The Future of Genetic Programming

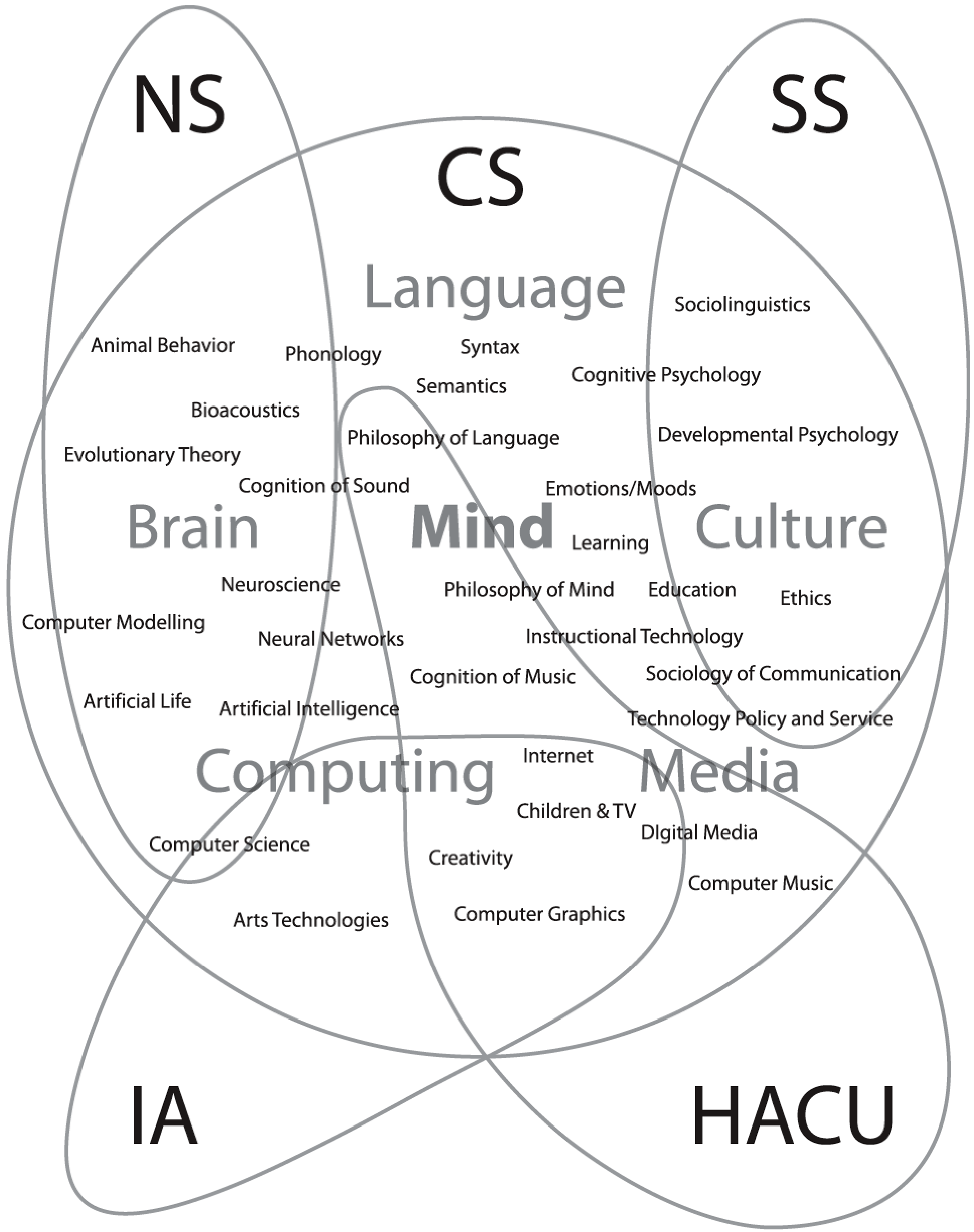
Lee Spector
Cognitive Science, Hampshire College
Computer Science, UMass Amherst
http://hampshire.edu/lspector

Outline

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

Background

- B.A., Oberlin College: Philosophy, Music/Art Technology
- Ph.D., U. Maryland, College Park: Computer Science (AI)
- Professor of Computer Science and Director, Institute for Computational Intelligence, Hampshire College.
  Past: Dean, Cognitive Science; MacArthur Chair; Co-chair, Re-visioning Committee; Faculty Trustee; Co-director of the Design, Art and Technology program; Member, Governance Task Force, Educational Policy Committee, etc.
- Adjunct Professor of Computer Science, U. Massachusetts, Amherst
- Editor-in-Chief, *Genetic Programming and Evolvable Machines* (Springer)
- Executive Committee, ACM-SIGEVO



CS Connections Sketch, January 27, 2004

Grants

- Google: CS Engagement Award, Programming for Science
- NSF: Human-Competitive Evolutionary Computation
- NSF: Four College Biomath Consortium
- NSF: Evolution of Robustly Intelligent Computational Systems
- Sherman Fairchild Foundation: Design, Art, and Technology
- NSF CreativeIT: The Computational Creativity Curriculum
- NSF Director's Award for Distinguished Teaching Scholars: Open-Ended Evolution in Visually Rich Virtual Worlds
- NSF, MRI/RUI: Acquisition of Instrumentation for Research in Genetic Programming, Quantum Computation, and Distributed Systems
- DARPA Agent Based Computing: Multi-type, Self-adaptive Genetic Programming for Complex Applications
- NSF Learning and Intelligent Systems: Inquiry-Based Science Education: Cognitive Measures and Systems Support

Not GP

- What, if anything, is a Wolf?
- Planning, Neuropsychology, and Artificial Intelligence: Cross-Fertilization
- Group size, individual role differentiation and effectiveness of cooperation in a homogeneous group of hunters
- Behind every innovative solution lies an obscure feature
- Wolf-pack (Canis lupus) hunting strategies emerge from simple rules in computational simulations
- Genetic Stability and Territorial Structure Facilitate the Evolution of Tag-mediated Altruism
- Hierarchy Helps it Work That Way
- Partial and total-order planning: evidence from normal and prefrontally damaged populations

Genetic Programming

- Evolution of computer programs

Genetic Programming

- Active evolution of computer programs

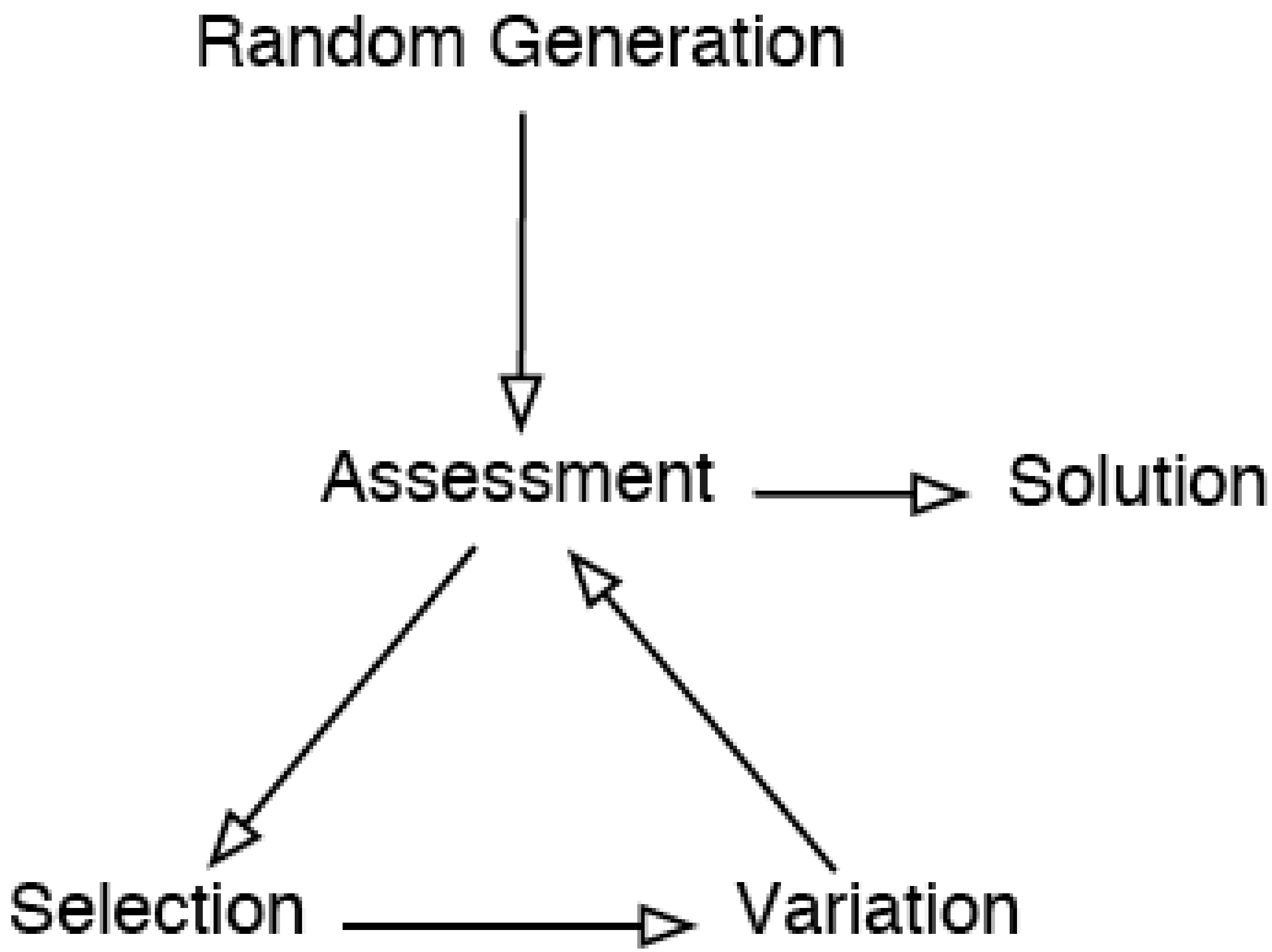
Genetic Programming

- Active evolution of computer programs
  - for solving problems
  - for advancing science and technology
  - for understanding life

Genetic Programming

- Active evolution of computer programs
  - for solving problems
  - for advancing science and technology
  - for understanding life

Genetic Algorithms



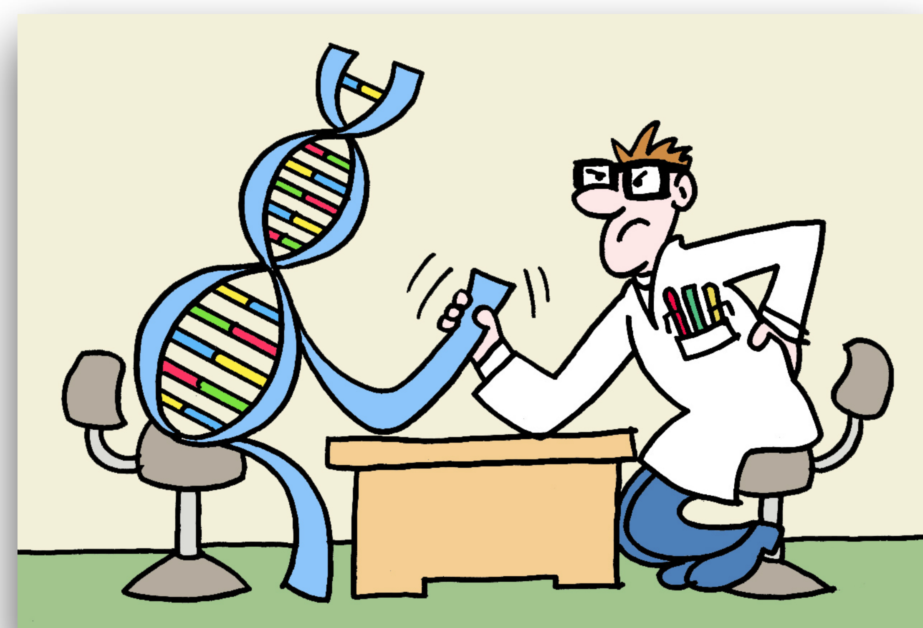
Genetic Programming

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

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

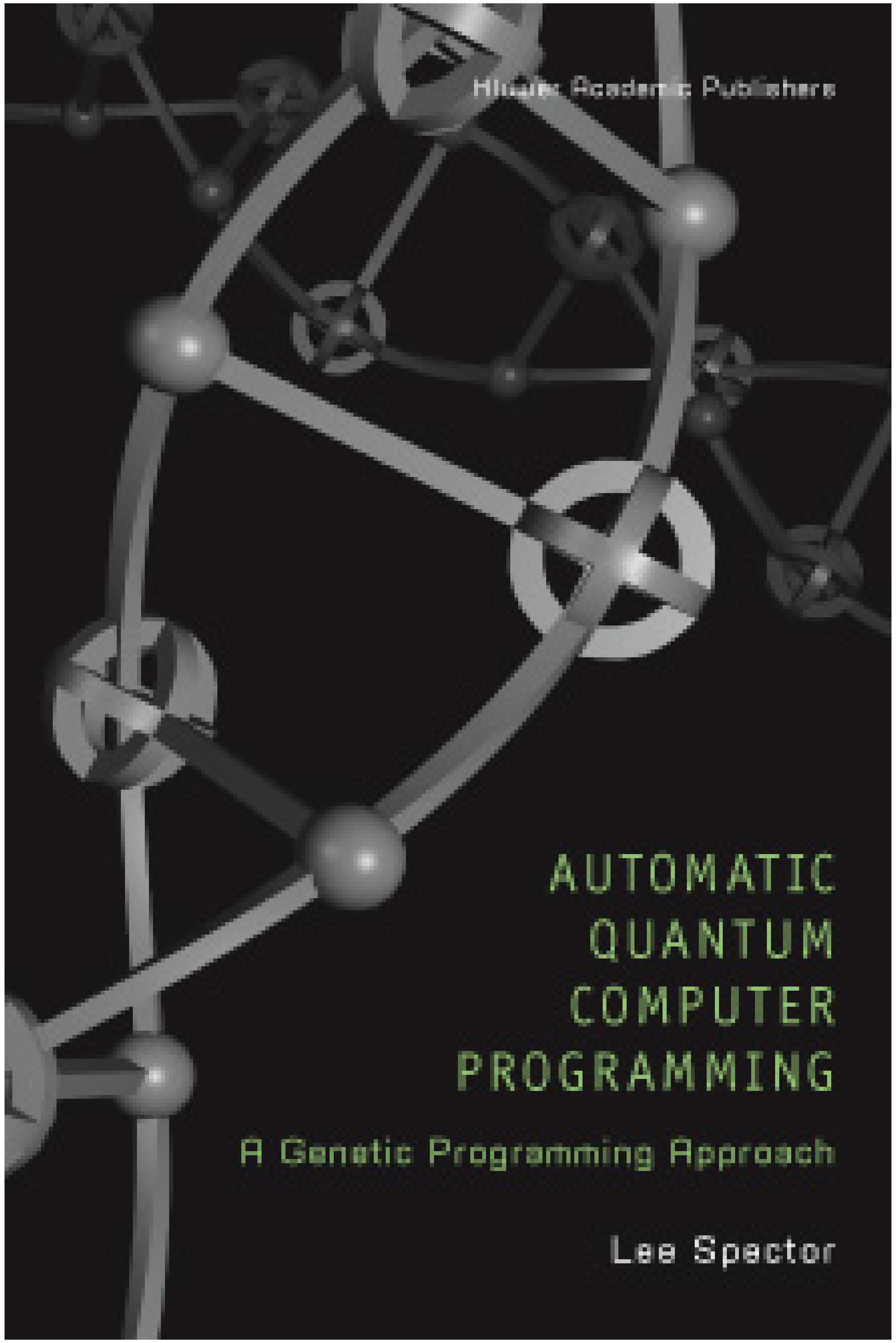
The Boston Globe

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."



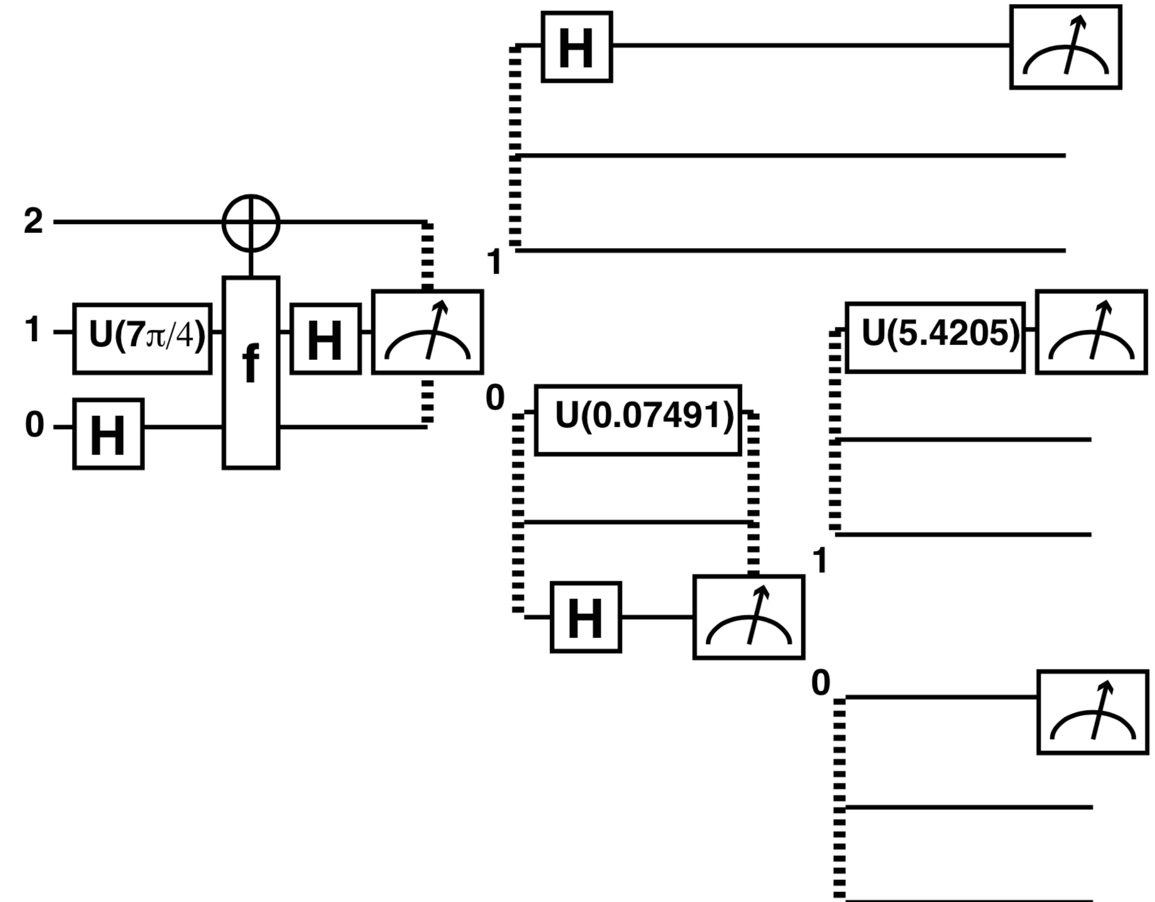


Figure 8.11. A gate array diagram for an evolved solution to the AND/OR oracle problem. The gate marked "f" is the oracle. The sub-diagrams on the right represent the possible execution paths following the intermediate measurements.

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

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
- Previous best methods are exponentially slow or produce enormous terms
- Want to be able to find small terms quickly

Significance, Time

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

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

DOI: 10.1142/S0218196713500227

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.

To the Future

- Expressive program representations (Push)
- Flexible genetic/epigenetic variation (Plush)
- Well-informed selection (Lexicase)

$\Rightarrow$ Automation of human programming

Program Representations

- Should facilitate the expression of programs that use:
  - Arbitrary data structures
  - Arbitrary control structures
  - Modularity
- Should facilitate the development of effective (and ideally evolvable) genetic/epigenetic variation operators

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

Plush

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

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

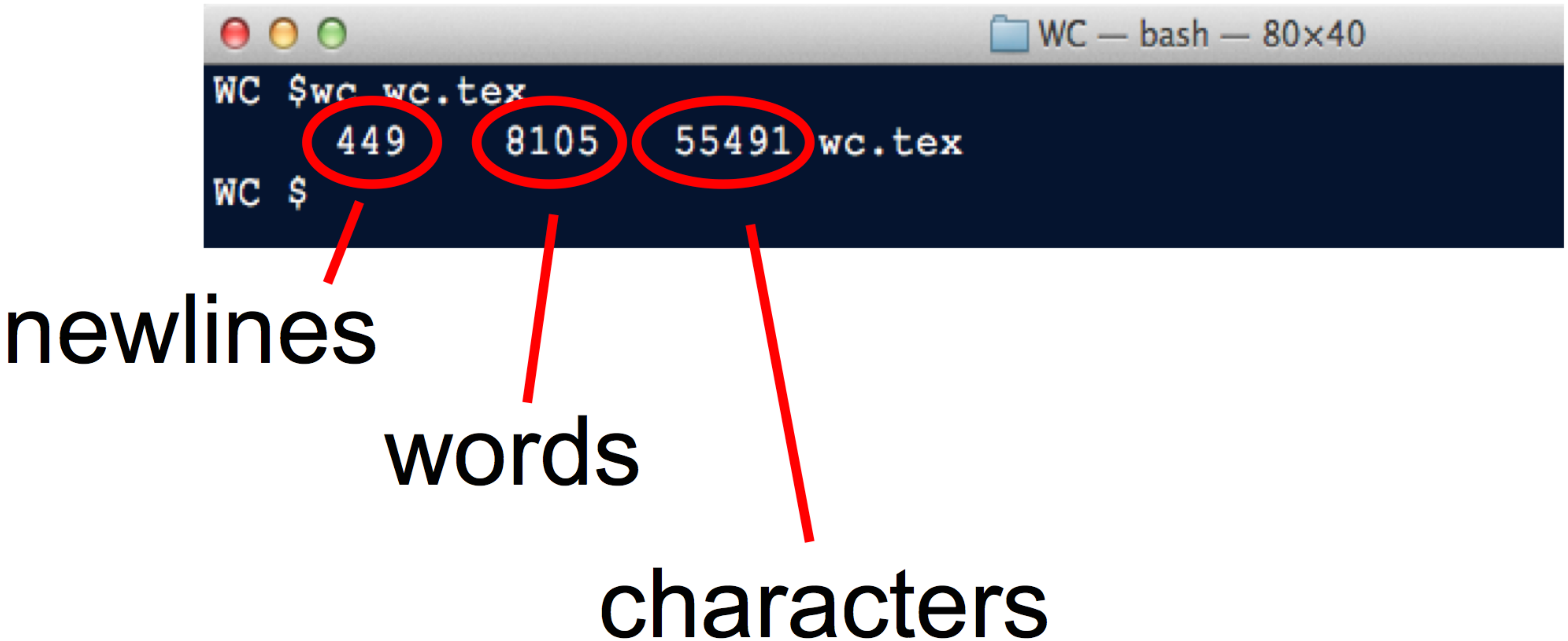
To select single parent:

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_stack-depth`, `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`, `inte-ger_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_frominte-ger`, `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

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 pro-

example, we can imagine a problem involving control of a simulated wind turbine in which some test cases focus on performance in low wind conditions while others focus on performance in high wind conditions. It may not be possible to optimize performance on all of these test cases simultaneously, and some sort of compromise may therefore be necessary. Many common parent selection approaches, such as tournament selection, introduce compromises between test cases by aggregating the performance of an individual on those test cases into a single fitness value. The method of compromise may be as simple as summing the test case errors, or their squares, into a single error value; more complex methods such as implicit fitness sharing [2] dynamically weight test cases based on population statistics before aggregating them.

By contrast, we wish to consider what we call "uncompromising" problems: problems for which any acceptable solution must perform as well on each test case as it is possible to perform on that test case; that is, an uncompromising problem is a problem for which it is not acceptable for a solution to perform sub-optimally on any one test case in exchange for good performance on others. More formally, consider a problem defined by the set of test cases T where the set of programs in the search space is P and $p_j(t_i)$ is the error produced by program $p_j \in P$ on test case $t_i \in T$ with lower error being better. This problem is *uncompromising* if a

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

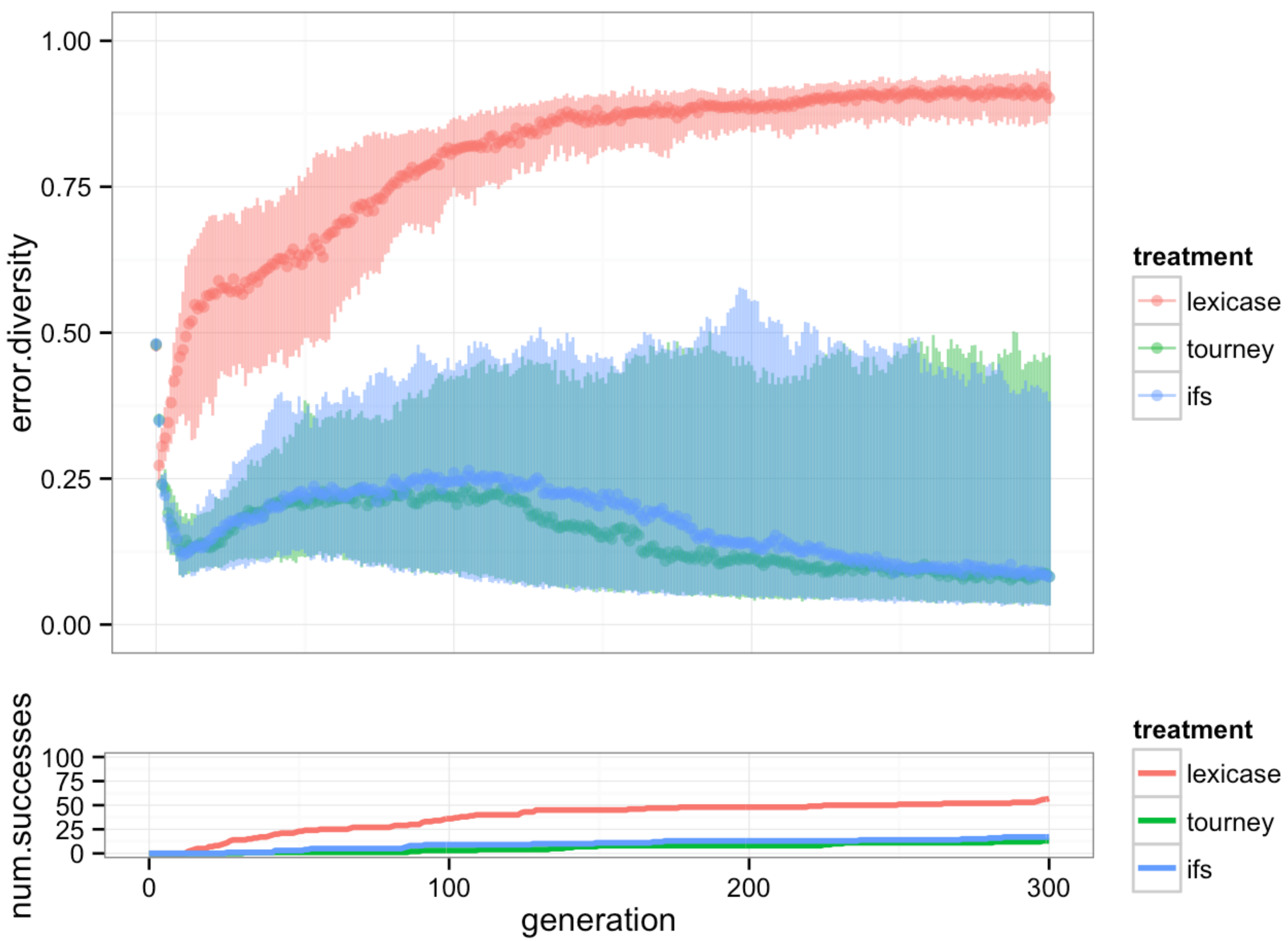
Table 3: Number of successful runs out of 100 for each setting, where "Tourn" is size 7 tournament selection, "IFS" is implicit fitness sharing with size 7 tournaments, and "Lex" is lexicase selection. For each problem, underline indicates significant improvement over the other two selection methods 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, ...

To the Future

- Expressive program representations (Push)
- Interactions among development, form, physics, behavior, and ecology (in virtual worlds)
- Evolution of reproduction and variation (autoconstructive evolution)

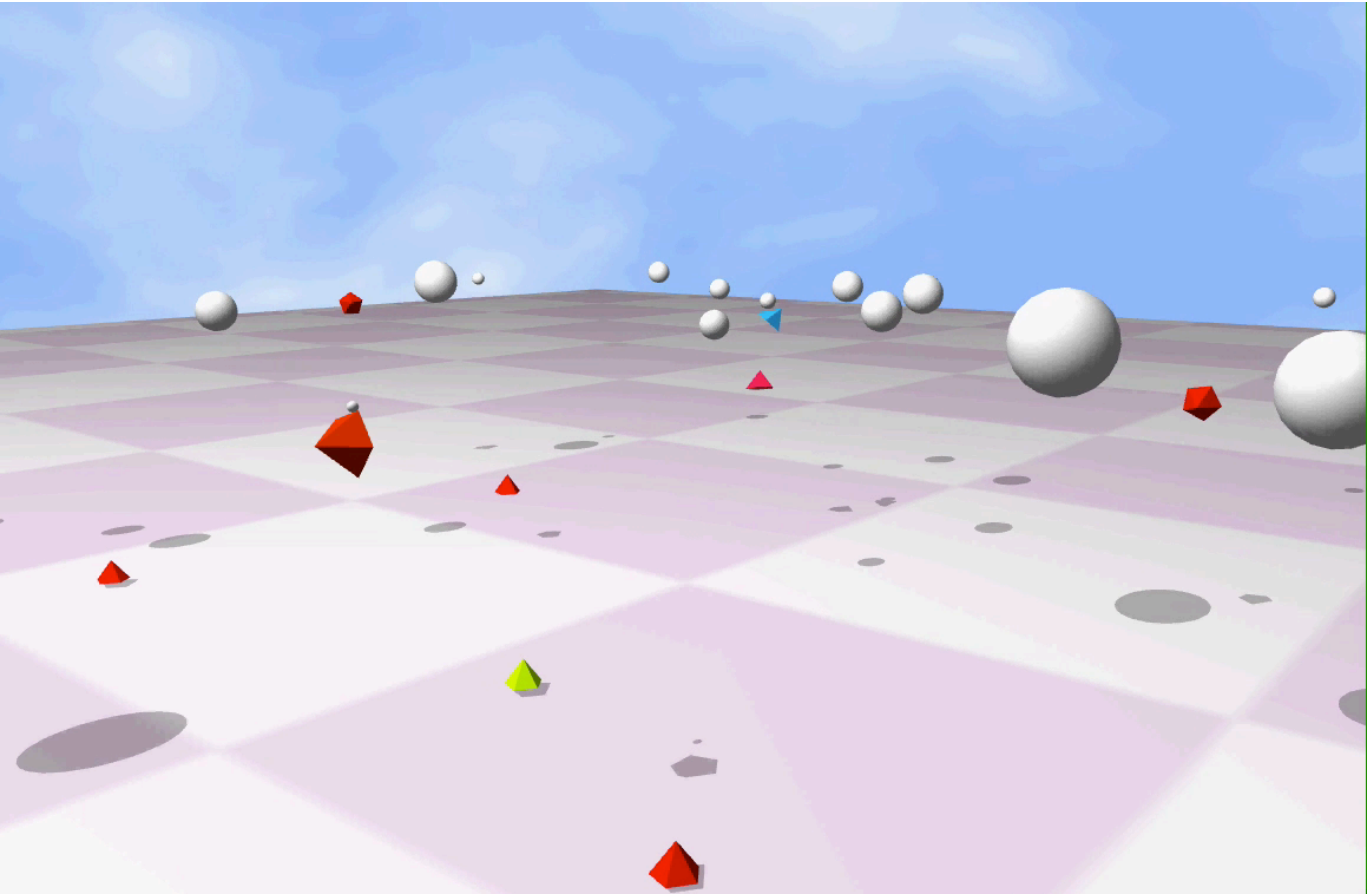
$\Rightarrow$ Evolution of adaptive complexity

Autoconstructive Evolution

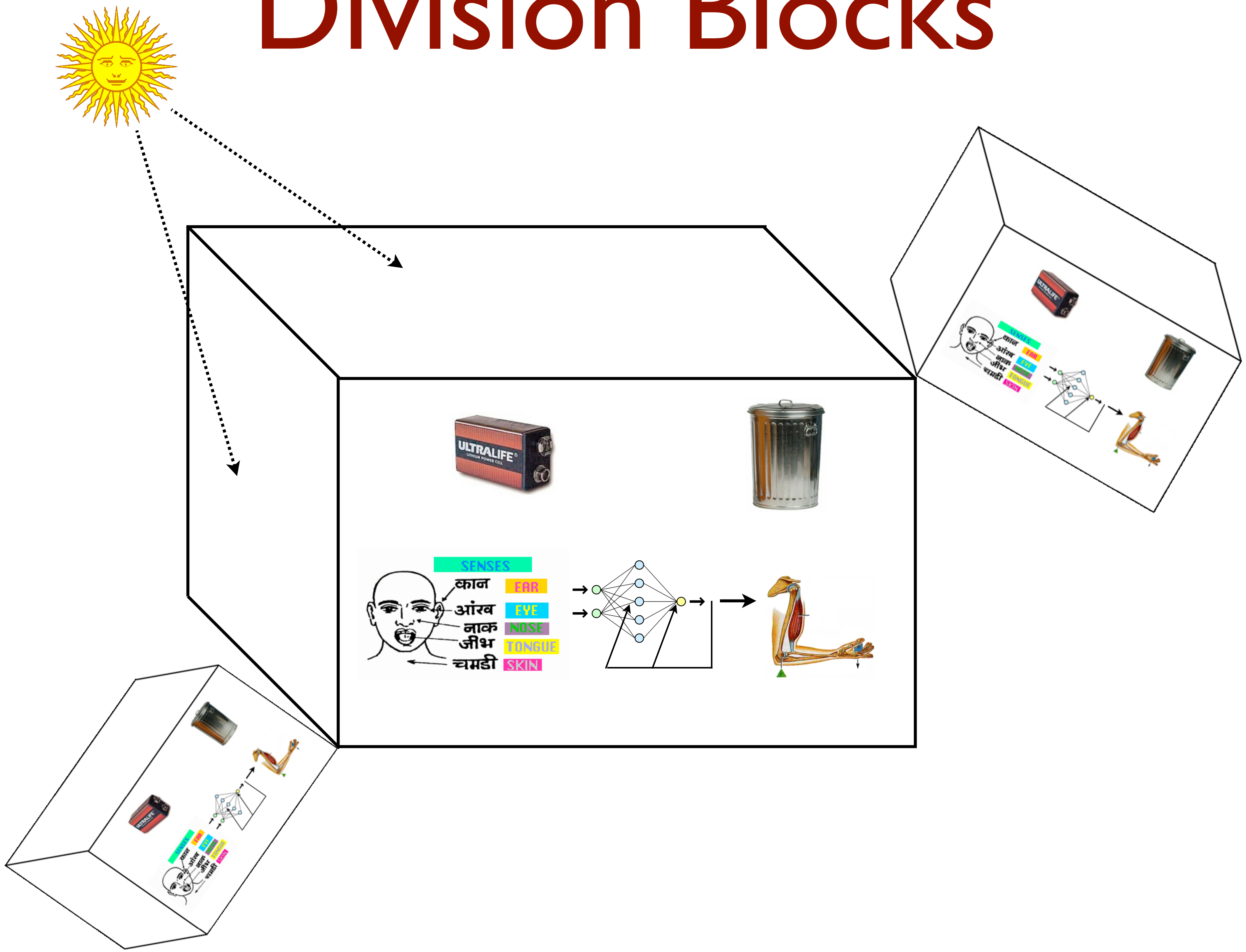
- Individual programs make their own children, with endogenous variation
- Hence they control their own mutation rates and methods, sexuality, reproductive timing, etc.
- The machinery of reproduction and diversification (i.e., the machinery of evolution) evolves
- Requires expressive program representations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

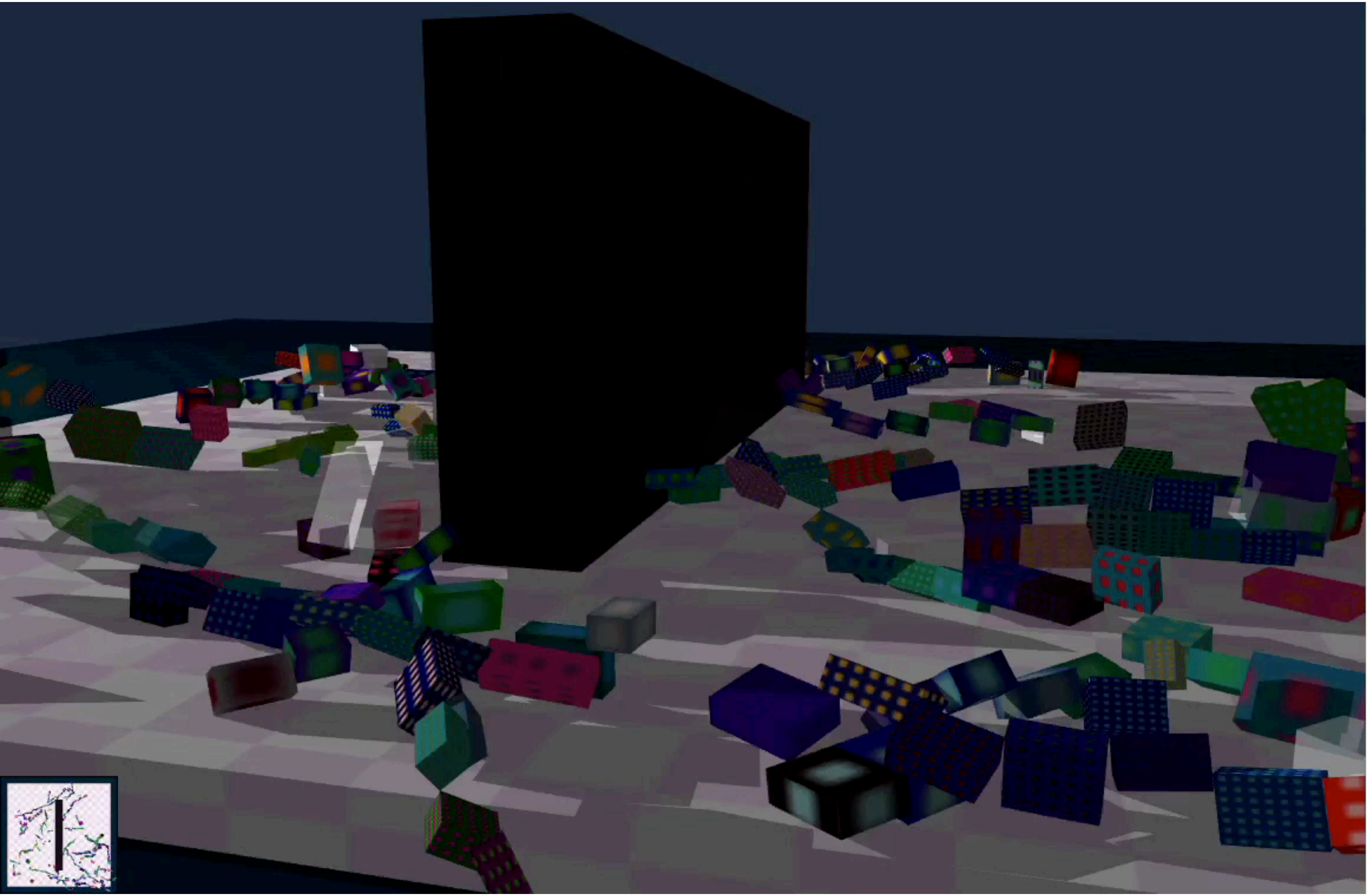
SwarmEvolve 2

- 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

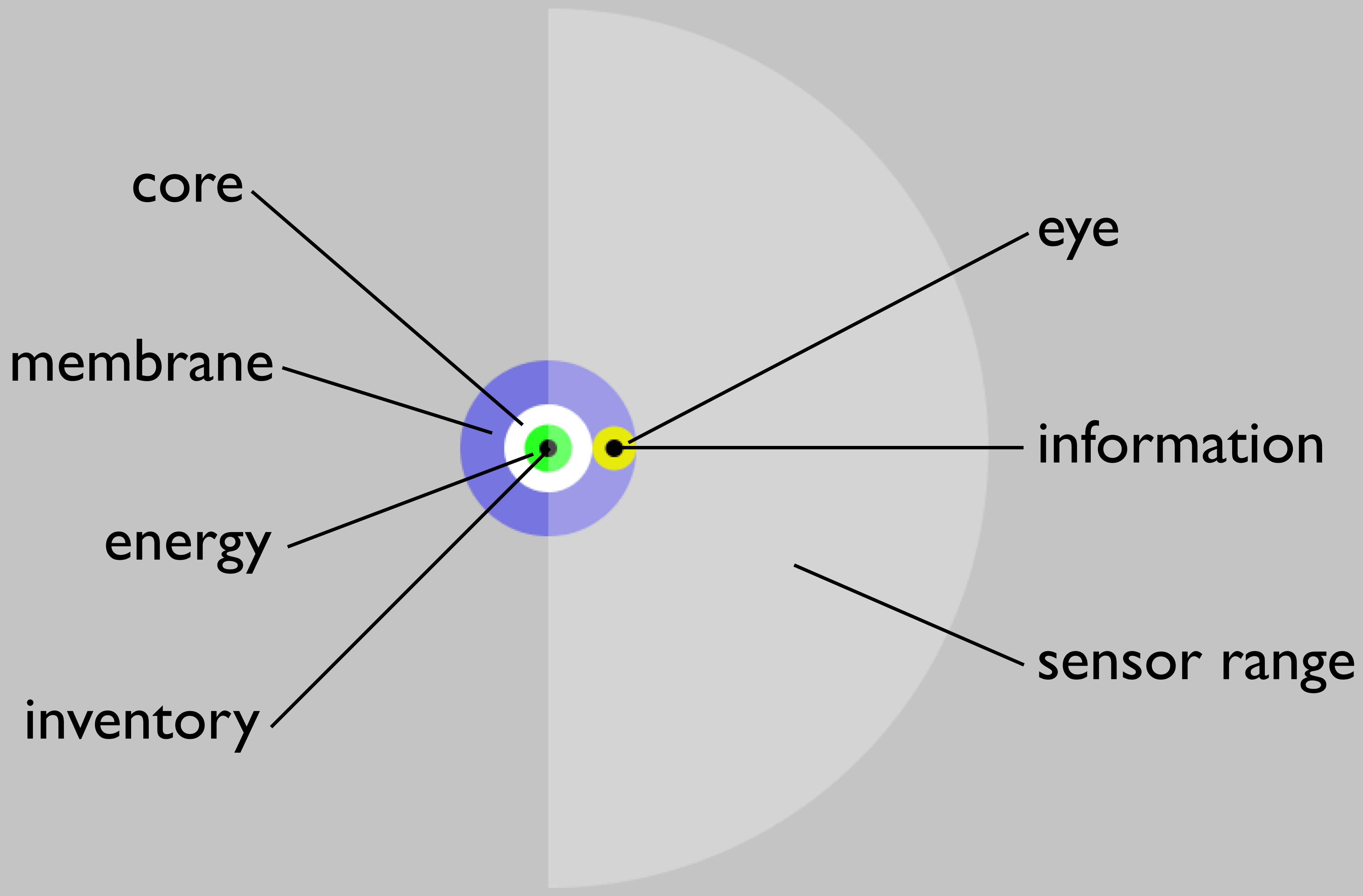


Division Blocks





Pucks


core
membrane
energy
inventory
eye
information
sensor range

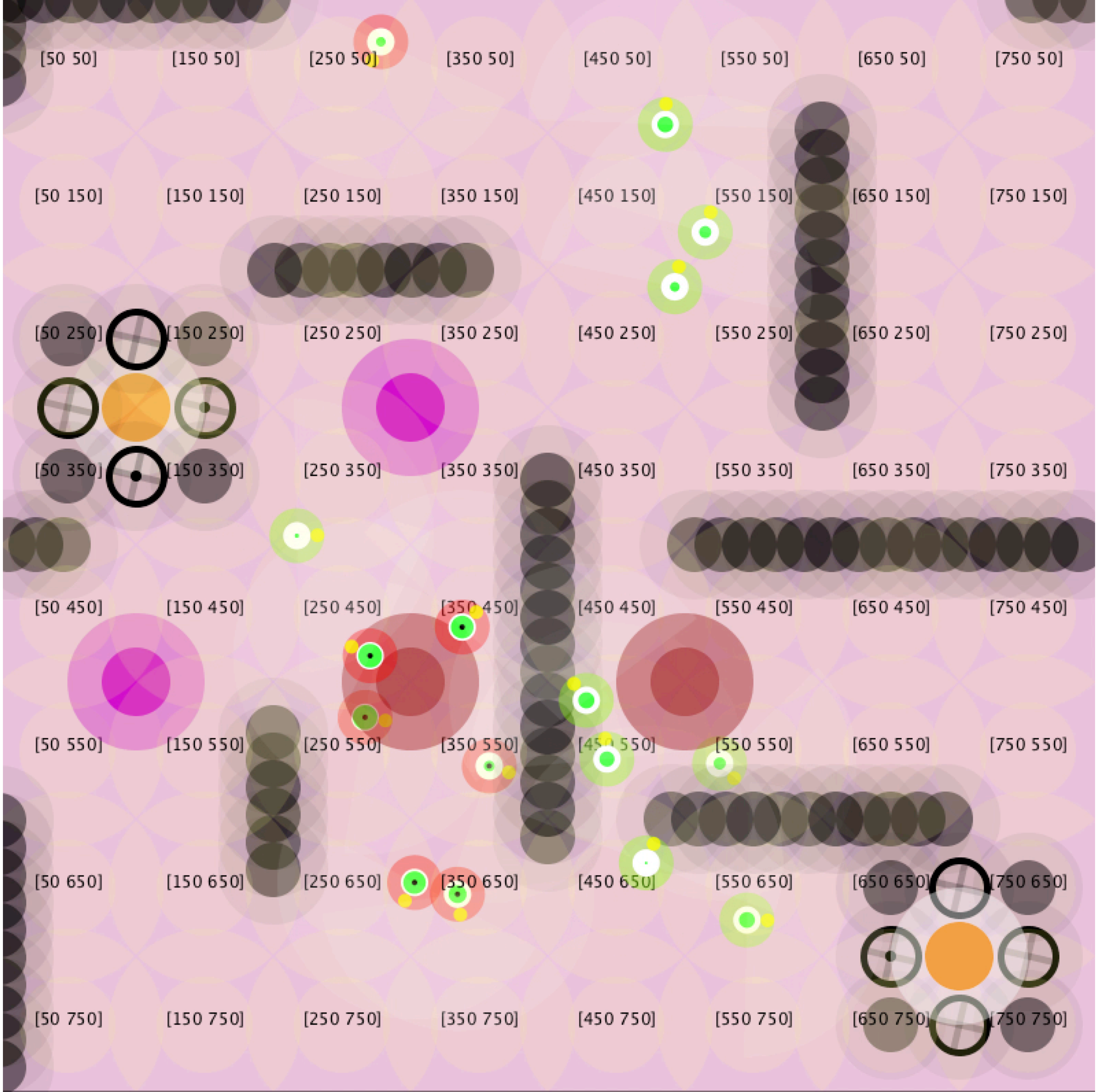

AREA-B
AREA-A

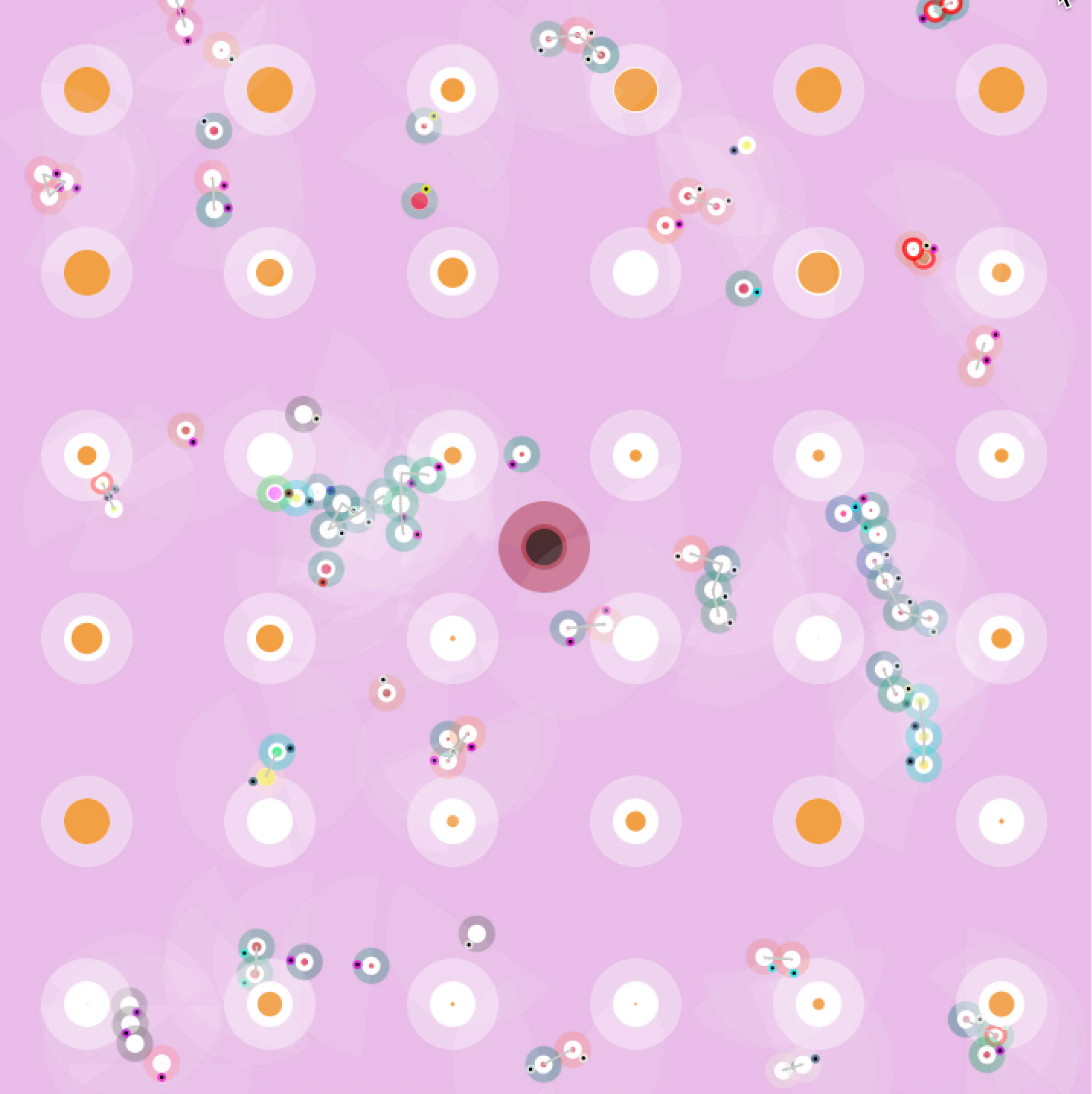
Action

- Pucks act by making proposals to the universe
- The universe accepts proposals permitted by physics and compatible with the proposals of other pucks
- When conflicts arise the universe arbitrates

Proposals

- Accelerate
- Rotate
- Remember
- Transact (via bid/ask):
  - Energy
  - Information
  - Inventory items
  - Binding
- Spawn new pucks


[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] [450 550] [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 advancing science and technology
  - 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