The Future of Genetic Programming

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

Outline

- Genetic programming
- Past and present
- Future
  - for solving problems
  - for understanding life
- Risks

Genetic Programming

- Evolution of computer programs

Genetic Programming

- Active evolution of computer programs

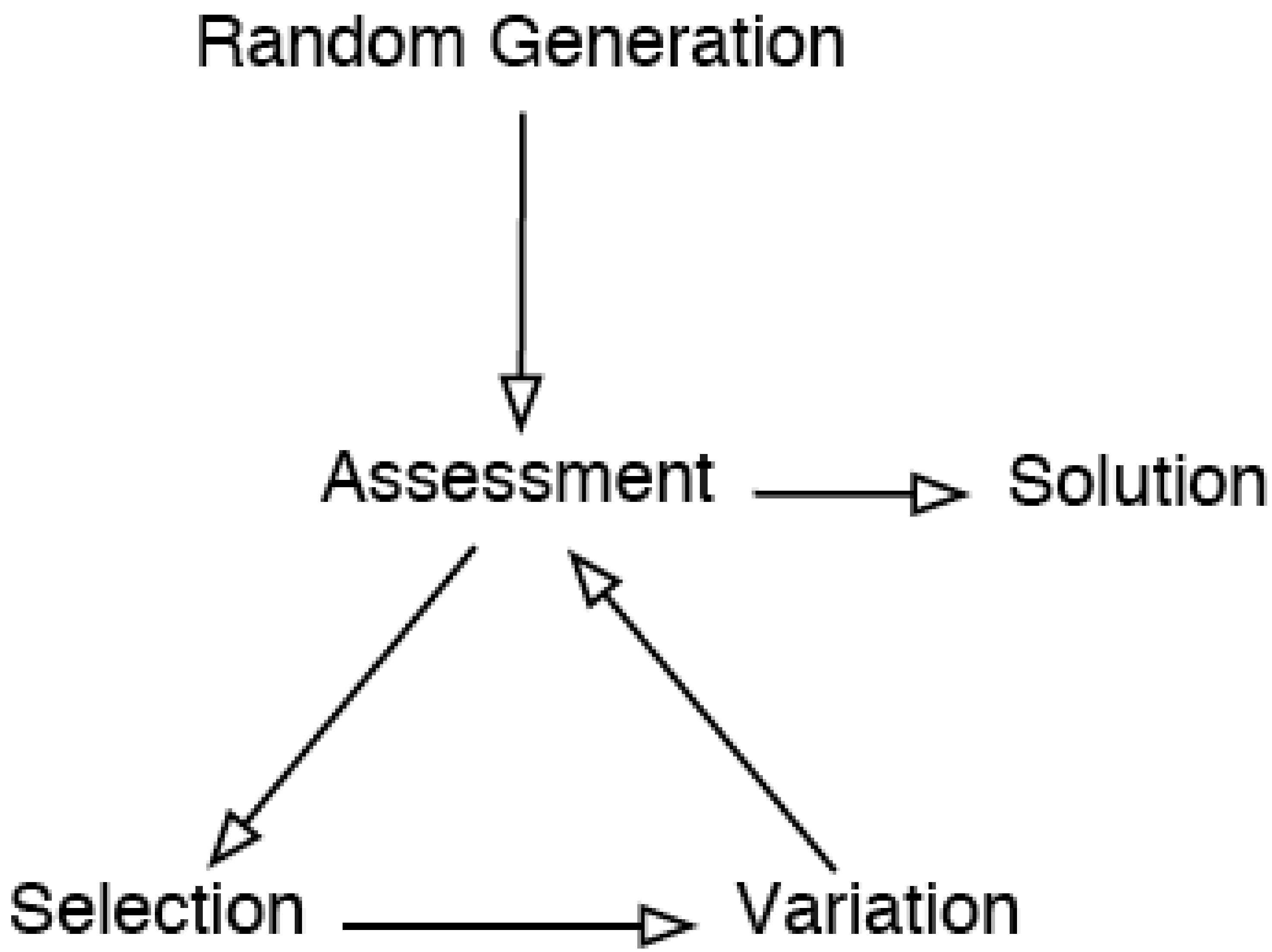
Genetic Programming

- Active evolution of computer programs
  - for solving problems
  - for producing software
  - for understanding life

Genetic Programming

- Active evolution of computer programs
  - for solving problems
  - for producing software
  - for understanding life

Genetic Algorithms



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, Ofria...).
- 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))))
```

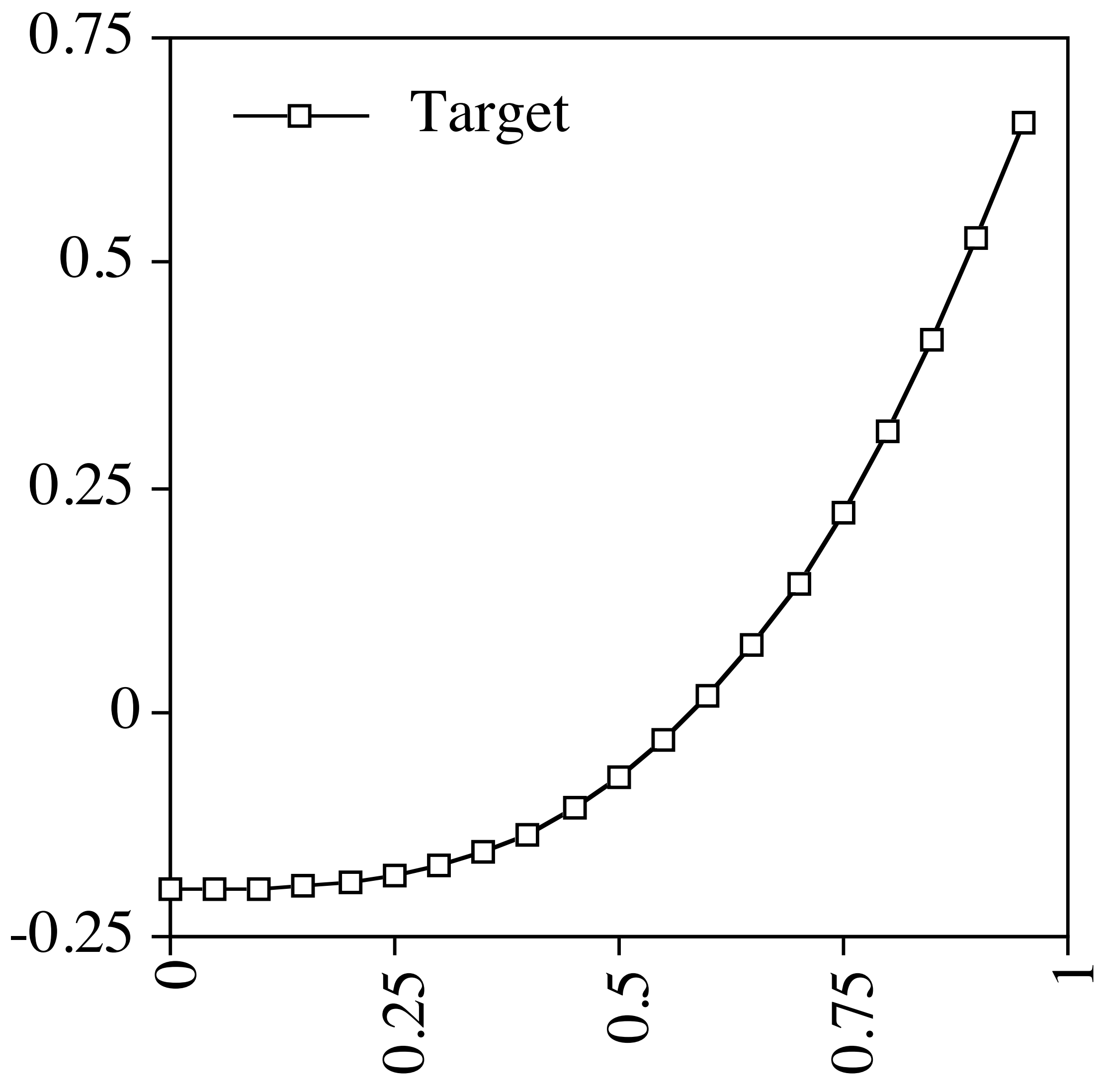
Symbolic Regression

- 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

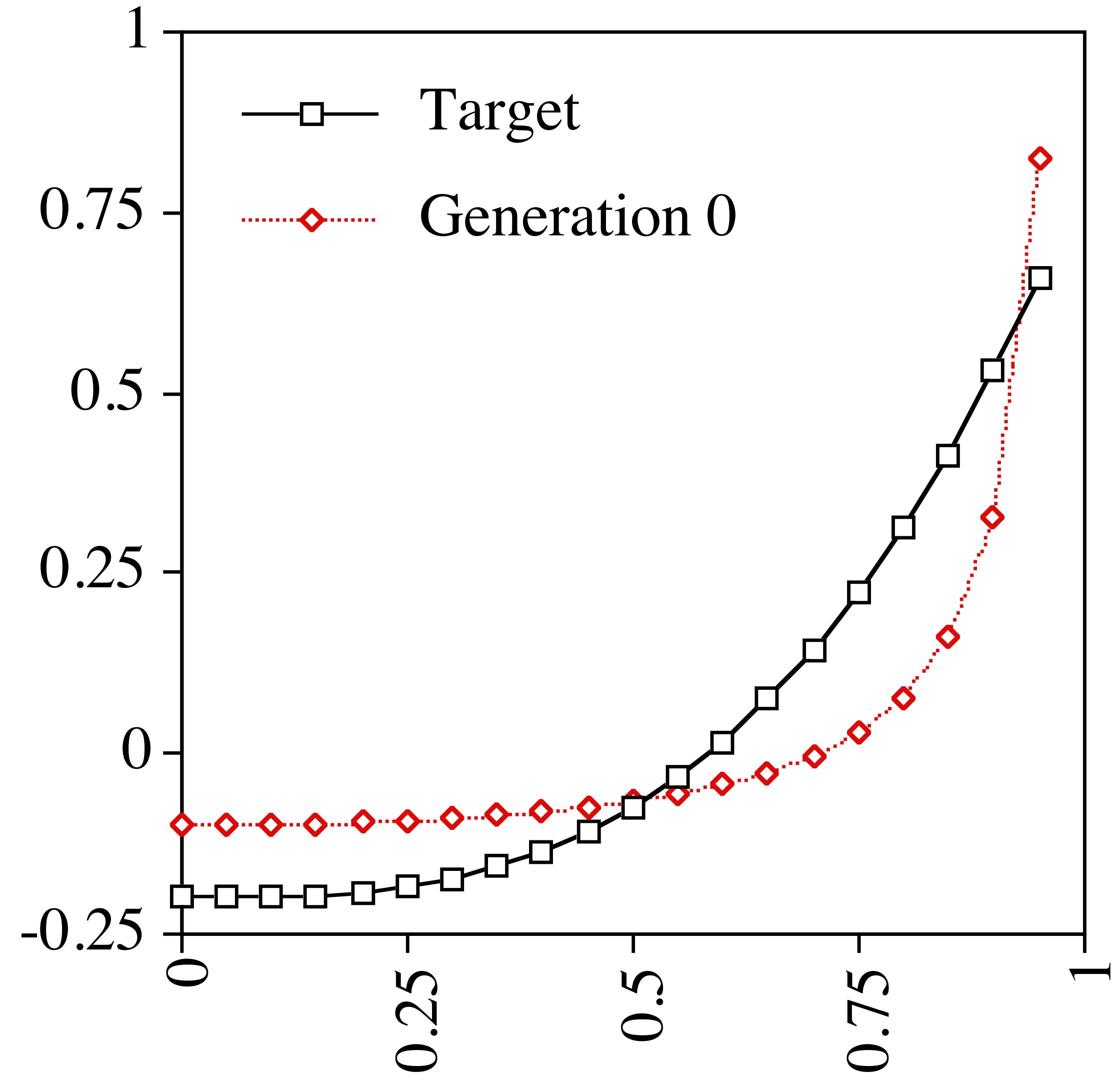
Maximum number of Generations: 51
Size of Population: 1000
Maximum depth of new individuals: 6
Maximum depth of new subtrees for mutants: 4
Maximum depth of individuals after crossover: 17
Fitness-proportionate reproduction fraction: 0.1
Crossover at any point fraction: 0.3
Crossover at function points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

Evolving $y = x^3 - 0.2$



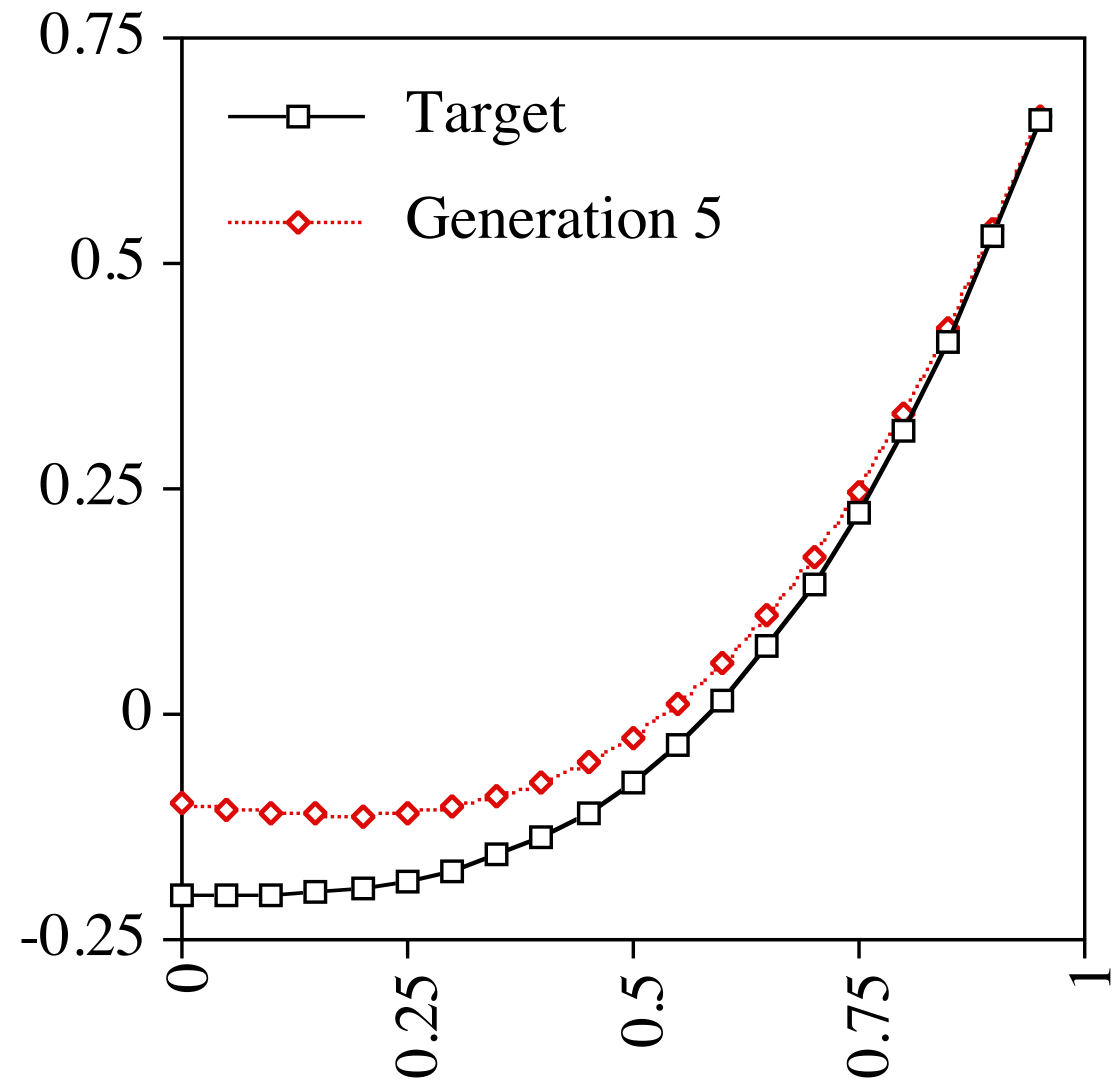
Best Program, Gen 0

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



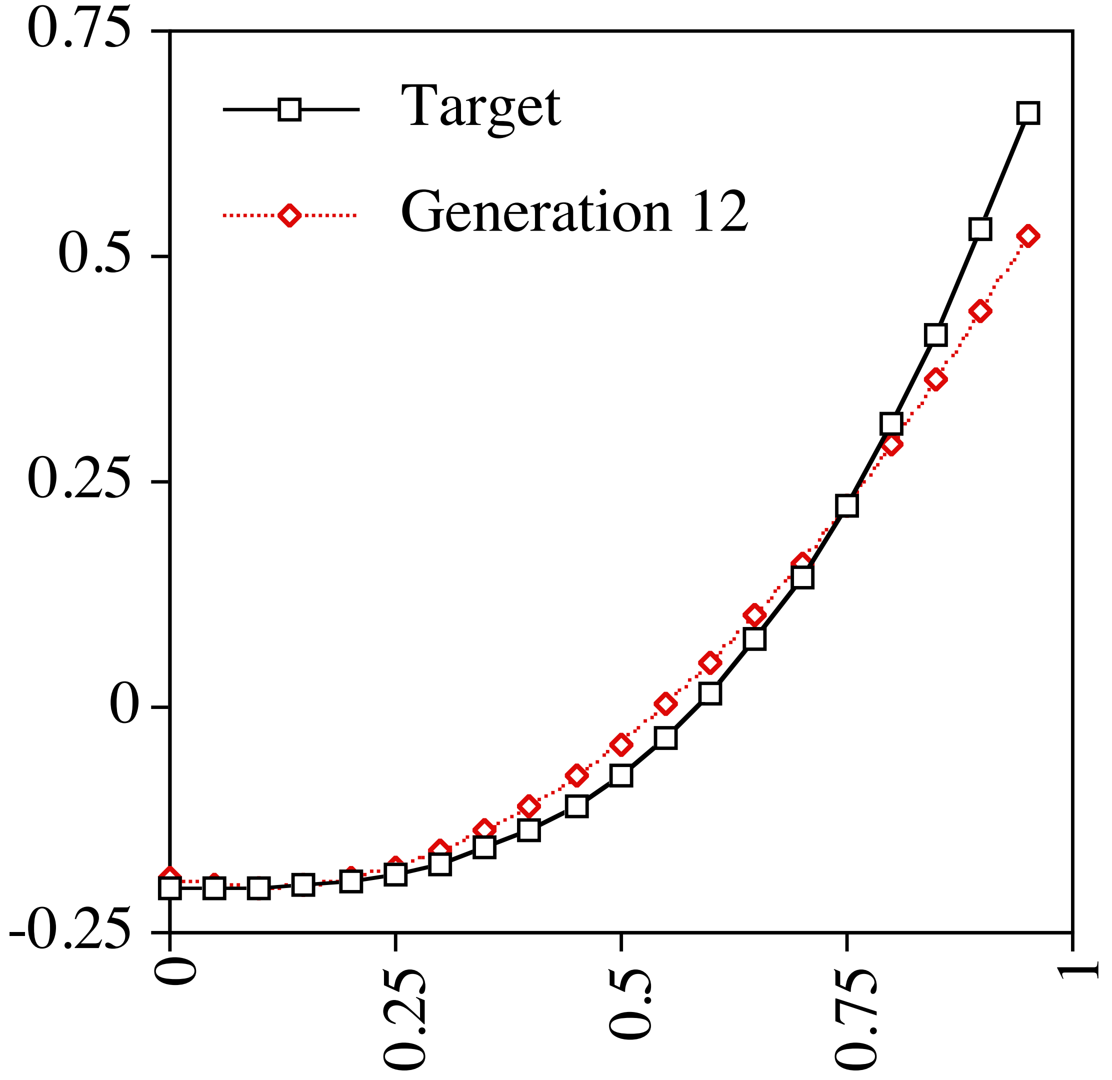
Best Program, Gen 5

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



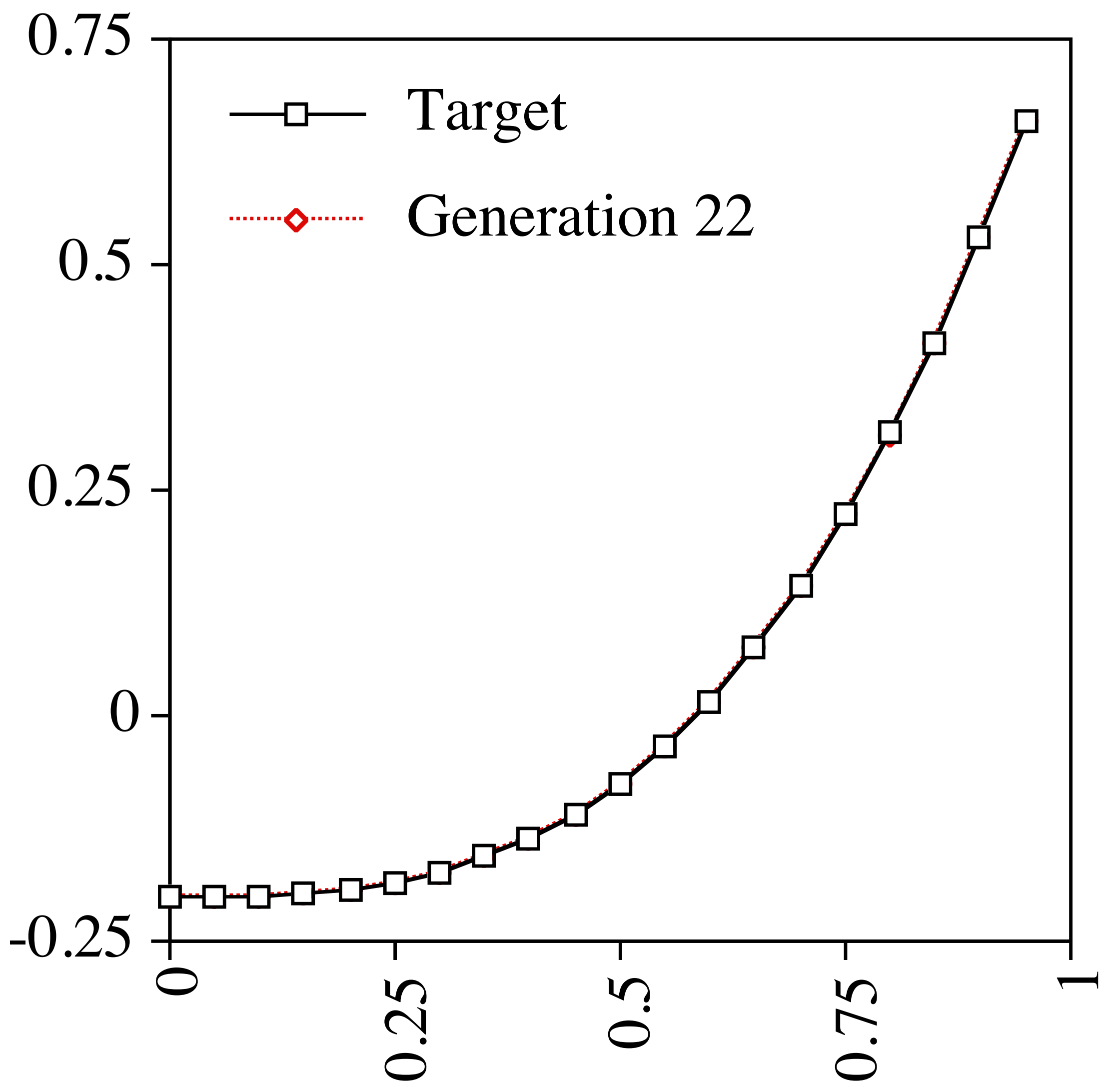
Best Program, Gen 12

```
(+ (- (- 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))
```



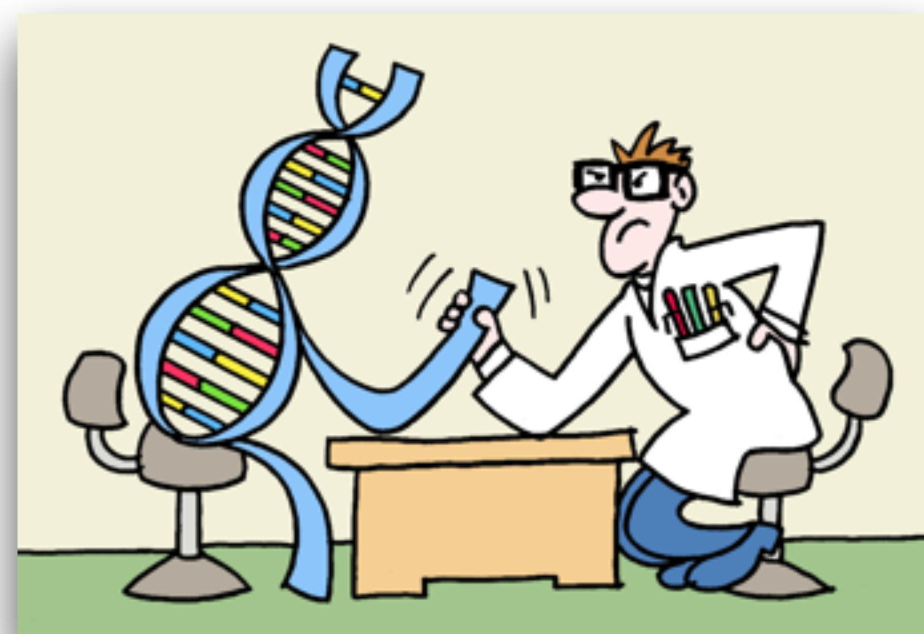
Best Program, Gen 22

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



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 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

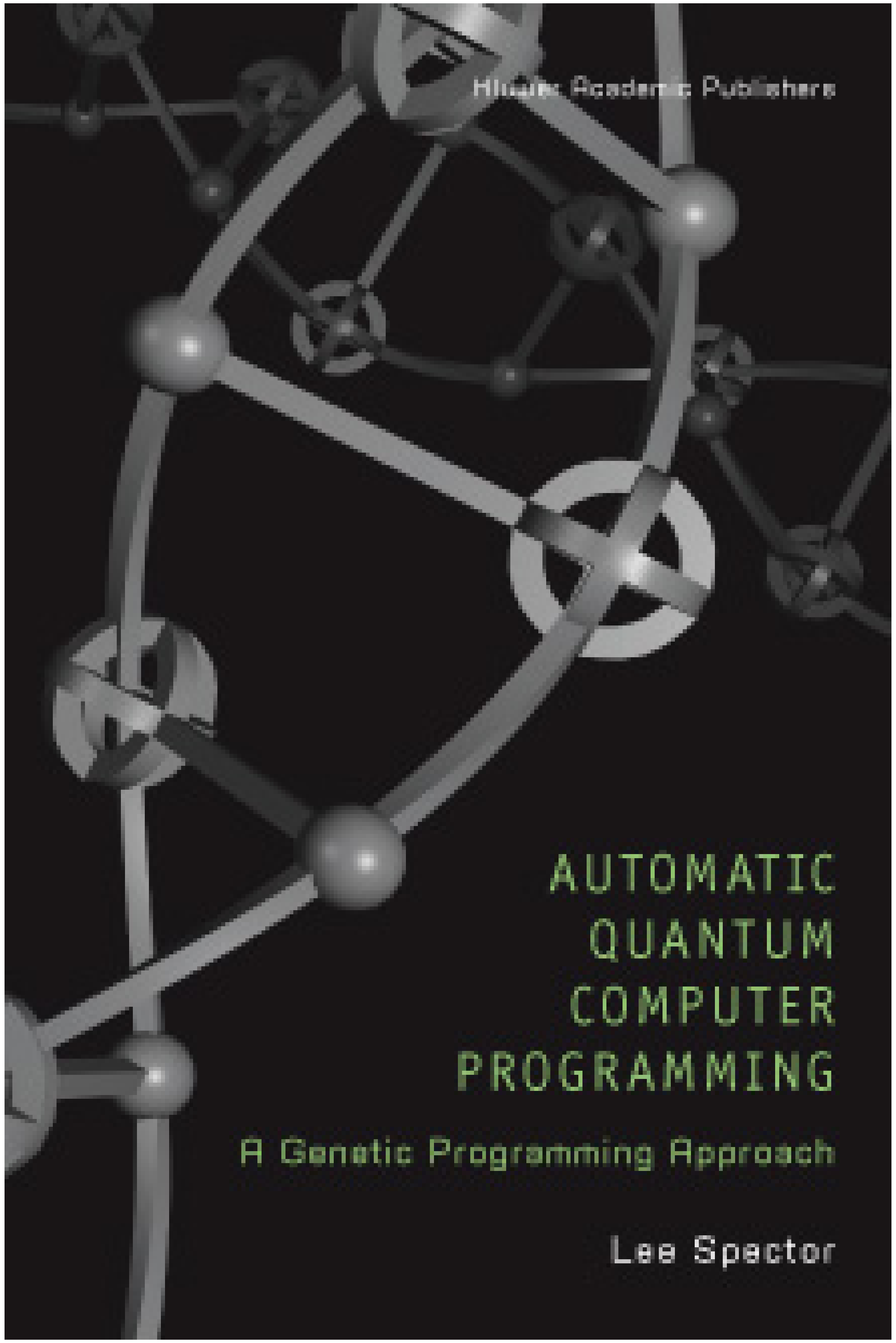
The Boston Globe

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."



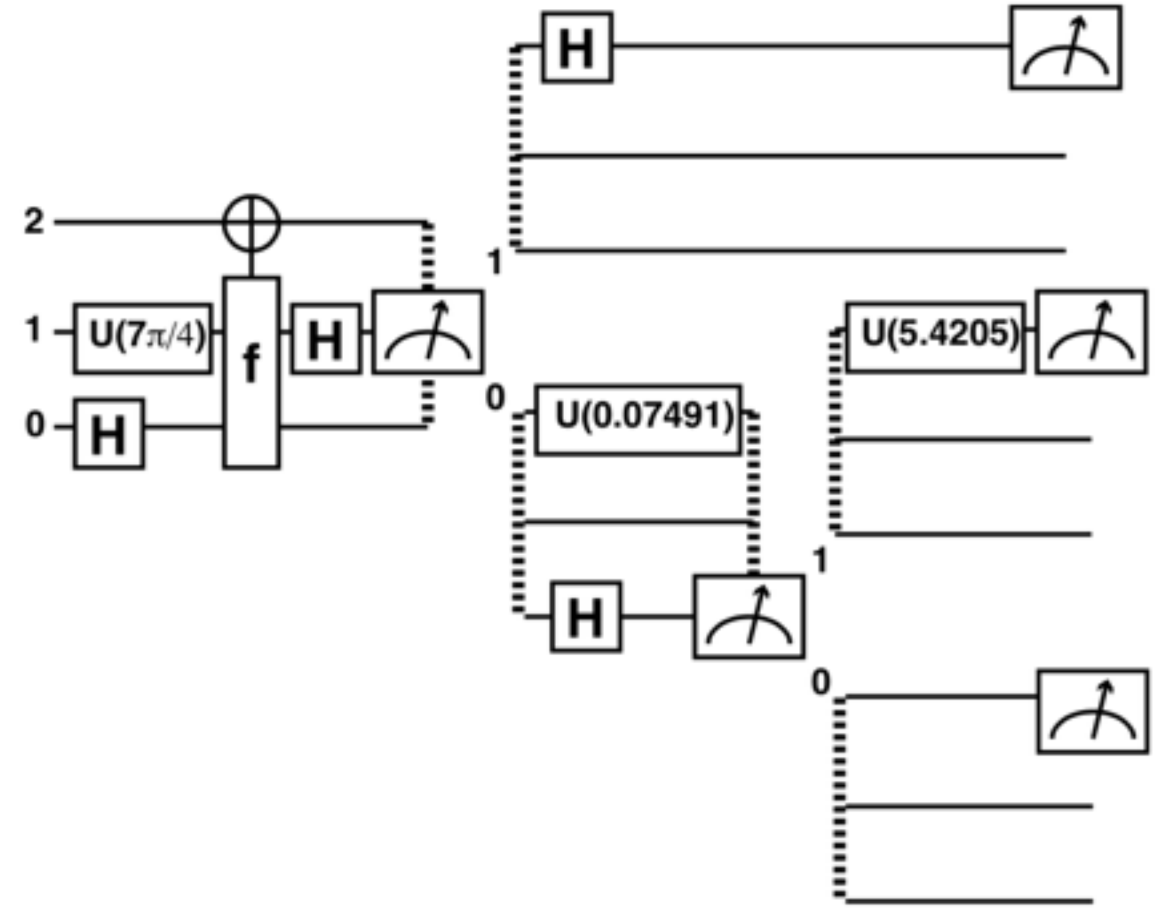


Figure 8.11. A gate array diagram for an evolved solution to the AND/OR oracle problem. The gate marked "f" is the oracle. The sub-diagrams on the right represent the possible execution paths following the intermediate measurements.

Humies 2004 GOLD MEDAL

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 2008
GOLD MEDAL

International Journal of Algebra and Computation
Vol. 23, No. 5 (2013) 1175–1205

DOI: 10.1142/S0218196713500227

World Scientific
www.worldscientific.com

EVOLUTION OF ALGEBRAIC TERMS 1: TERM TO TERM OPERATION CONTINUITY

DAVID M. CLARK

Mathematics Department
State University of New York at New Paltz
FOB E1, New Paltz, New York 12561, USA
clarkd@newpaltz.edu

Received 29 January 2012
Accepted 22 May 2013
Published 18 June 2013

Communicated by R. McKenzie

This study was inspired by recent successful applications of evolutionary computation to the problem of finding terms to represent arbitrarily given operations on a primal groupoid. Evolution requires that small changes in a term result in small changes in the associated term operation. We prove a theorem giving two readily testable conditions under which a groupoid must have this continuity property, and offer evidence that most primal groupoids satisfy these conditions.

Keywords: Evolutionary computation; term generation; term operation; primal algebras.

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

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

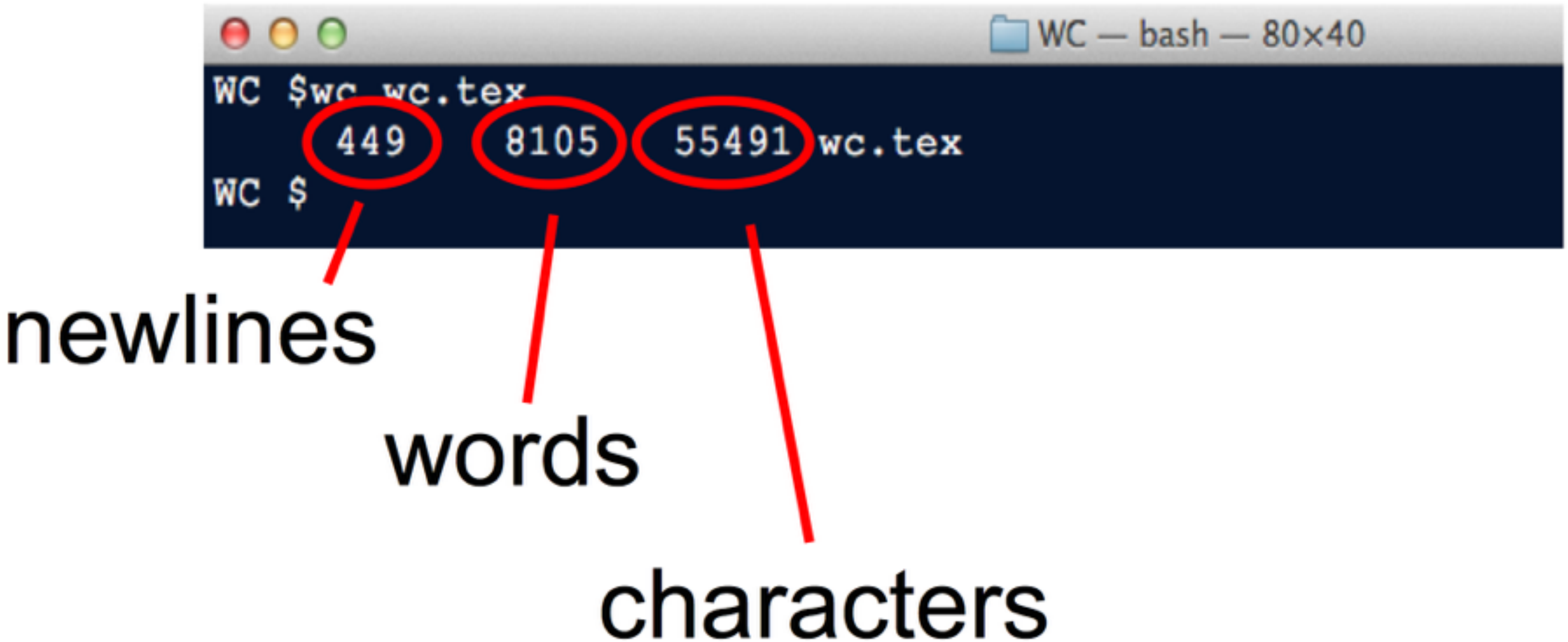
To select single parent:

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

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|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

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

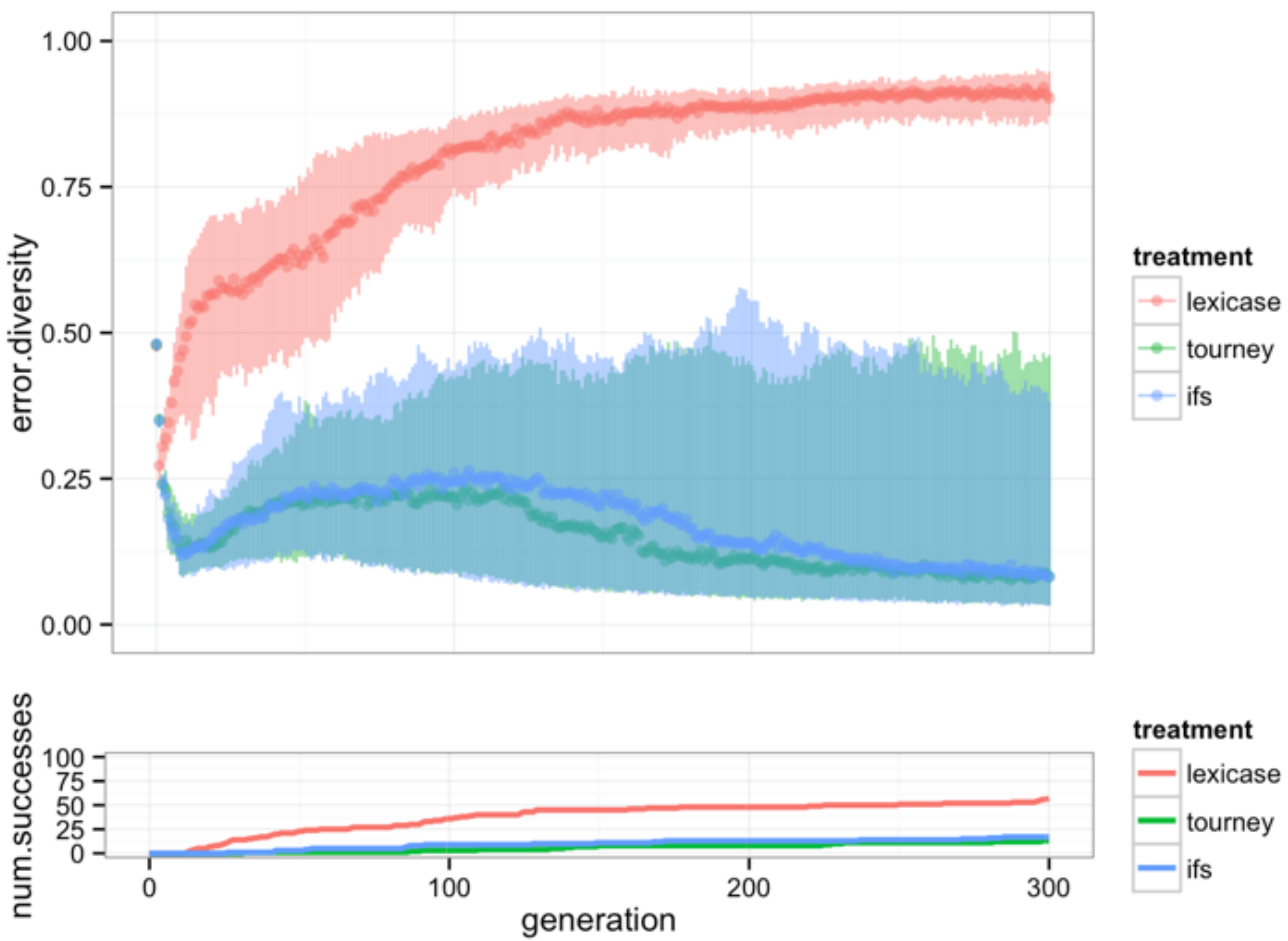
Table 3: Number of successful runs out of 100 for each setting, where "Tourn" is size 7 tournament selection, "IFS" is implicit fitness sharing with size 7 tournaments, and "Lex" is lexicase selection. For each problem, underline indicates significant improvement over the other two selection methods at $p < 0.05$ based on a pairwise chi-square test with Holm correction [12], or a pairwise Fisher's exact test with Holm correction if any number of successes is below 5 [10]. The "Size" column indicates the smallest size of any simplified solution program

Problem	Tourn	IFS	Lex	Size
Number IO	68	72	<u>98</u>	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	<u>51</u>	9
String Differences	0	0	0	
Even Squares	0	0	2	37
Wallis Pi	0	0	0	
String Lengths Backwards	7	10	<u>66</u>	9
Last Index of Zero	8	4	<u>21</u>	5
Vector Average	14	13	16	7
Count Odds	0	0	8	7
Mirror Image	46	64	<u>78</u>	4
Super Anagrams	0	0	0	
Sum of Squares	2	0	6	7
Vectors Summed	0	0	1	11
X-Word Lines	0	0	<u>8</u>	15
Pig Latin	0	0	0	
Negative To Zero	10	8	<u>45</u>	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	<u>98</u>	81	8
Syllables	1	7	18	14
Problems Solved	13	13	22	

Plot Medians and Quartiles

RSWN (Replace Space with Newline)

```
add_generational_success_counts_plot(data_rswn, plot_diversity_medians_and_quartiles(data_rswn))
```



Life involves the evolution of programs

Life i s the
evolution of programs

Life is the
evolution of programs

Digital Organisms

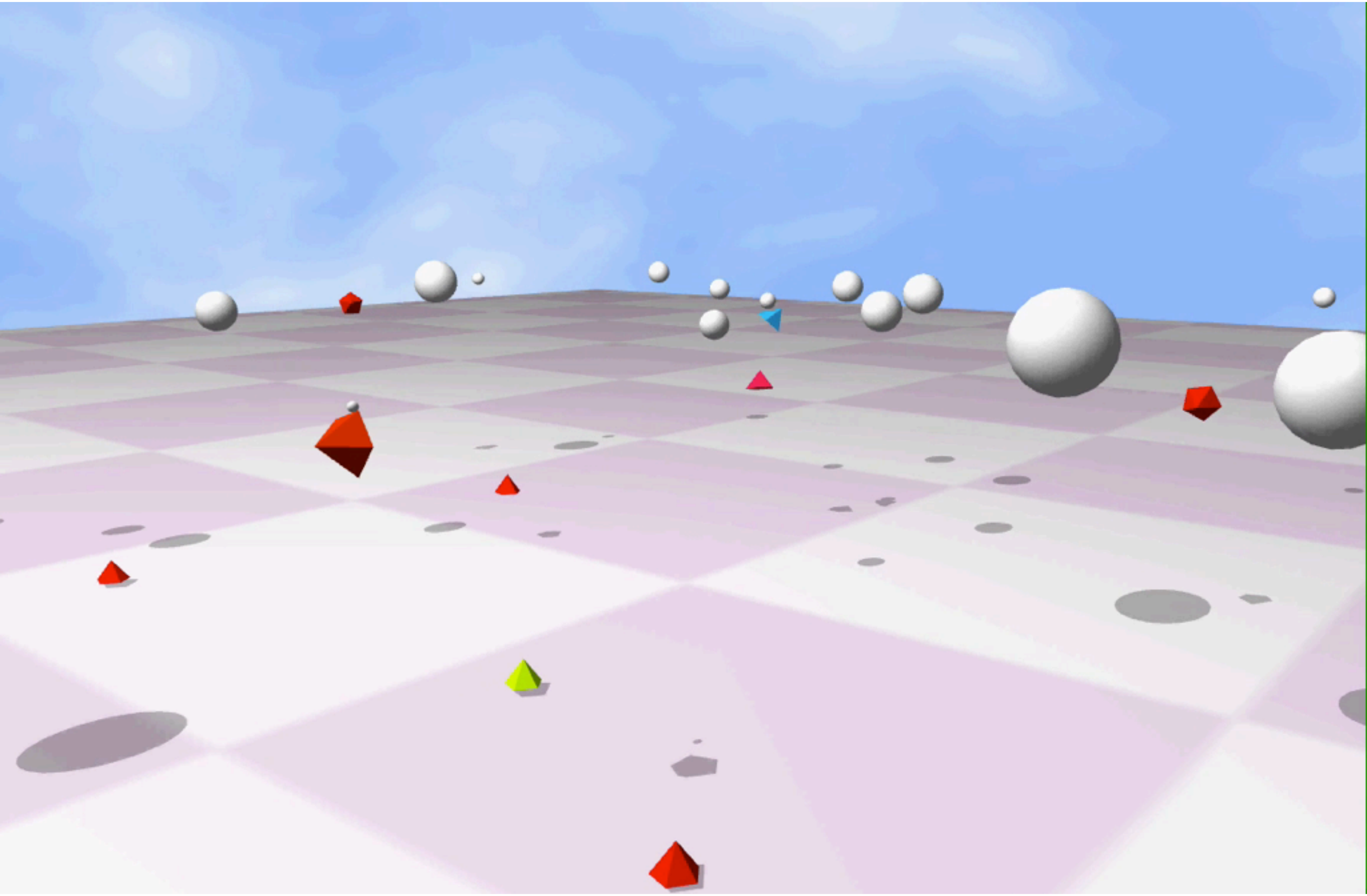
- For the study of general principles of living systems
- Populations of individuals that act locally in environments
- Explore, in silico, key aspects of evolutionary processes
- Core War, Tierra, Avida, Echo, Polyworld, Framsticks, ...

Autoconstructive Evolution

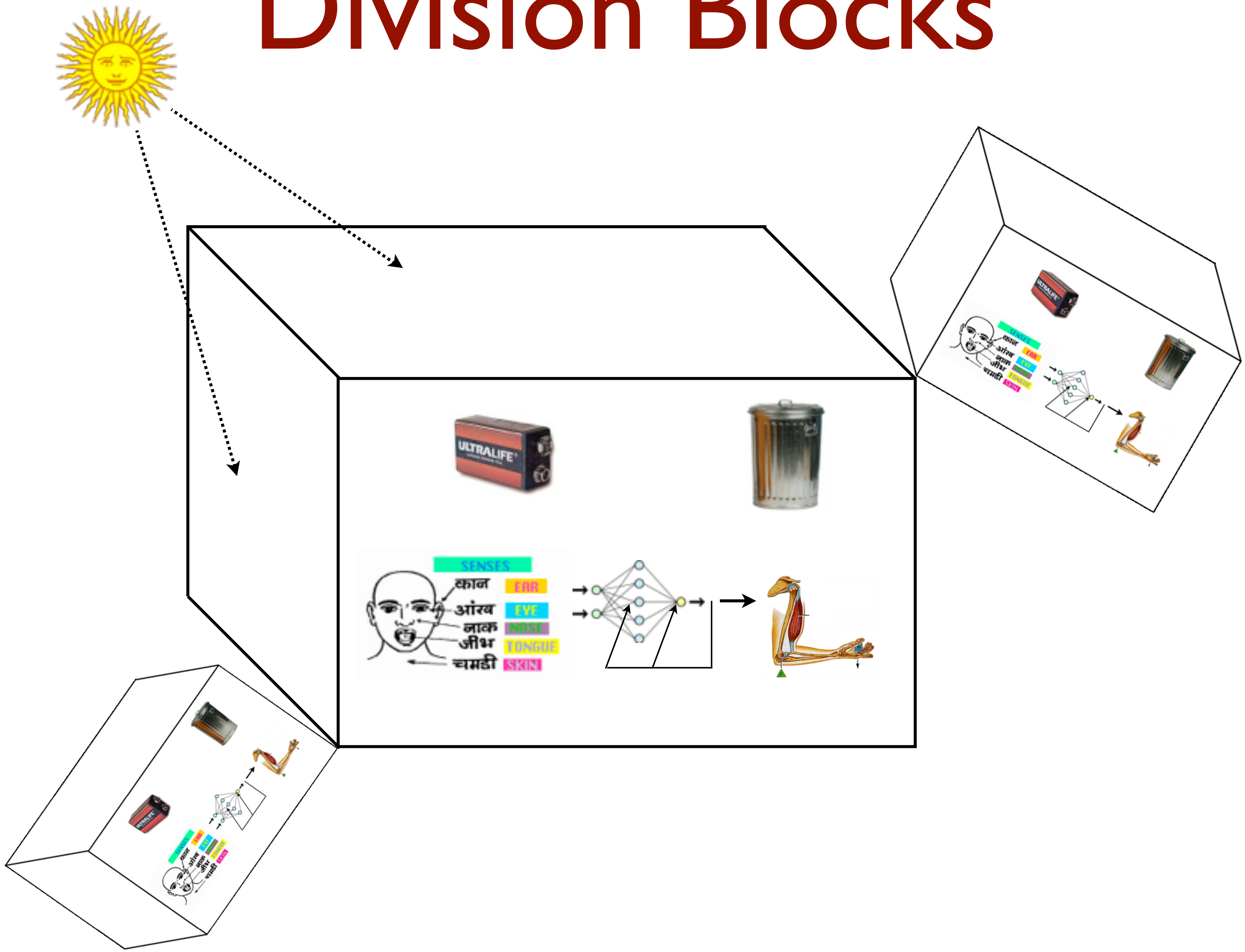
- Individual programs make their own children, with endogenous variation
- Hence they control their own mutation rates and methods, sexuality, reproductive timing, etc.
- The machinery of reproduction and diversification (i.e., the machinery of evolution) evolves
- Requires expressive program representations (like Push)

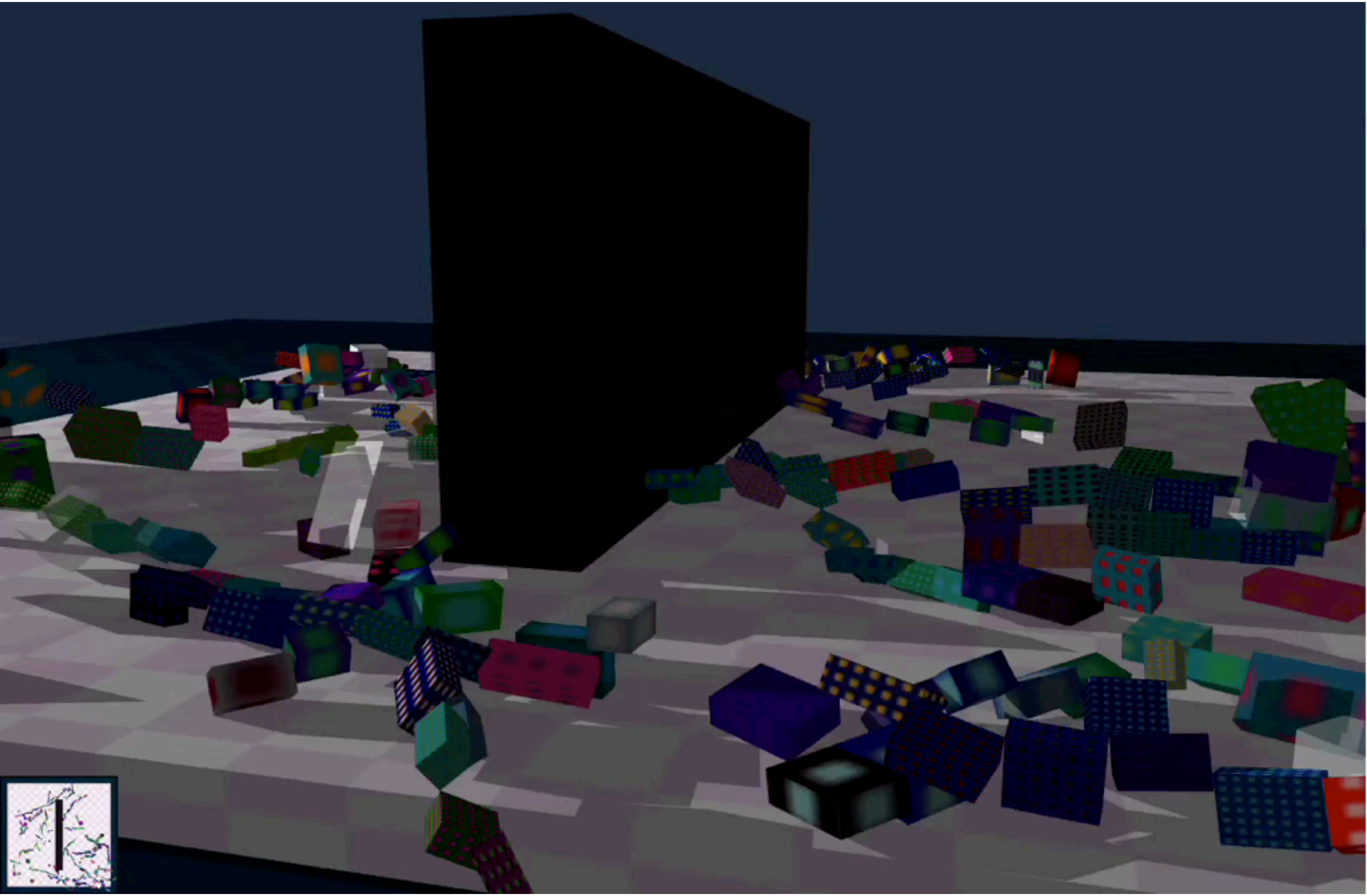
SwarmEvolve 2

- A "swarm-like" agent environment with energy dynamics and conservation
- Behavior (including action, communication, energy sharing, and reproduction) controlled by evolved Push programs
- Supports exploration of relations between adaptation and various kinds of resource sharing, under a range of environmental settings

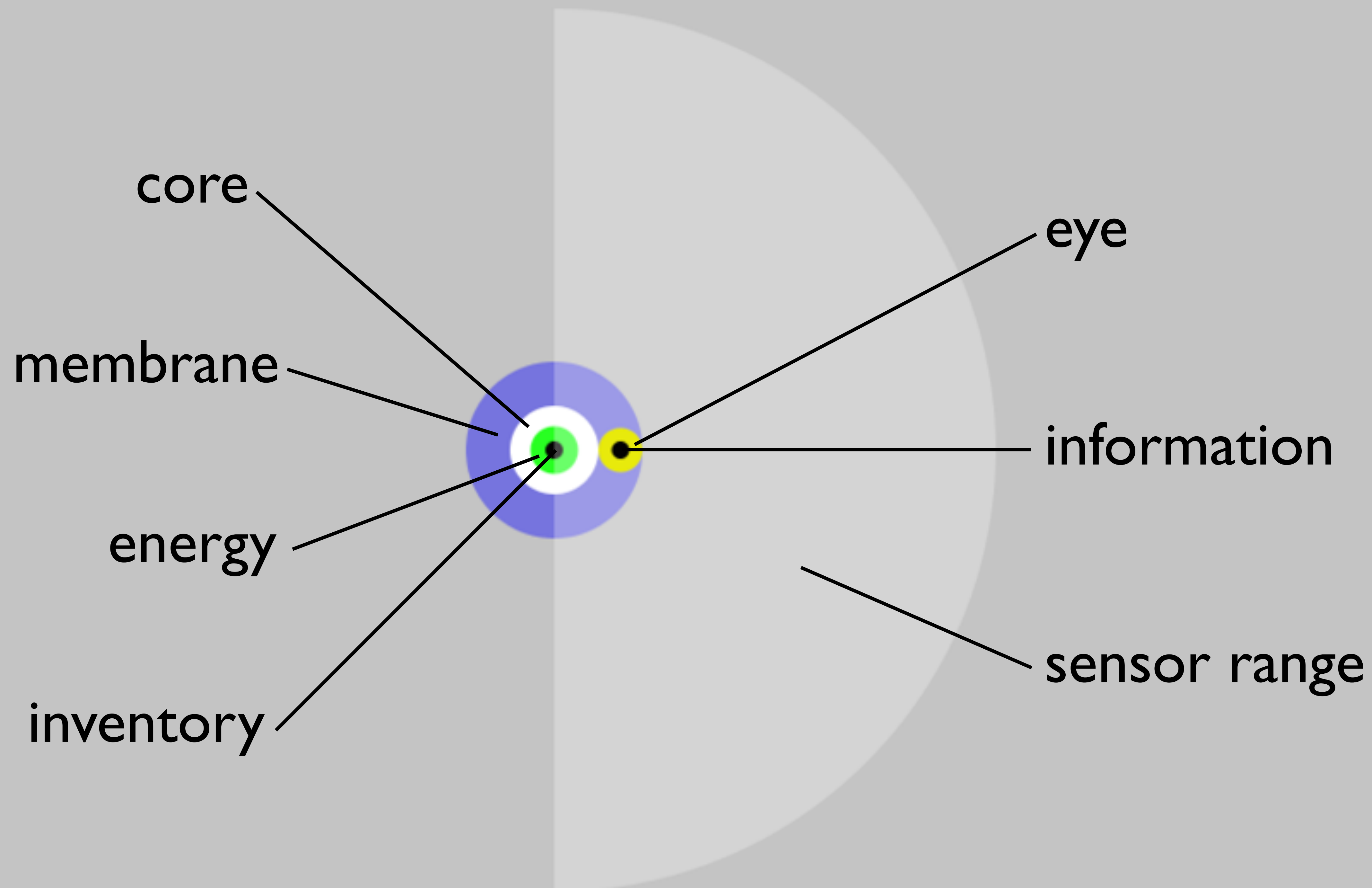


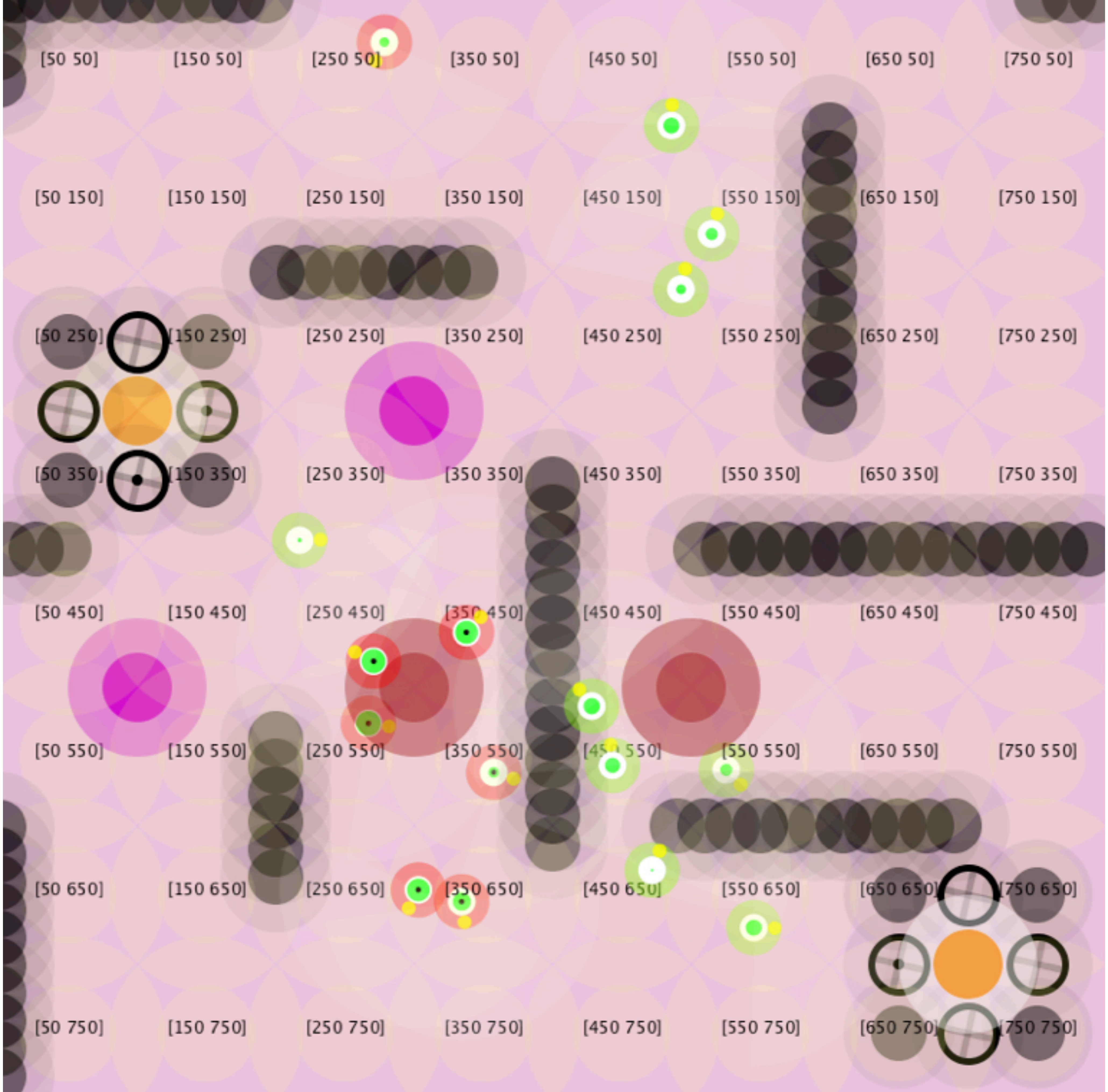
Division Blocks

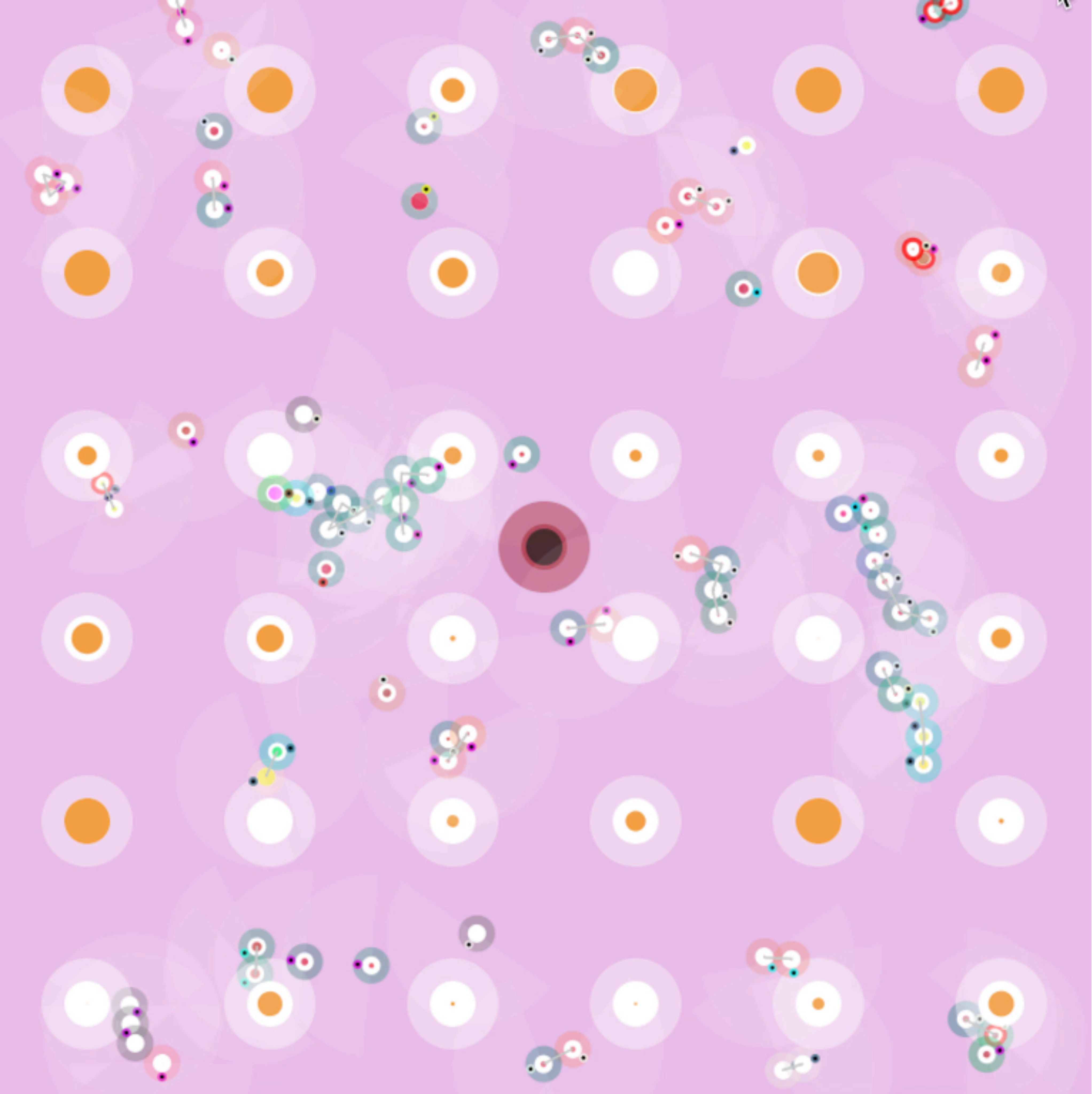




Pucks


core
membrane
energy
inventory
eye
information
sensor range


[50 50]
[150 50]
[250 50]
[350 50]
[450 50]
[550 50]
[650 50]
[750 50]
[50 150]
[150 150]
[250 150]
[350 150]
[450 150]
[550 150]
[650 150]
[750 150]
[50 250]
[150 250]
[250 250]
[350 250]
[450 250]
[550 250]
[650 250]
[750 250]
[50 350]
[150 350]
[250 350]
[350 350]
[450 350]
[550 350]
[650 350]
[750 350]
[50 450]
[150 450]
[250 450]
[350 450]
[450 450]
[550 450]
[650 450]
[750 450]
[50 550]
[150 550]
[250 550]
[350 550]
[550 550]
[650 550]
[750 550]
[50 650]
[150 650]
[250 650]
[350 650]
[450 650]
[550 650]
[650 650]
[750 650]
[50 750]
[150 750]
[250 750]
[350 750]
[450 750]
[550 750]
[650 750]
[750 750]



Genetic Programming

- Active evolution of computer programs
    - for solving problems
    - for producing software
    - for understanding life

Prospects

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

Risks

- Technology that we don't understand
- Human competitive technology

Thanks

- David Clark, Moshe Sipper, and members of the Hampshire College Computational Intelligence Lab including Tom Helmuth, Bill La Cava, 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.

The Future of Genetic Programming

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