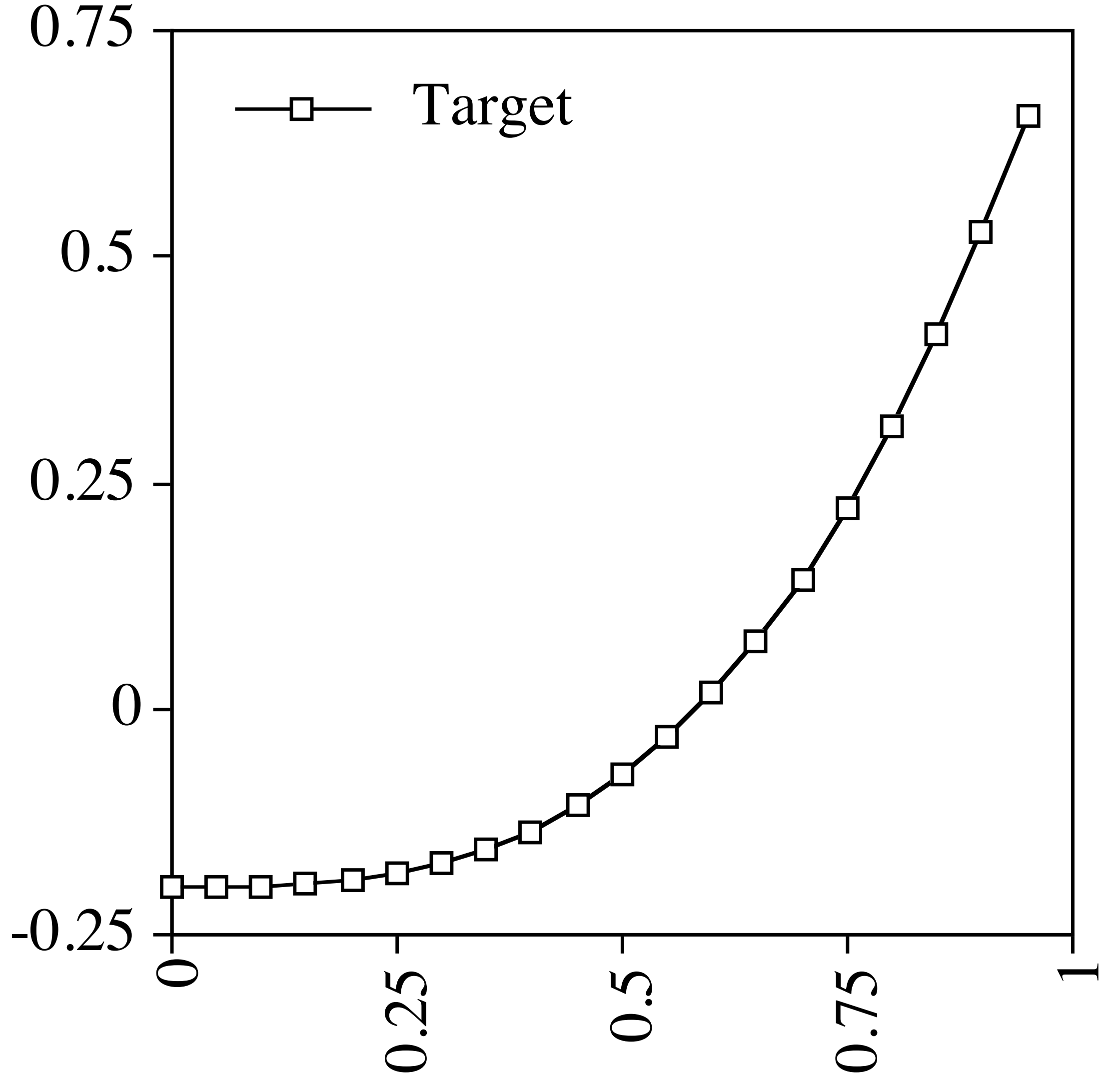
Symbolic Regression

- A simple example
- Given a set of data points, evolve a program that produces y from x.
- Primordial ooze: +, -, *, %, x, 0.1
- Fitness = error (smaller is better)

GP Parameters

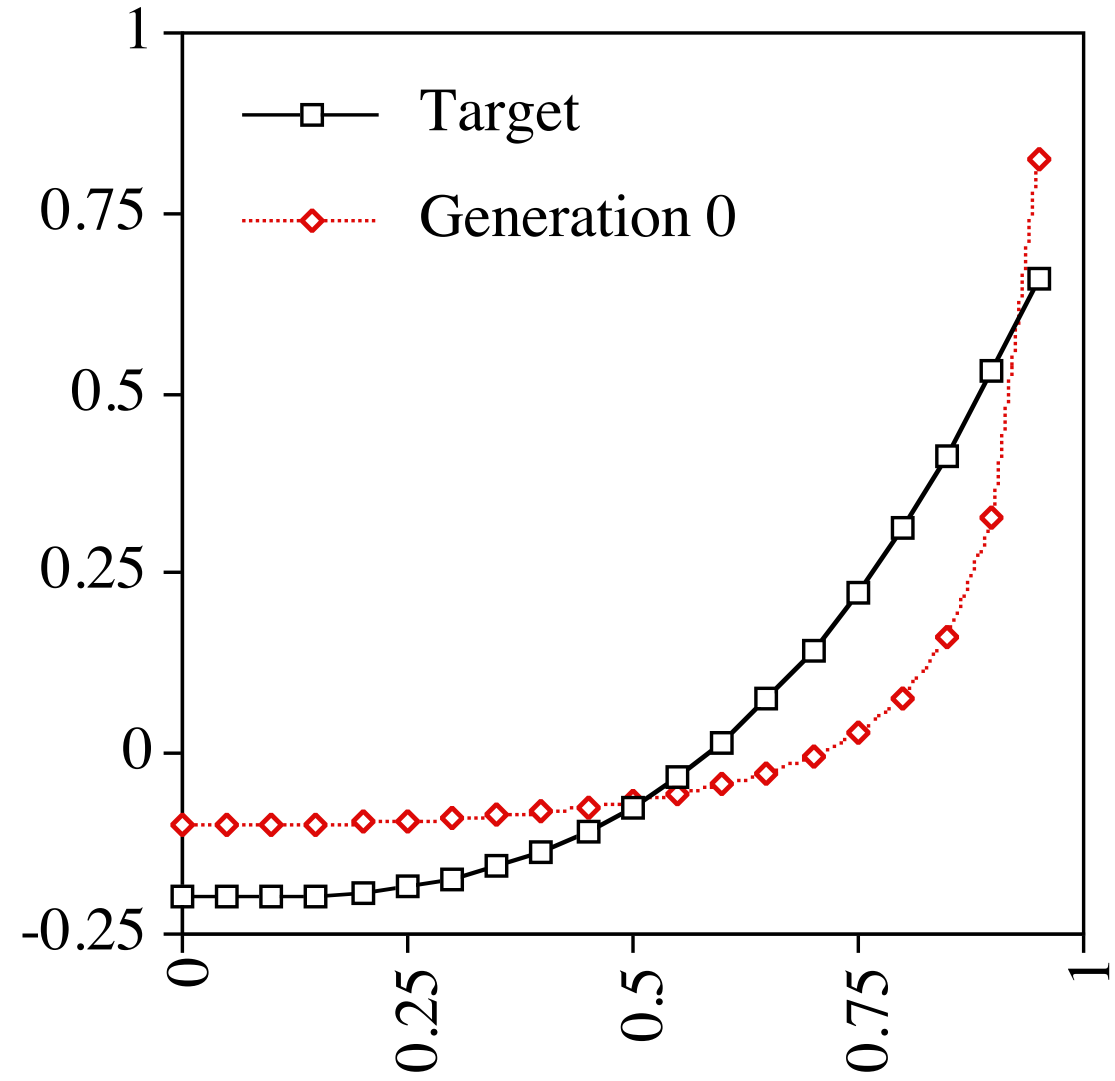
Maximum number of Generations: 51
Size of Population: 1000
Maximum depth of new individuals: 6
Maximum depth of new subtrees for mutants: 4
Maximum depth of individuals after crossover: 17
Fitness-proportionate reproduction fraction: 0.1
Crossover at any point fraction: 0.3
Crossover at function points fraction: 0.5
Selection method: FITNESS-PROPORTIONATE
Generation method: RAMPED-HALF-AND-HALF
Randomizer seed: 1.2

Evolving $y = x^3 - 0.2$



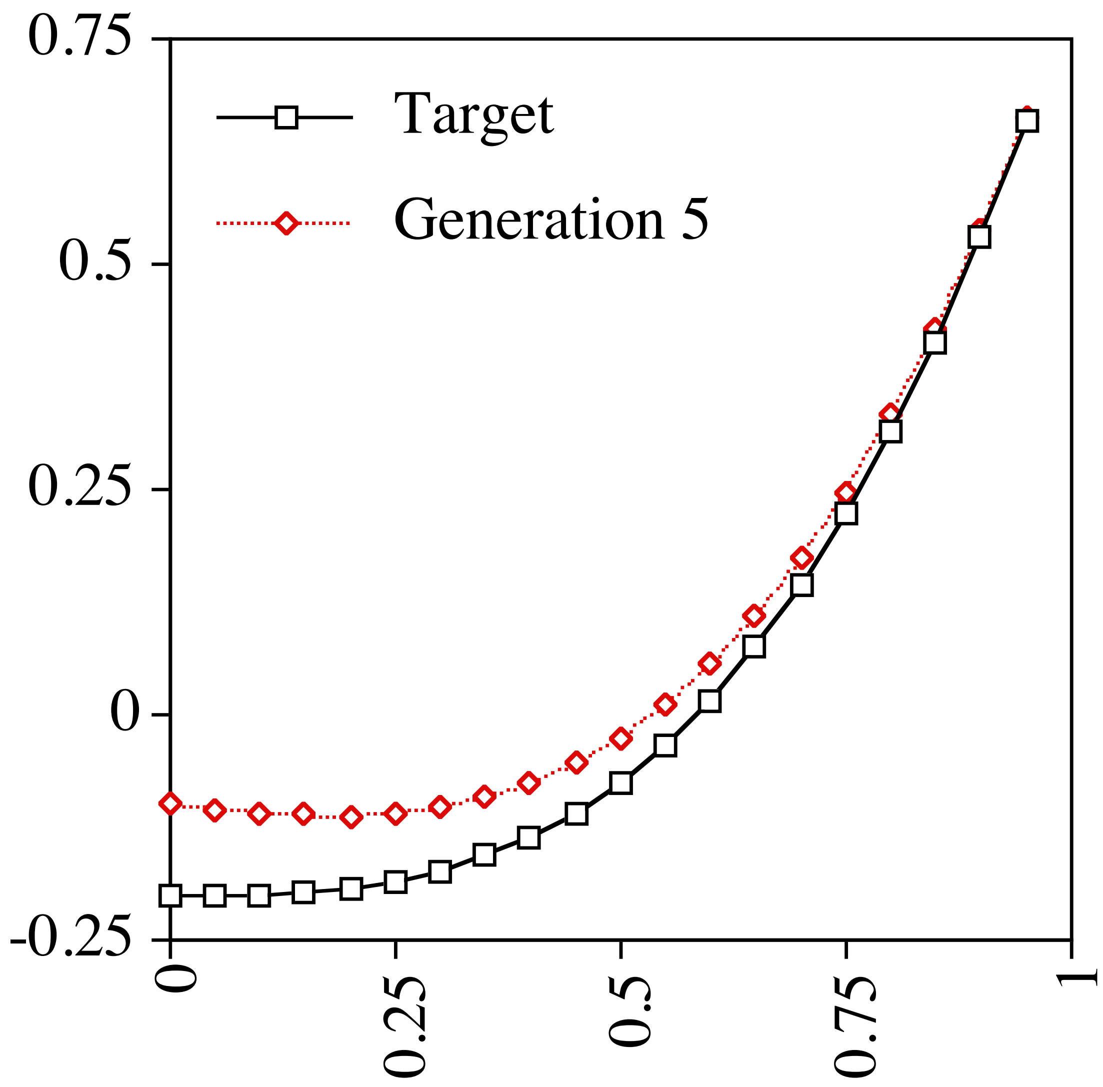
Best Program, Gen 0

```
(- (% (* 0.1
         (* X X))
      (- (% 0.1 0.1)
         (* X X)))
   0.1)
```



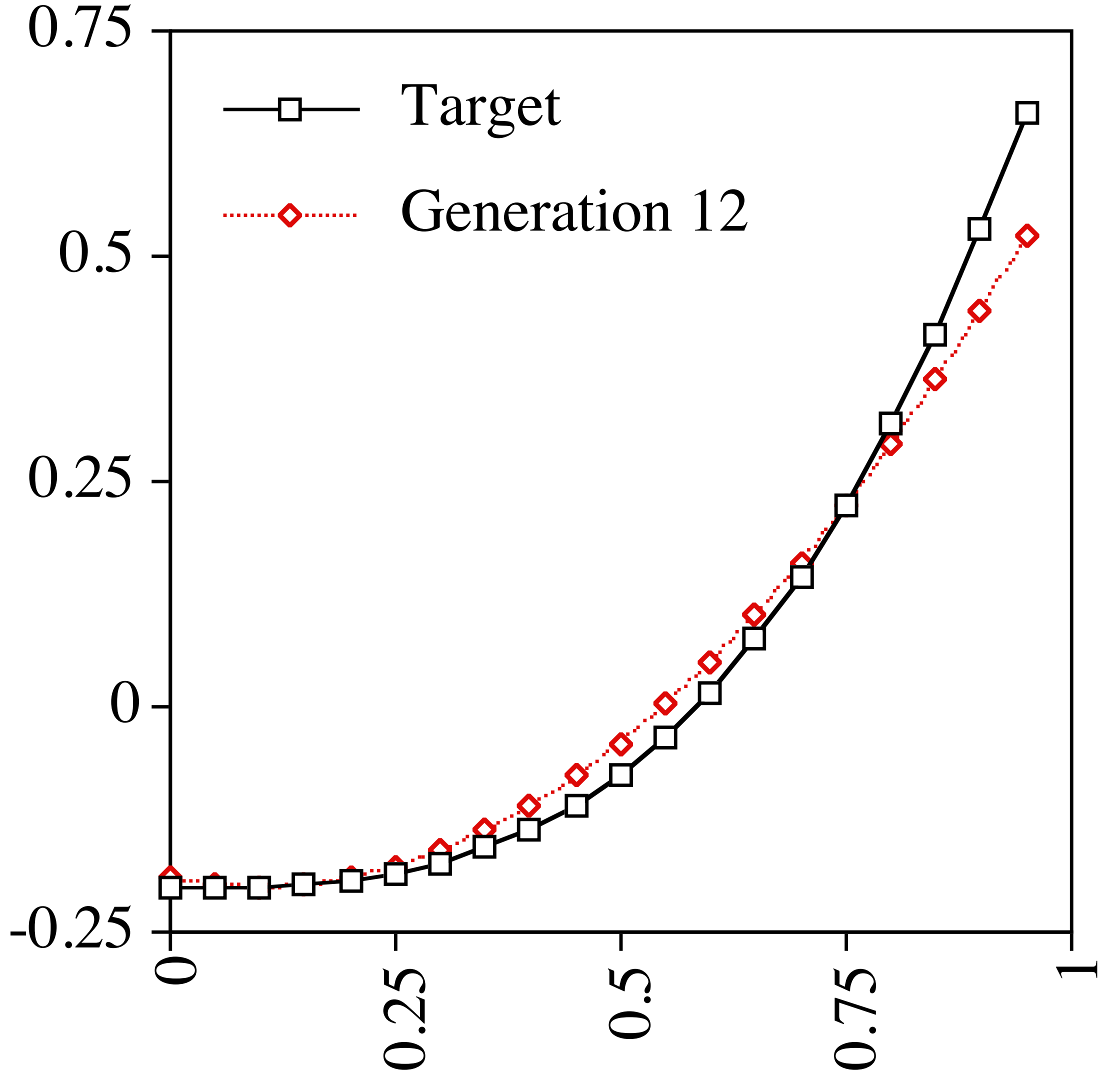
Best Program, Gen 5

```
(- (* (* (% X 0.1)
         (* 0.1 X))
      (- X
         (% 0.1 X)))
   0.1)
```



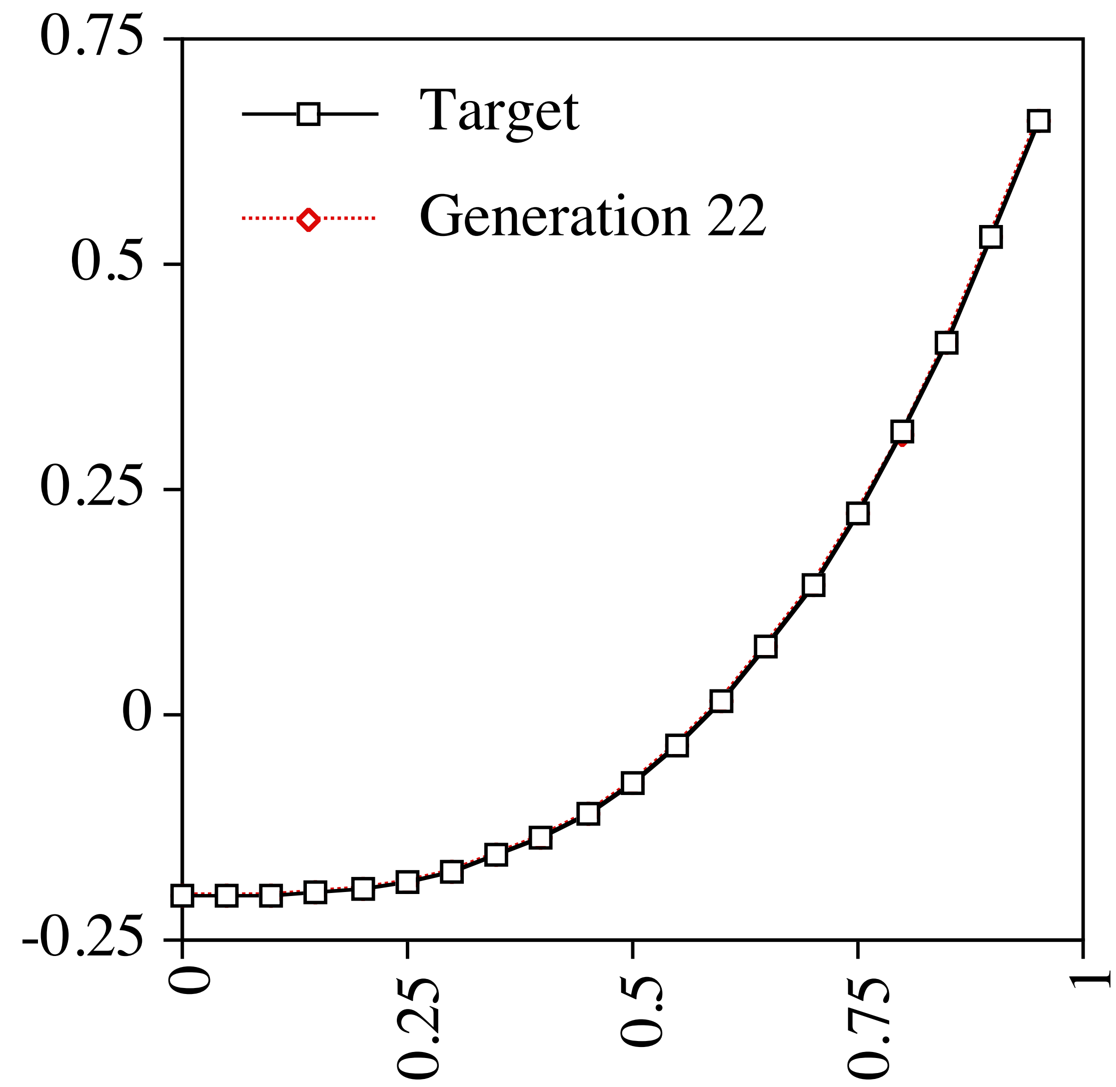
Best Program, Gen 12

```
(+ (- (- 0.1
         (- 0.1
            (- (* X X)
               (+ 0.1
                  (- 0.1
                     (* 0.1
                        0.1))))))
      (* X
         (* (% 0.1
               (% (* (* (- 0.1 0.1)
                        (+ X
                           (- 0.1 0.1)))
                     X)
                  (+ X (+ (- X 0.1)
                          (* X X)))))
            (+ 0.1 (+ 0.1 X)))))
   (* X X))
```



Best Program, Gen 22

```
(- (- (* X (* X X))
      0.1)
   0.1)
```



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

Humies 2008
GOLD MEDAL

Goal

- Find finite algebra terms that have certain special properties
- For decades there was no way to produce these terms in general, short of exhaustive search
- Current best methods produce enormous terms
- Want to be able to find small terms quickly

Significance, Time

	Uninformed Search Expected Time (Trials)
3 element algebras Mal'cev Pixley/majority discriminator	 5 seconds $(3^{15} \approx 10^{7})$ 1 hour $(3^{21} \approx 10^{10})$ 1 month $(3^{27} \approx 10^{13})$
4 element algebras Mal'cev Pixley/majority discriminator	 10^{3} years $(4^{28} \approx 10^{17})$ 10^{10} years $(4^{40} \approx 10^{24})$ 10^{24} years $(4^{64} \approx 10^{38})$

Significance, Time

	Uninformed Search Expected Time (Trials)	GP Time
3 element algebras Mal'cev Pixley/majority discriminator	 5 seconds ($3^{15} \approx 10^7$) 1 hour ($3^{21} \approx 10^{10}$) 1 month ($3^{27} \approx 10^{13}$)	 1 minute 3 minutes 5 minutes
4 element algebras Mal'cev Pixley/majority discriminator	 10^3 years ($4^{28} \approx 10^{17}$) 10^{10} years ($4^{40} \approx 10^{24}$) 10^{24} years ($4^{64} \approx 10^{38}$)	 30 minutes 2 hours ?

Significance, Size

Term Type	Primality Theorem
Mal'cev	$10,060,219$
Majority	$6,847,499$
Pixley	$1,257,556,499$
Discriminator	$12,575,109$

(for A_l)

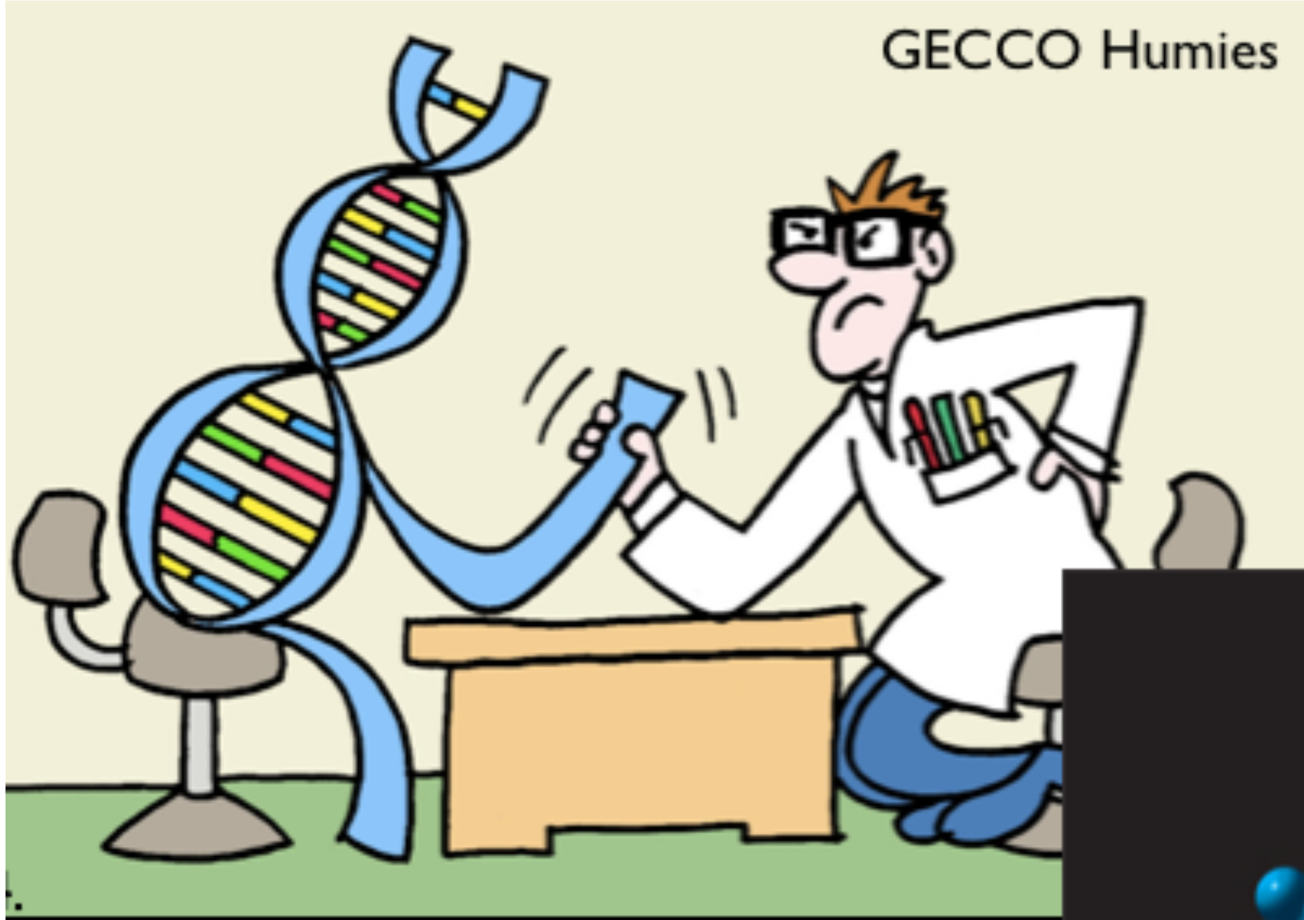
Significance, Size

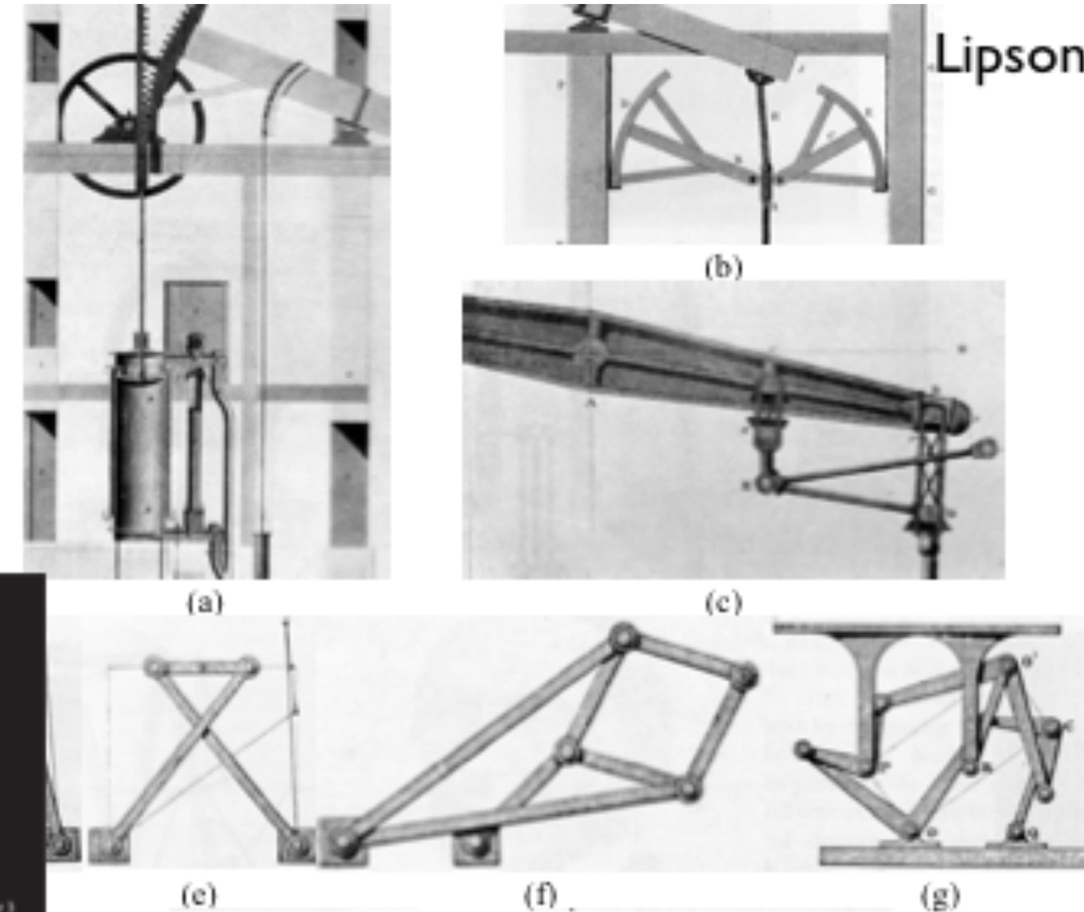
Term Type	Primality Theorem	GP
Mal'cev	$10,060,219$	12
Majority	$6,847,499$	49
Pixley	$1,257,556,499$	59
Discriminator	$12,575,109$	39

(for A_l)

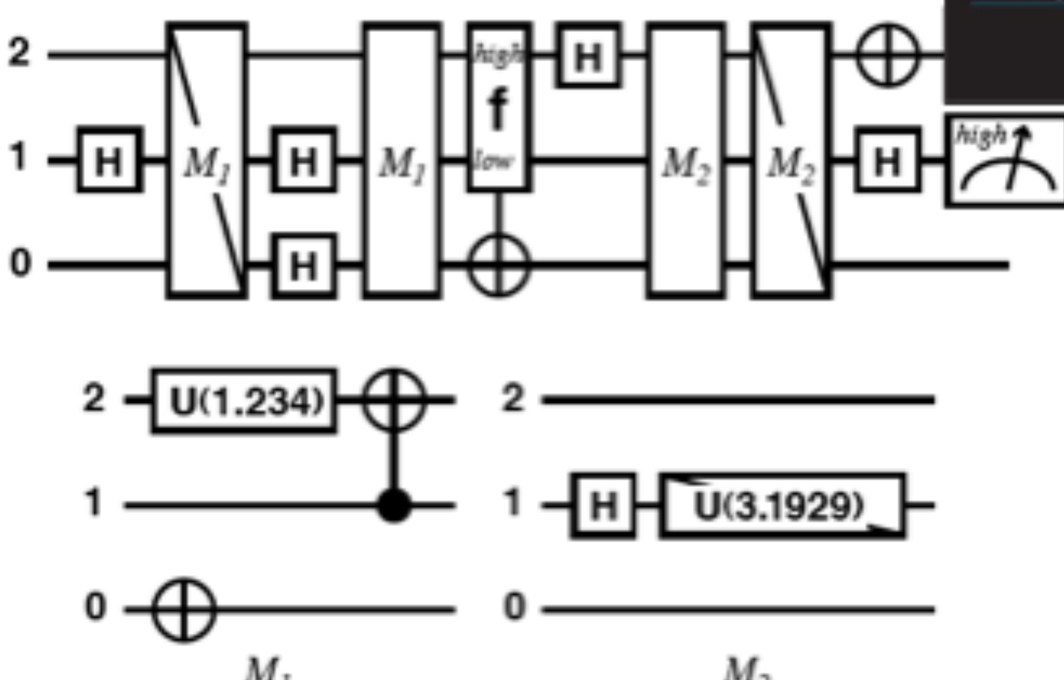
Human Competitive?

- **Rather: human-WHOMPING!**
- Outperforms humans *and all other known methods* on significant problems, providing benefits of *several orders of magnitude* with respect to search speed and result size
- Here GP has provided the first solution to a previously open problem in the field

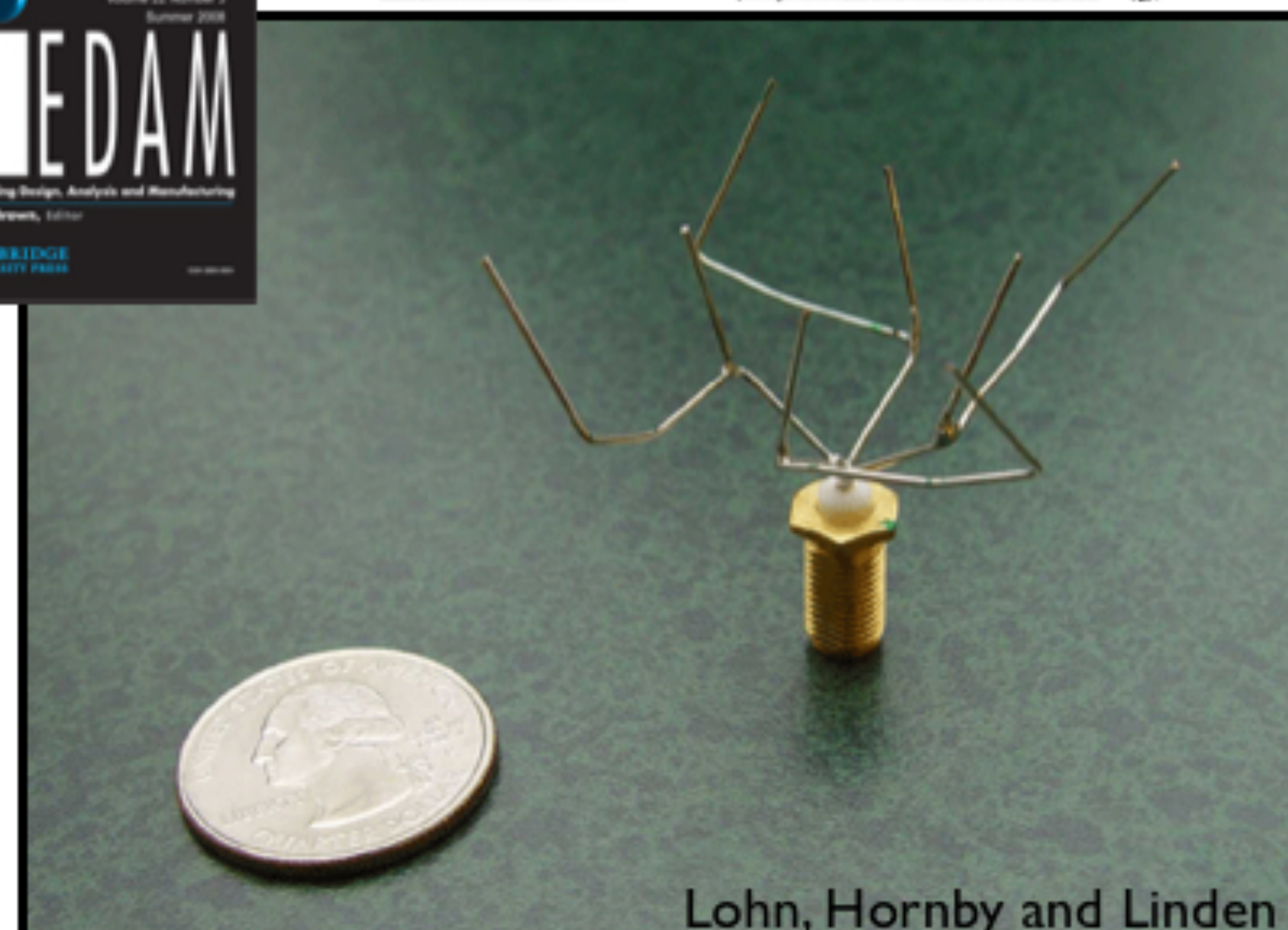

GECCO Humies


Lipson
(a)
(b)
(c)
(e)
(f)
(g)


AIEDAM
Artificial Intelligence for Engineering Design, Analysis and Manufacturing
David C. Brown, Editor
Cambridge University Press


2
1
0
H
M_1
high
f
low
M_2
U(1.234)
U(3.1929)
M_1
M_2

Spector


Lohn, Hornby and Linden

GPTP 2014

Analyzing a Decade of Human-Competitive ("HUMIE") Winners: What Can We Learn?

Karthik Kannappan, Lee Spector, Moshe Sipper, Thomas Helmuth, William Lacava, Jake Wisdom, Omri Bernstein

Humies Criteria

- 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).

Humies Algorithms

Algorithm	Count
Genetic Programming (GP)	22
Genetic Algorithms (GA)	15
Evolutionary Strategies (ES)	2
Differential Evolution (DE)	1
Genetics Based Machine Learning (GBML)	1
Metaheuristic	1

Humies Applications

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)

Humies Problem Types

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

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."

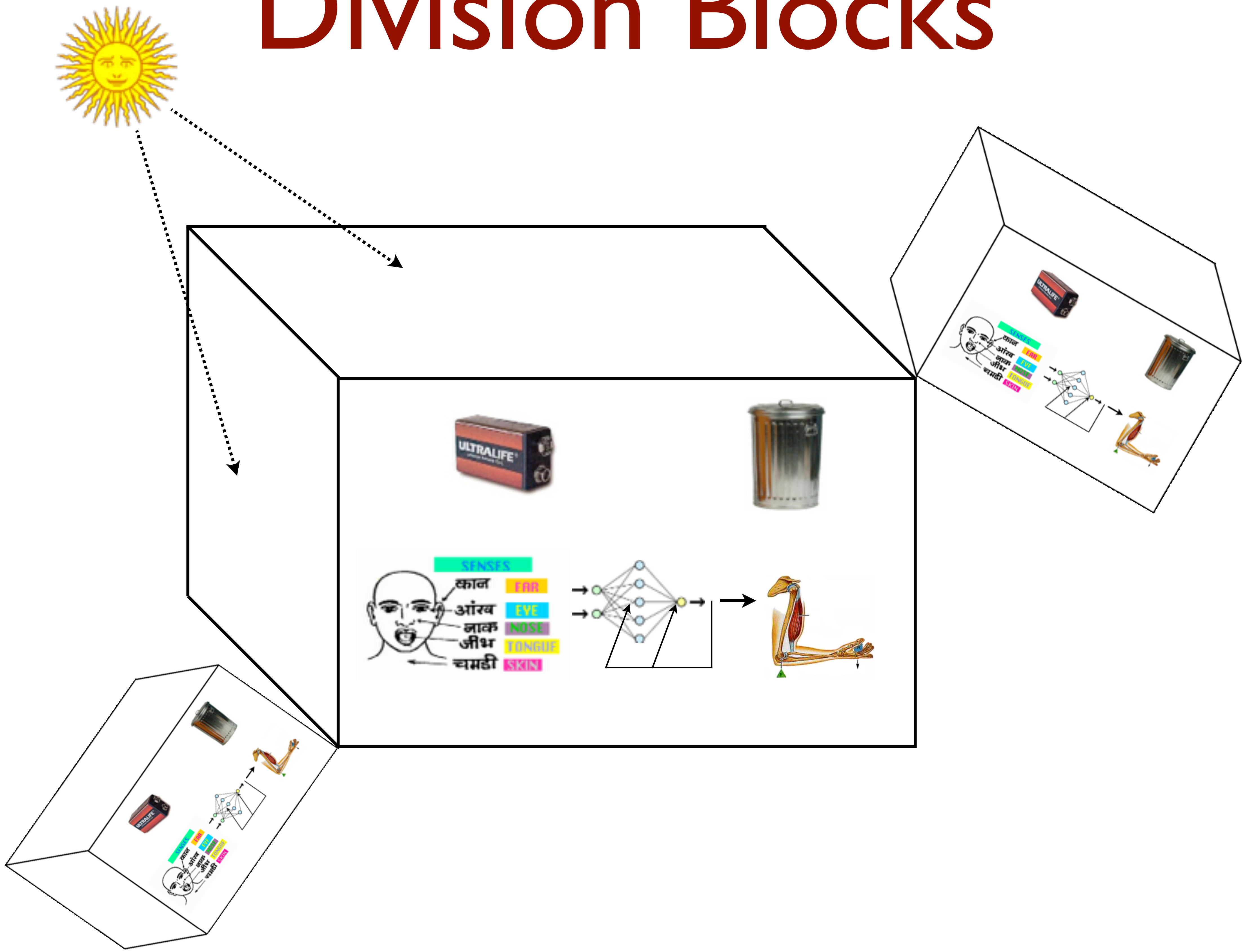
Digital Organisms

- For the study of general principles of living systems
- Populations of individuals that act locally in an environment
- Explore, in silico, key interactions among development, form, physics, behavior (including reproductive behavior), and ecology that underpin biological evolution
- How do these factors interact, under natural selection, to produce adaptive complexity?
- Core War, Tierra, Avida, Echo, Polyworld, Framsticks, ...

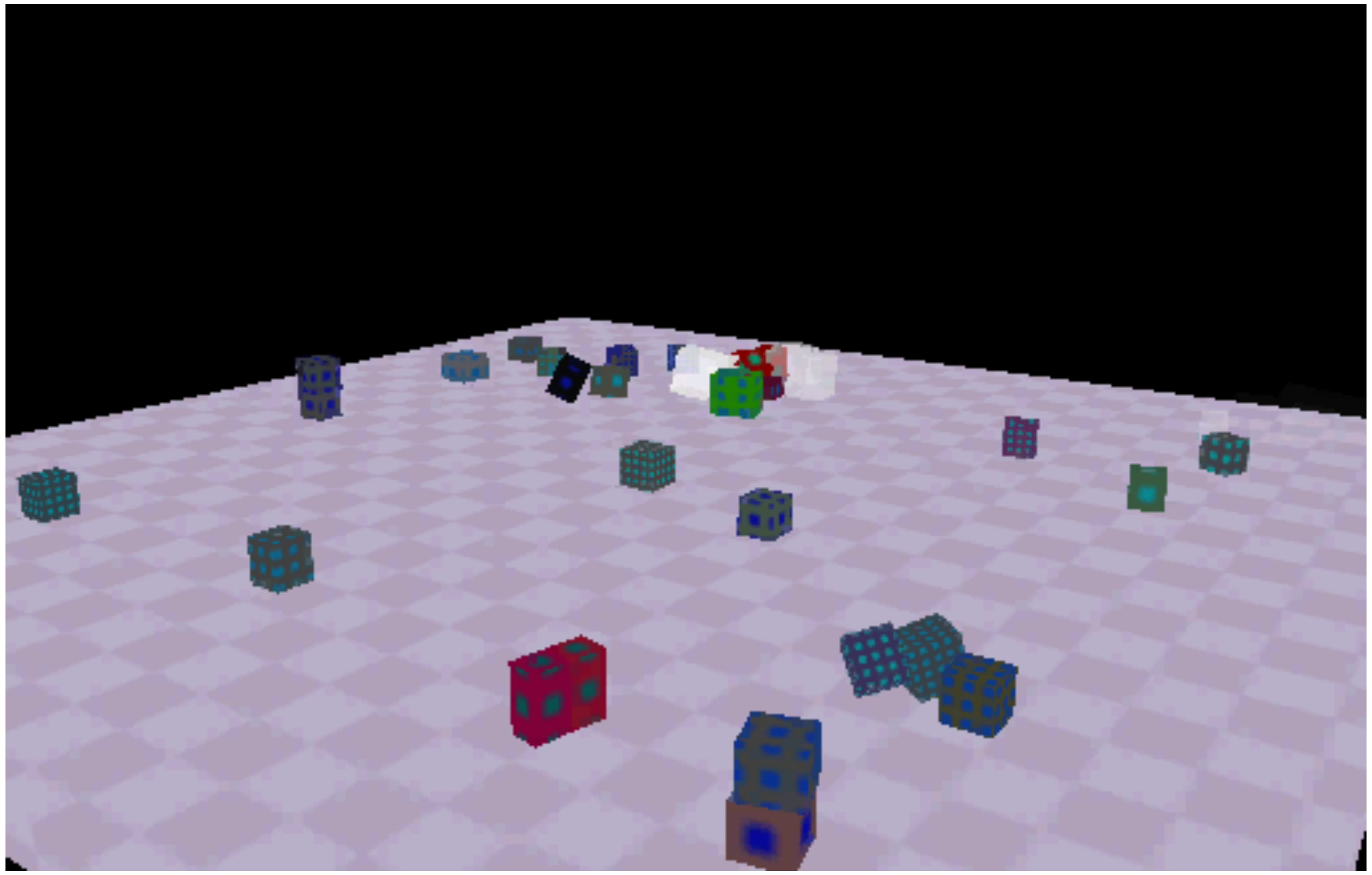
Unfortunately Necessary

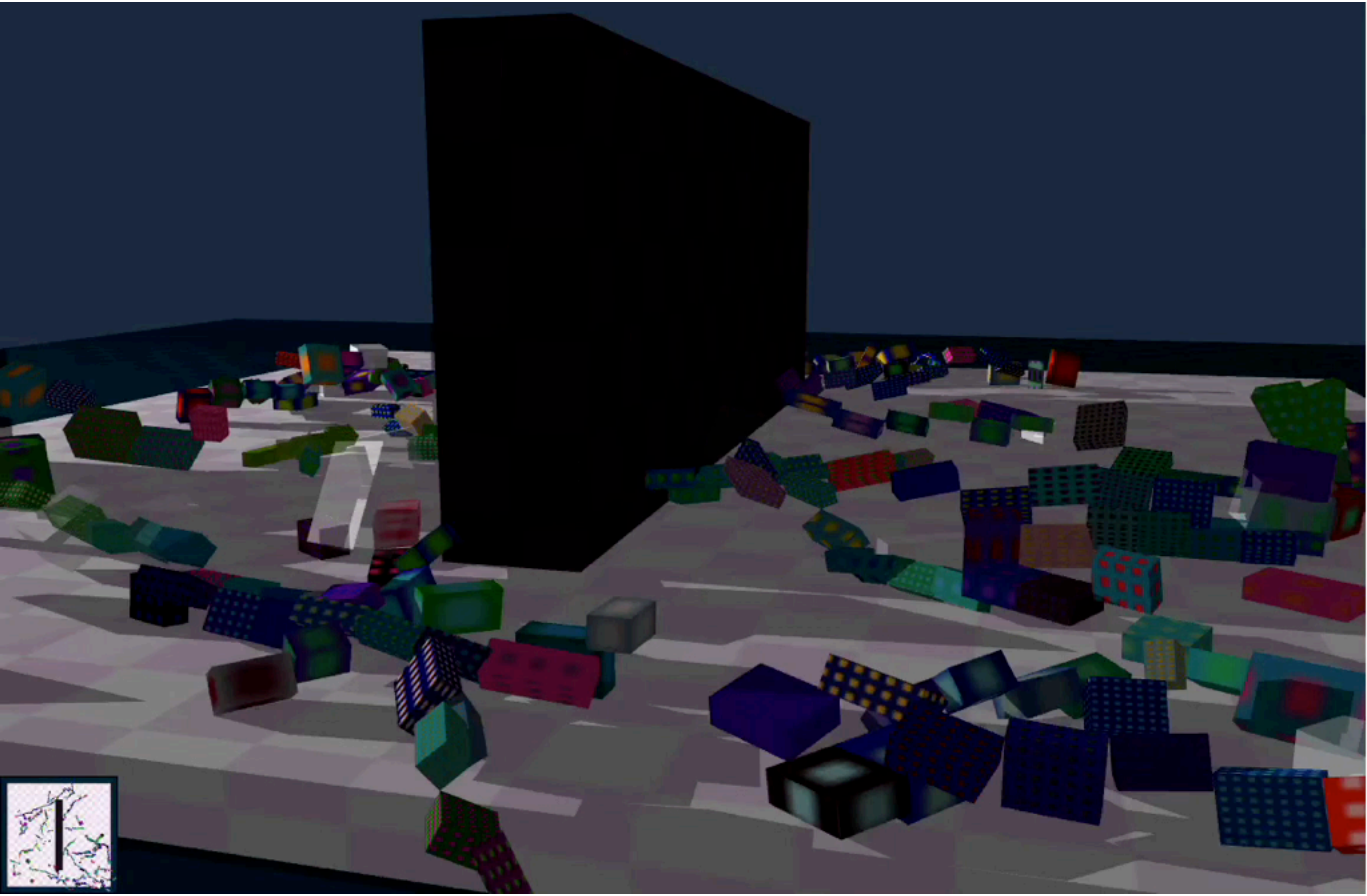
- Outrageous simplifications
- Combinations of features normally observed at radically different scales

Division Blocks

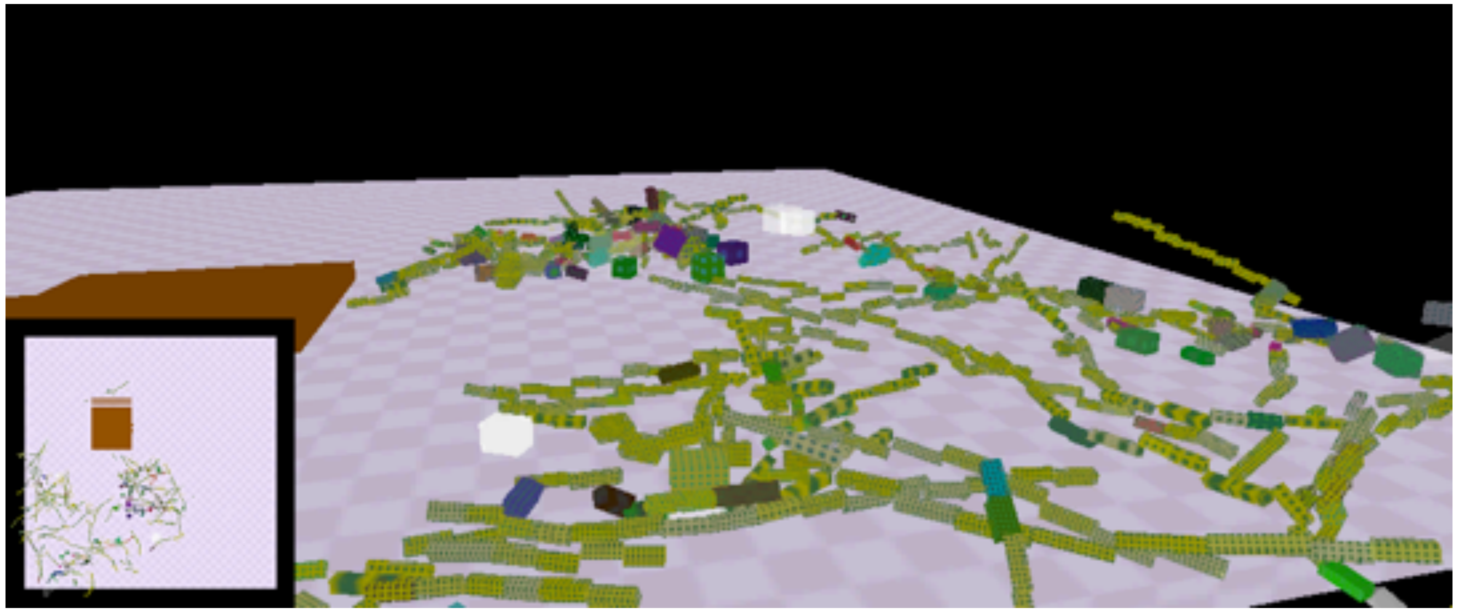


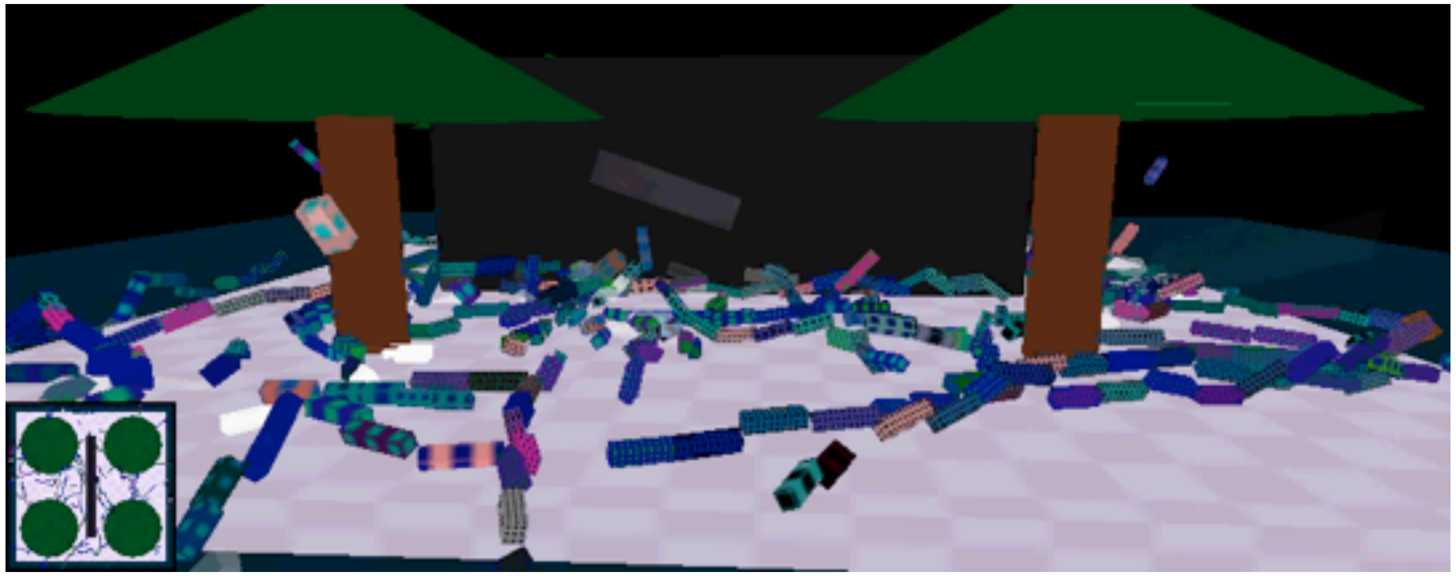
Reproductive Competence

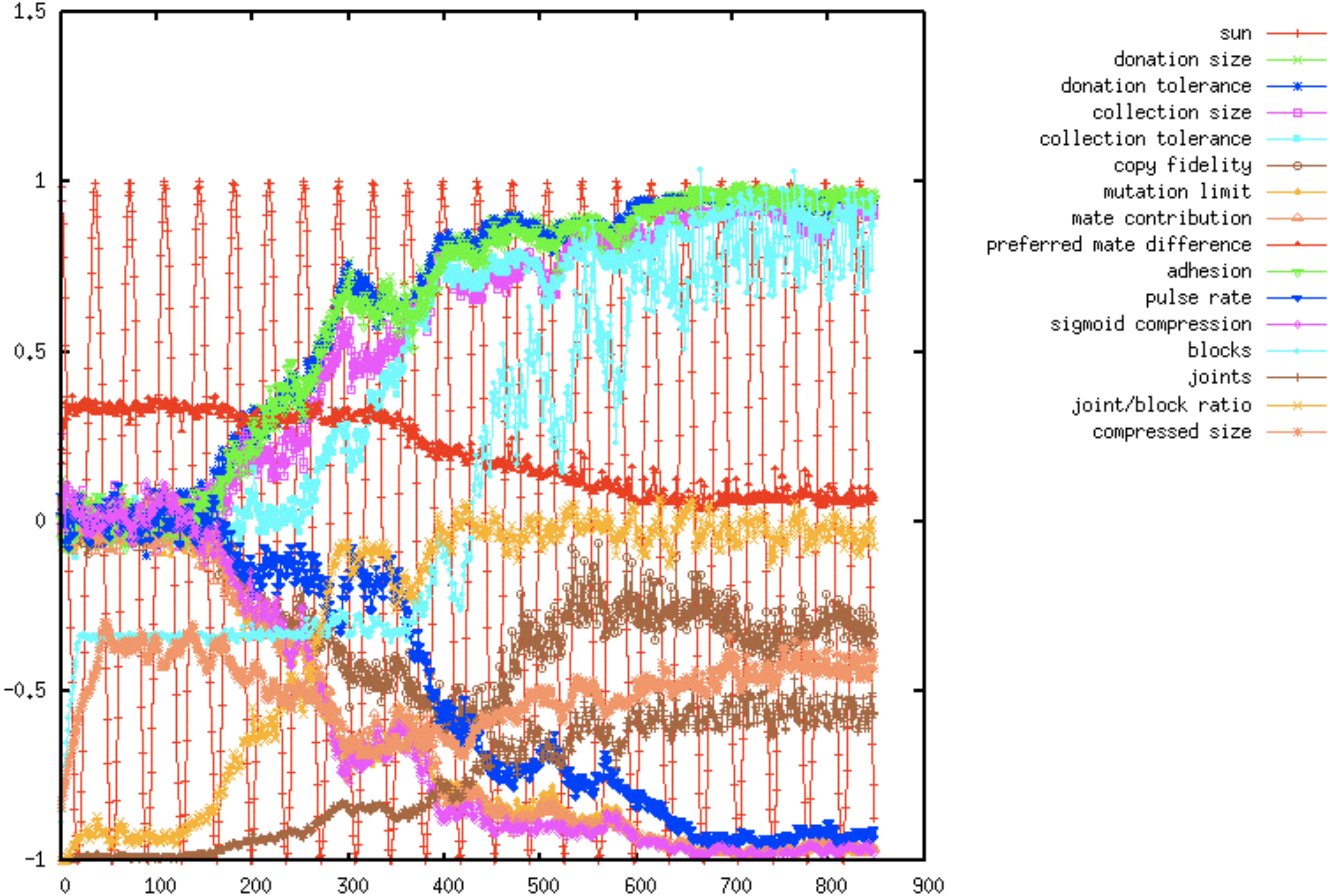




Variations







1.5
1
0.5
0
-0.5
-1
0
100
200
300
400
500
600
700
800
900
sun
donation size
donation tolerance
collection size
collection tolerance
copy fidelity
mutation limit
mate contribution
preferred mate difference
adhesion
pulse rate
sigmoid compression
blocks
joints
joint/block ratio
compressed size

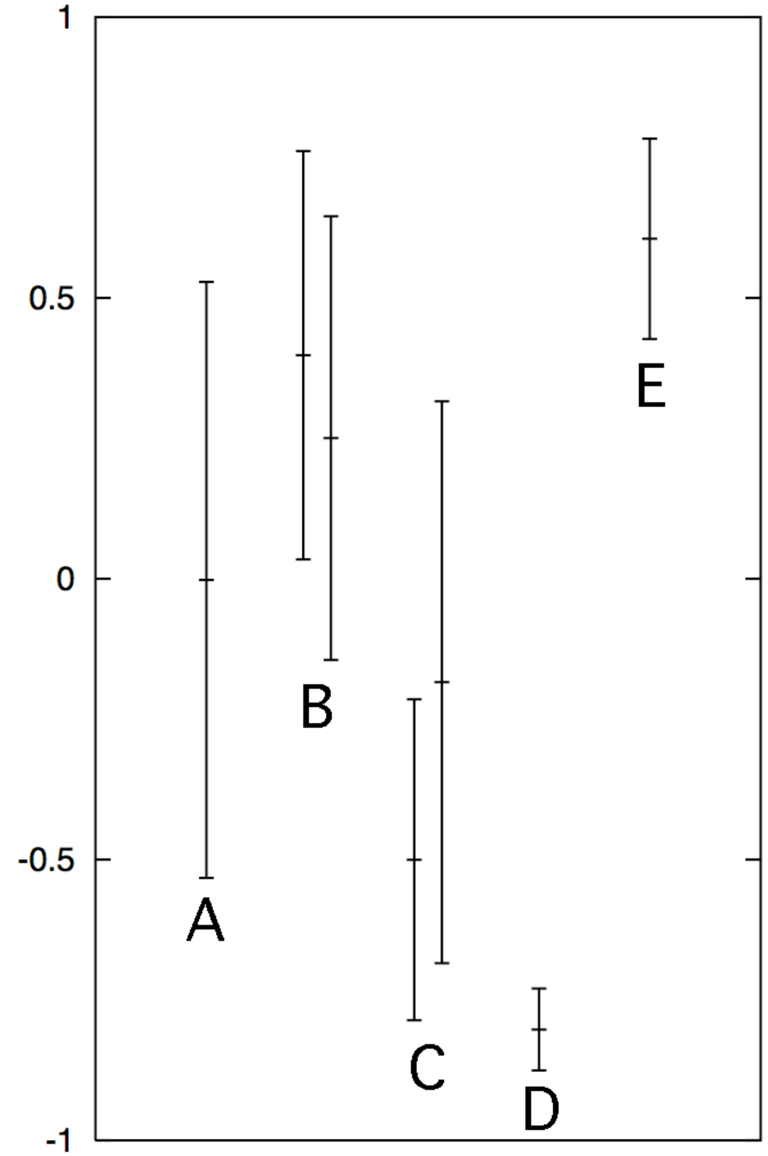


Figure 4: Averaged data from 40 **runs of the Division Blocks system, collected after** 1000 **time steps of reproductive competence. Error bars indicate** ± 1 **standard deviation. A: average tag values; B: average** `donationsize` **(left) and** `donationtolerance` **(right); C: average** `stemdonationsize` **(left) and** `stemdonationtolerance` **(right); D: average** `matecontribution`**; E: average** `adhesion`**.**

Expressiveness

- What set of computations can be expressed in the language?
- Maximally (equally) expressive:
  - Turing machine tables
  - Lambda calculus expressions
  - Partial recursive functions
  - Register machine programs
  - Assembly language programs
  - etc.

Evolvability

The fact that a computation can be expressed in a formalism does not imply that a correct expression can be produced in that formalism by a human programmer or by an evolutionary process.

Nature’s Language

- Can be characterized at multiple levels
- Carbon chemistry
- Combinatoric representations
- Vast, interconnected space of structures and dynamics
- Expressive!

Data/Control Structure

- Data abstraction and organization

  Data types, variables, name spaces, data structures, ...

- Control abstraction and organization

  Conditionals, loops, modules, threads, ...

Structure via GP (1)

- Specialize GP techniques to directly support human programming language abstractions
- Strongly typed genetic programming
- Module acquisition/encapsulation systems
- Automatically defined functions
- Automatically defined macros
- Architecture altering operations

ADFs

- All programs in the population have the same, pre-specified architecture
- Genetic operators respect that architecture
- Significant implementation costs
- Significant pre-specification
- Architecture-altering operations: more power and higher costs

Structure via GP (2)

- Evolve programs in a minimal-syntax language that is nonetheless expressive enough to support a full range of data and control abstractions
- Orchestrate data flows via stacks, not via syntax
- Minimal syntax + maximal semantics
- Push

Push

- Designed for program evolution
- Stack-based postfix language with one stack per type
- Types include: integer, float, boolean, string, code, exec, vector, [add more as needed]
- Minimal syntax:
  program → instruction | literal | (program*)
- Missing argument? NOOP

Sample Push Instructions

Stack manipulation instructions (all types)	POP, SWAP, YANK, DUP, STACKDEPTH, SHOVE, FLUSH, $=$
Math (INTEGER and FLOAT)	$+$, $-$, $/$, $*$, $>$, $<$, MIN, MAX
Logic (BOOLEAN)	AND, OR, NOT, FROMINTEGER
Code manipulation (CODE)	QUOTE, CAR, CDR, CONS, INSERT, LENGTH, LIST, MEMBER, NTH, EXTRACT
Control manipulation (CODE and EXEC)	DO*, DO*COUNT, DO*RANGE, DO*TIMES, IF

Why Push?

- Highly expressive: data types, data structures, variables, conditionals, loops, recursion, modules, ...
- Elegant: minimal syntax and a simple, stack-based execution architecture
- Evolvable
- Extensible
- Supports uniform variation
- Supports several forms of meta-evolution

Selection

- In genetic programming, selection is typically based on average performance across all test cases
- In nature, selection is typically based on sequences of interactions with the environment

Tournament Selection

- For some pre-determined tournament size *n*
- Choose *n* individuals from the population randomly
- Select the best of these *n*
- "The best" is the one with the best average performance across all test cases

Implicit Fitness Sharing

- The average is weighted so that test cases for which the population performs worse count for more

Lexicase Selection

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

Lexicase Selection

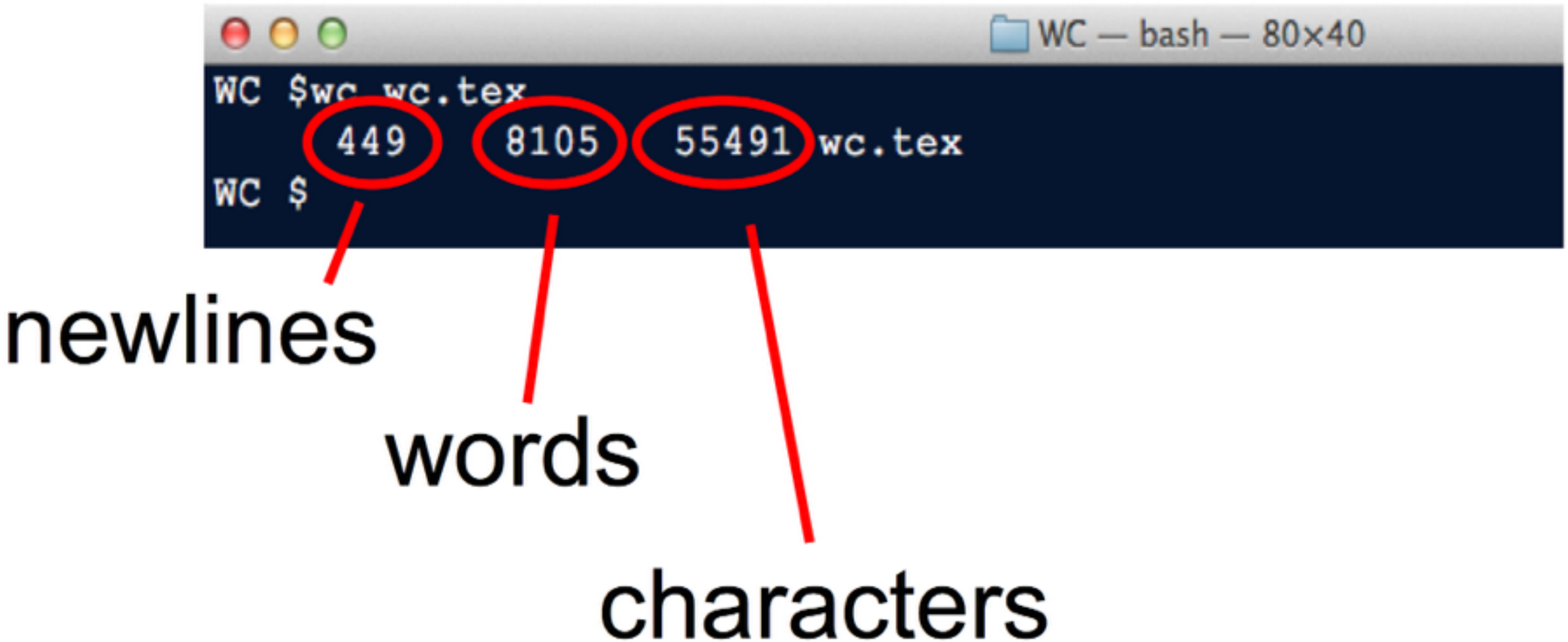
To select single parent:

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

wc


WC — bash — 80×40
WC $wc wc.tex
449 8105 55491 wc.tex
WC $
newlines
words
characters

wc Test Cases

- 0 to 100 character files
- Random string (200 training, 500 test)
- Random string ending in newline (20 training, 50 test)
- Edge cases (22; empty string, multiple newlines, etc.)

Instructions

- General purpose
- I/O
- Control flow
- Tags for modularity
- String, integer, and boolean
- Random constants

Input	file_readchar, file_readline, file_EOF, file_begin
Output	output_charcount, output_wordcount, output_linecount
Exec	exec_pop, exec_swap, exec_rot, exec_dup, exec_yank, exec_yankdup, exec_shove, exec_eq, exec_stackdepth, exec_when, exec_if, exec_do*times, exec_do*count, exec_do*range, exec_y, exec_k, exec_s
Tag ERCs	tag_exec, tag_integer, tag_string, tagged
String	string_split, string_parse_to_chars, string_whitespace, string_contained, string_reverse, string_concat, string_take, string_pop, string_eq, string_stackdepth, string_rot, string_yank, string_swap, string_yankdup, string_flush, string_length, string_shove, string_dup
Integer	integer_add, integer_swap, integer_yank, integer_dup, integer_yankdup, integer_shove, integer_mult, integer_div, integer_max, integer_sub, integer_mod, integer_rot, integer_min, integer_inc, integer_dec
Boolean	boolean_swap, boolean_and, boolean_not, boolean_or, boolean_frominteger, boolean_stackdepth, boolean_dup
ERC	Integer from [-100, 100] {"\n", "\t", "␣" } $\{x \mid x$ is a non-whitespace character$\}$

wc Results

Selection	Tournament Size	Successes (200 runs)
Lexicase	-	11
Tournament	3	0
	5	0
	7	0
Implicit Fitness Sharing	3	0
	5	0
	7	0

Epistasis

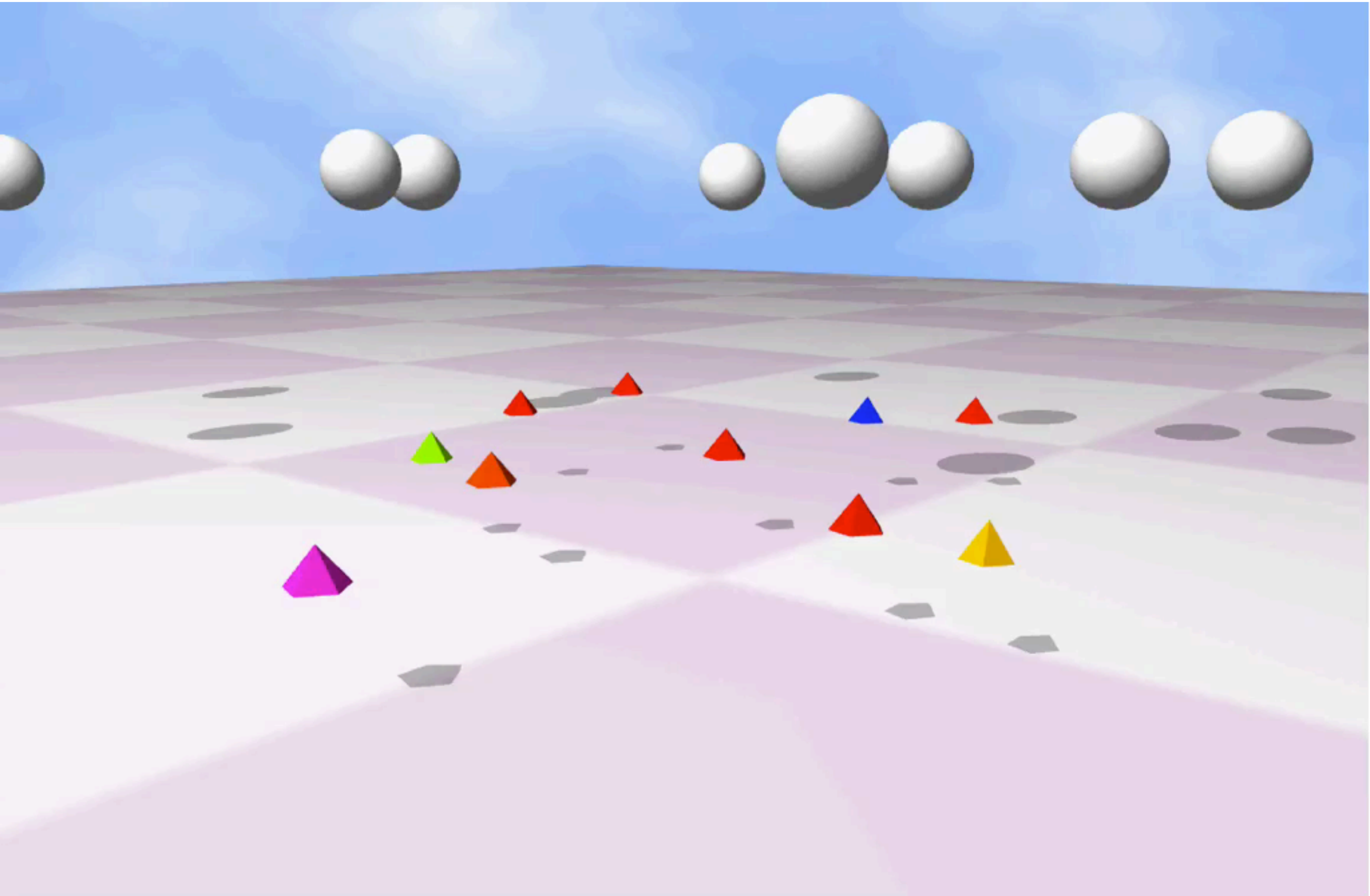
- Collaboration with Jason Moore at the Geisel School of Medicine at Dartmouth
- Genetic analysis of susceptibility to human diseases
- Difficult because of epistatic interactions among genes
- Using GP to find genome classifiers
- Hypothesis: Because it selects for performance on combinations of cases, lexicase selection will help GP systems to find classifiers that recognize expistatic interactions

Autoconstructive Evolution

- Individual programs make their own children
- Hence they control their genetic representations, mutation rates, sexuality, reproductive timing, etc.
- The machinery of reproduction and diversification (i.e., the machinery of evolution) evolves

SwarmEvolve 2

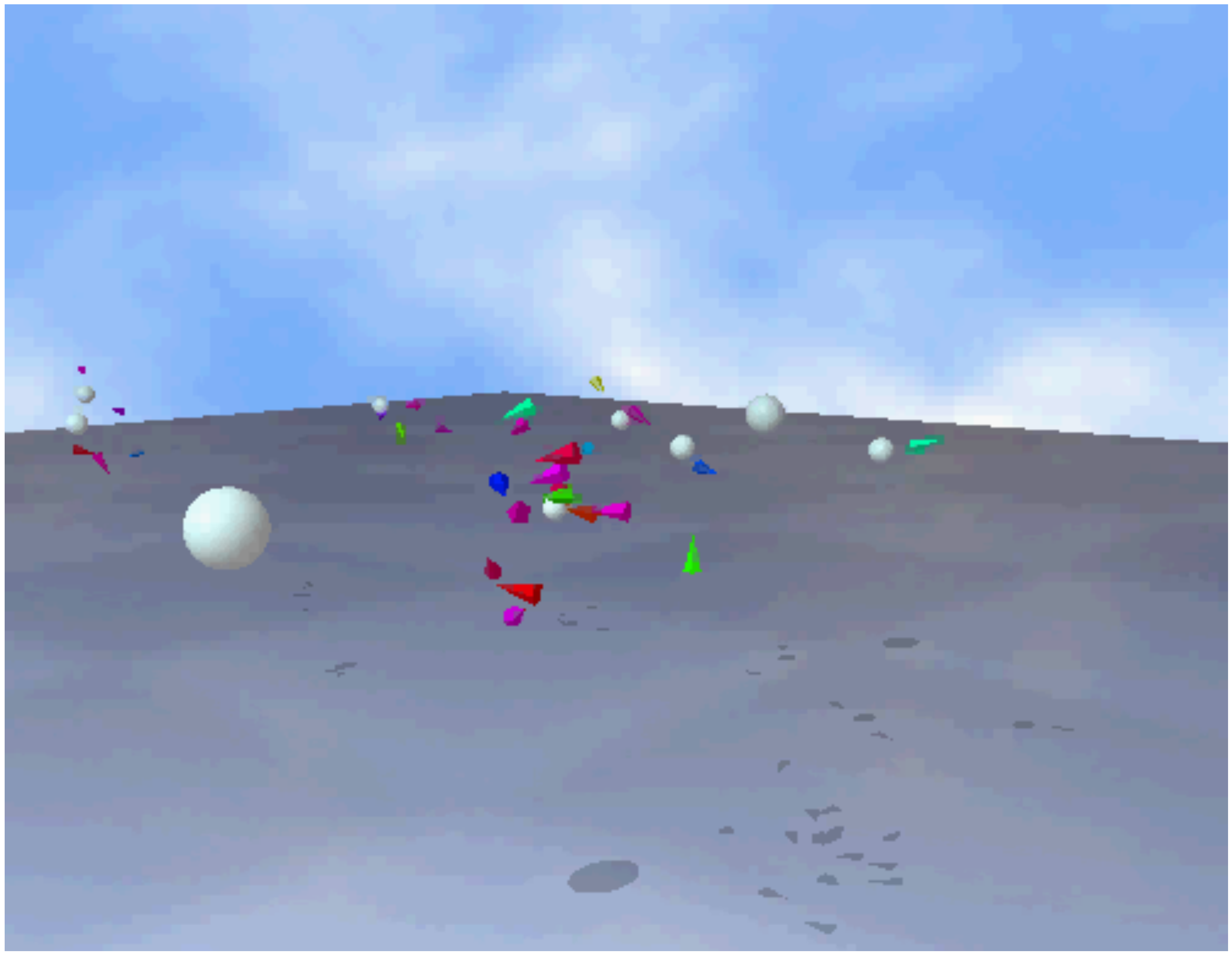
- A "swarm-like" agent environment with energy dynamics and conservation
- Behavior (including action, communication, energy sharing, and reproduction) controlled by evolved Push programs



Evolved Strategy

- Reckless goal-seeking + sharing
- Functional instructions of evolved code:

  (toFood feedOther myAge spawn randF)
- Accelerates directly toward nearest goal, feeds others, and turns random colors
- Evolved mutation regime: rate $\propto$ 1/age
- High goal coverage, low lifetimes



Sharing and Adaptation

- Sharing is with closest agent of similar/dissimilar color
- Recipient must have less energy than provider
- **Mutual**: Share only if recipient tried to share
- **Charity**: Share regardless of recipient's behavior
- **Waste**: All energy lost (a control)
- **No-op**: No energy changes (another control)
- Various settings of environmental stability parameter

Results (1,625 Runs)

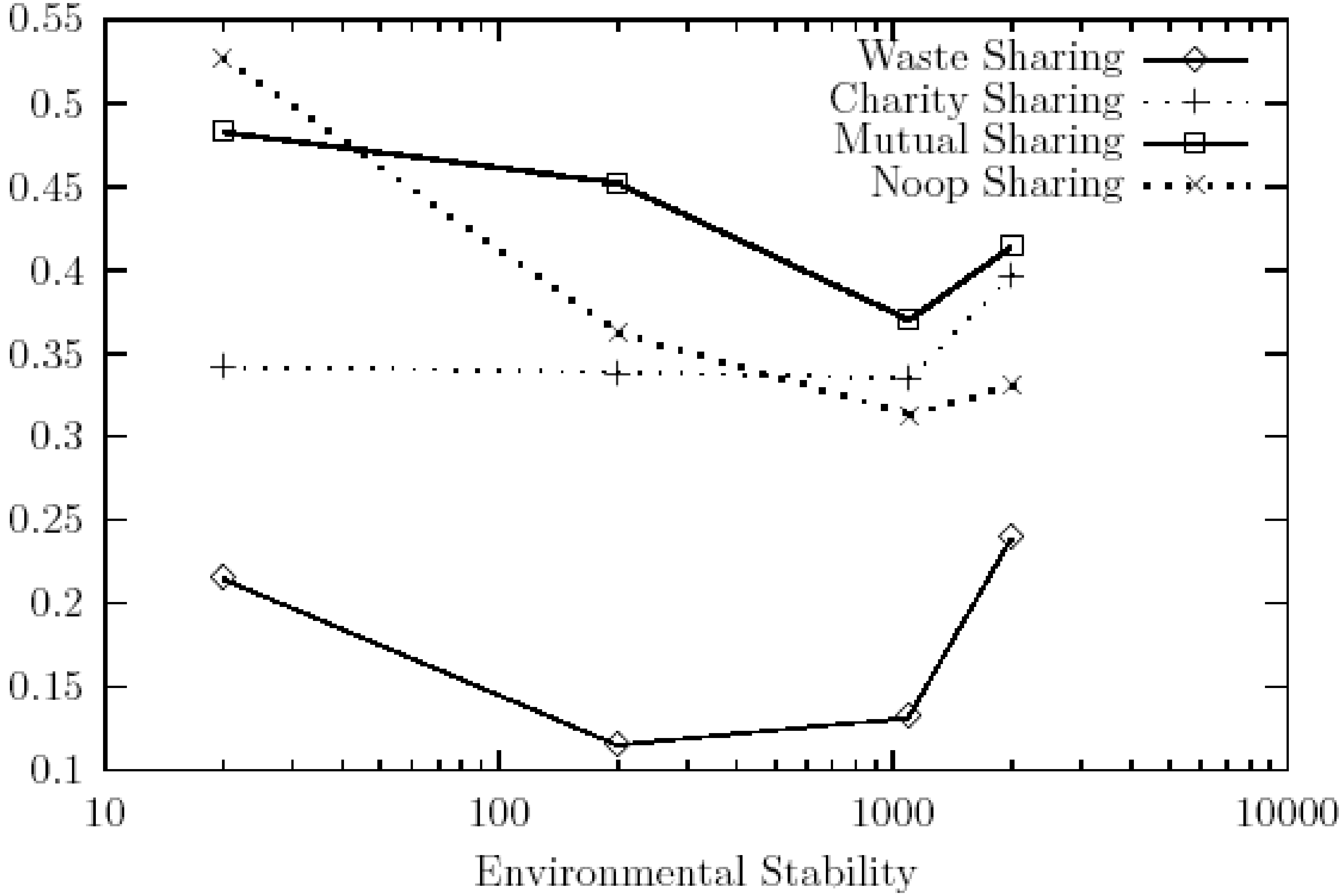


Fig. 4. Proportion of agents that share food (on the y axis) graphed vs. environmental (energy source) stability (on the x axis) for four sharing conditions (see text).

Conclusions

- Genetic programming can solve difficult and important problems
- Digital organisms can illuminate aspects of biological evolution
- Expressive program representations can enhance the utility of genetic programming and of digital organism systems
- Hints from nature can lead to abstractions that facilitate program evolution, as in the case of lexicase selection

Future

- Automatic programming of large-scale software systems
- Computational life forms demonstrating open-ended evolution and emergent evolutionary transitions
- Significant discoveries, produced by evolutionary processes, in many areas of science and engineering

Thanks

- David Clark, Moshe Sipper, and members of the Hampshire College Computational Intelligence Lab including Tom Helmuth, Jon Klein, and Karthik Kannappan for specific contributions to these slides.
- 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.