Biologically-Inspired Evolution of Computer Programs:

Tag-based Modularity in Genetic Programming

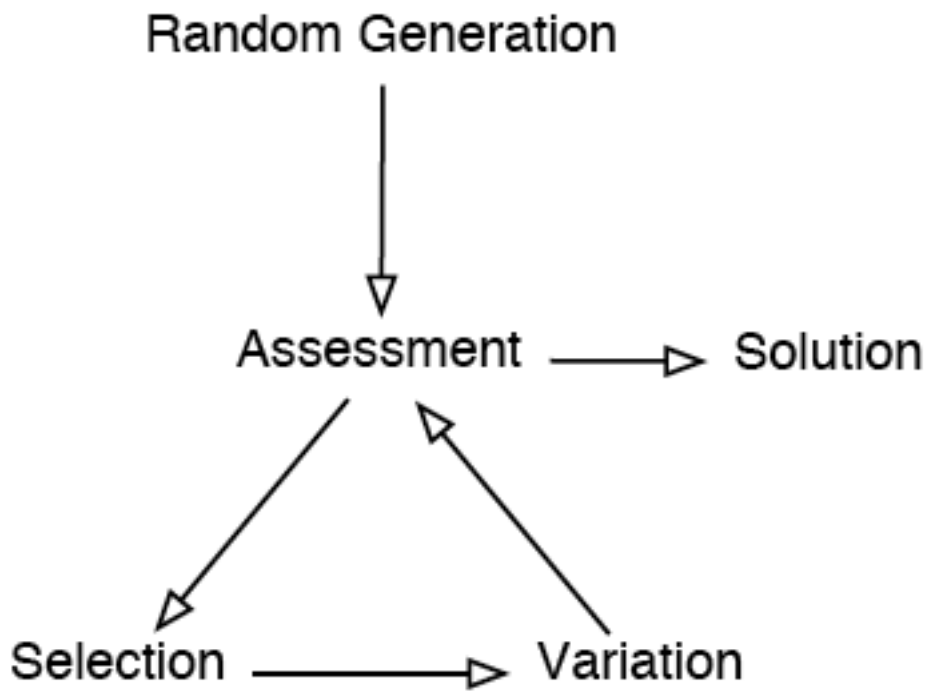
Lee Spector
Cognitive Science
Hampshire College

Presenting joint work with Brian Martin, Kyle Harrington, and Thomas Helmuth

Outline

- Genetic programming (GP)
- Modularity in GP
- Tags
- Tag-based modularity in GP

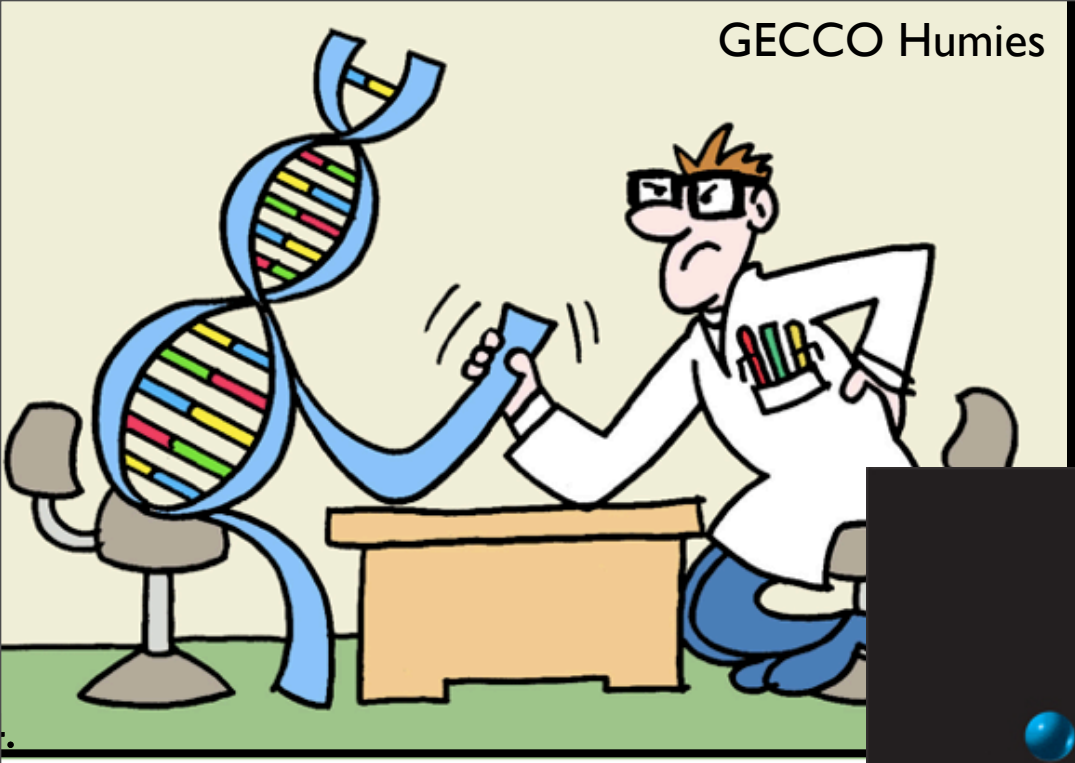
Evolutionary Computation



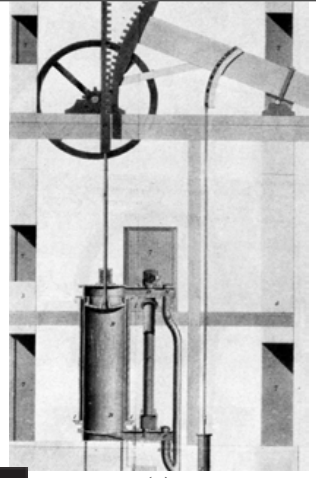
Genetic Programming

- Evolutionary computing to produce executable computer programs.
- Programs are tested by executing them.

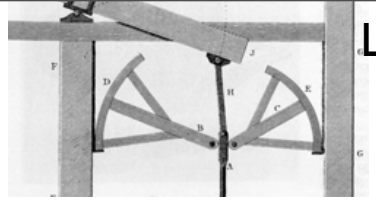
GECCO Humies



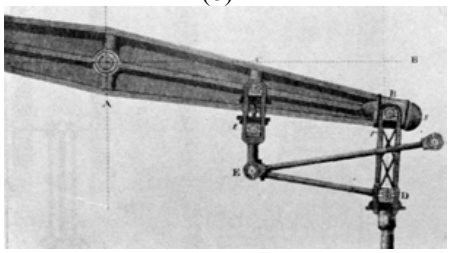
Lipson



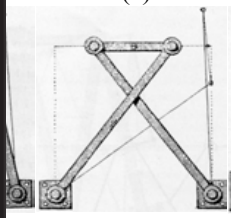
(a)



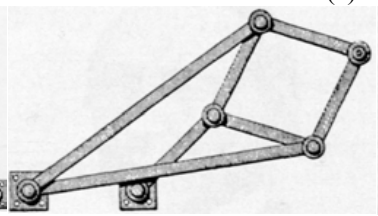
(b)



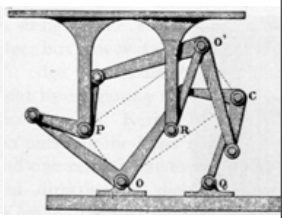
(c)



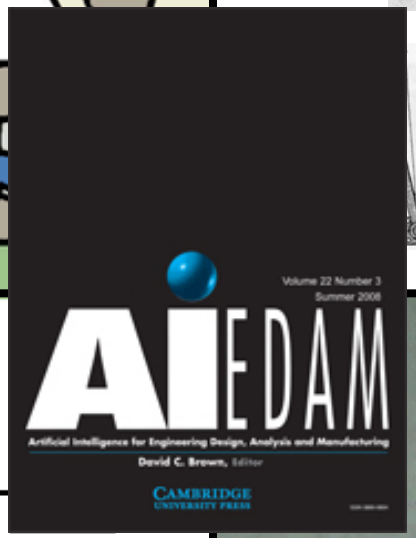
(e)

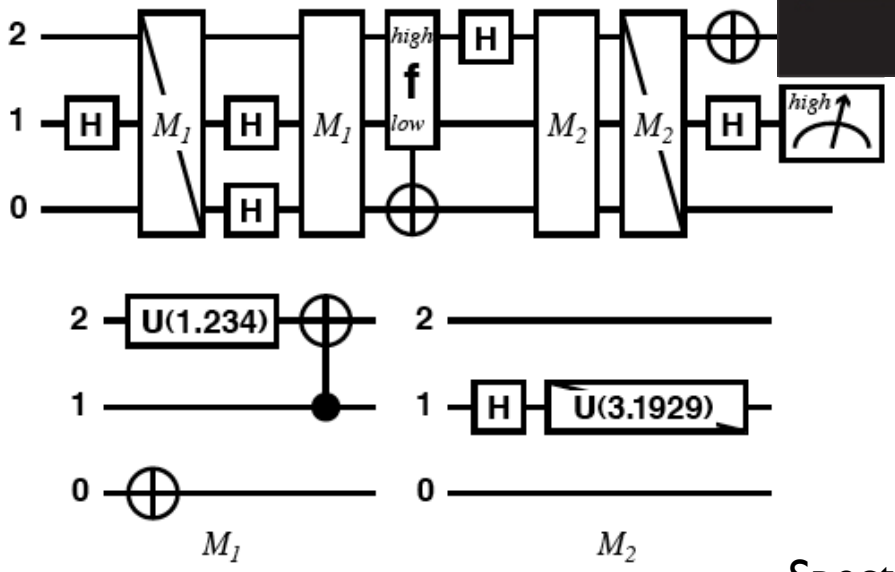


(f)

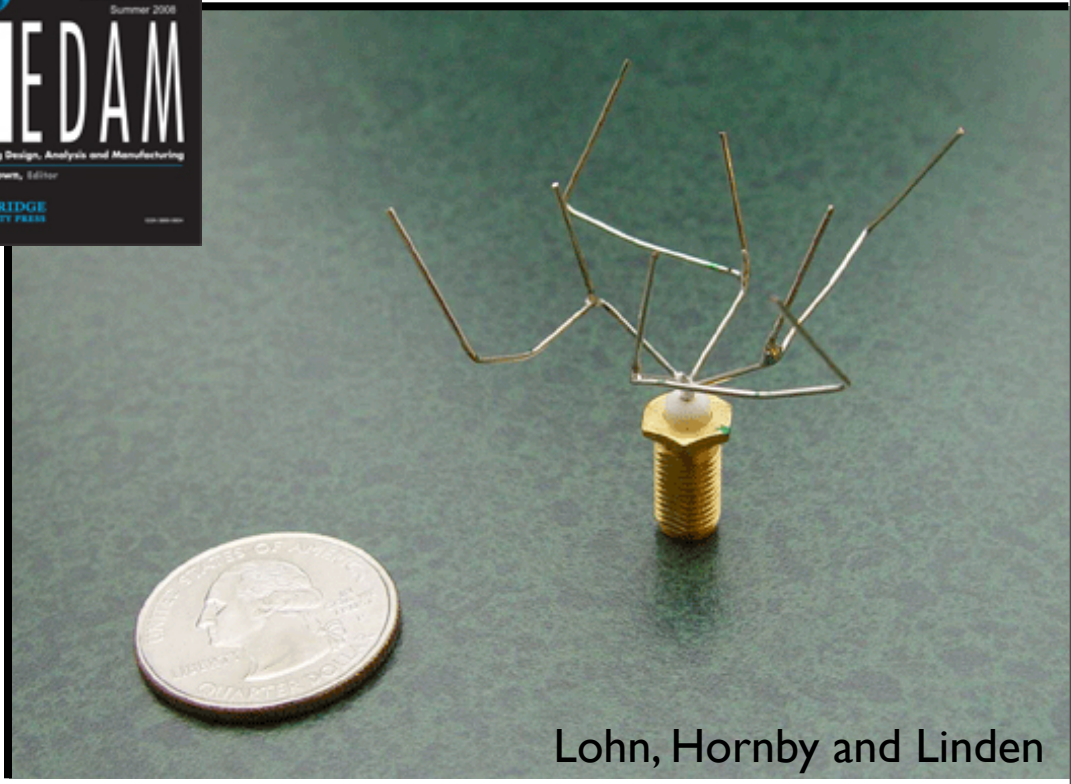


(g)





Spector



Lohn, Hornby and Linden

Evolution, the Designer

"Darwinian evolution is itself a designer worthy of significant respect, if not religious devotion." *Boston Globe* OpEd, Aug 29, 2005

WHAT WOULD DARWIN SAY? | LEE SPECTOR

And now, digital evolution

The Boston Globe

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.

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

Given a set of data points, evolve a program that produces *y* from *x*.

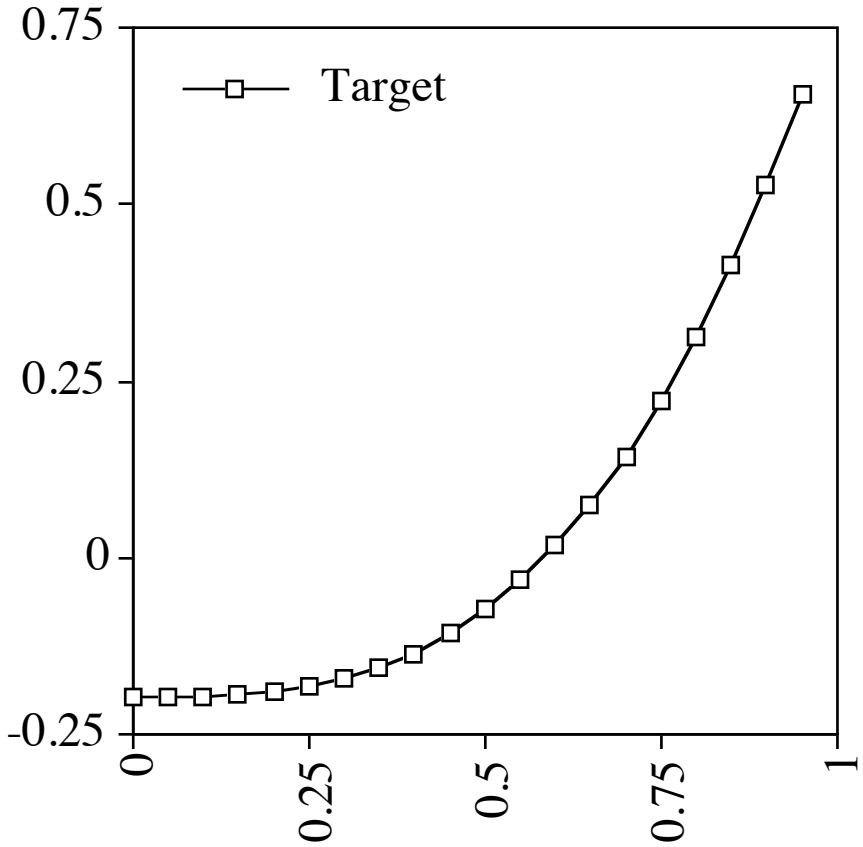
Primordial ooze: +, -, *, %, *x*, 0.1

Fitness = error (smaller is better)

GP Parameters

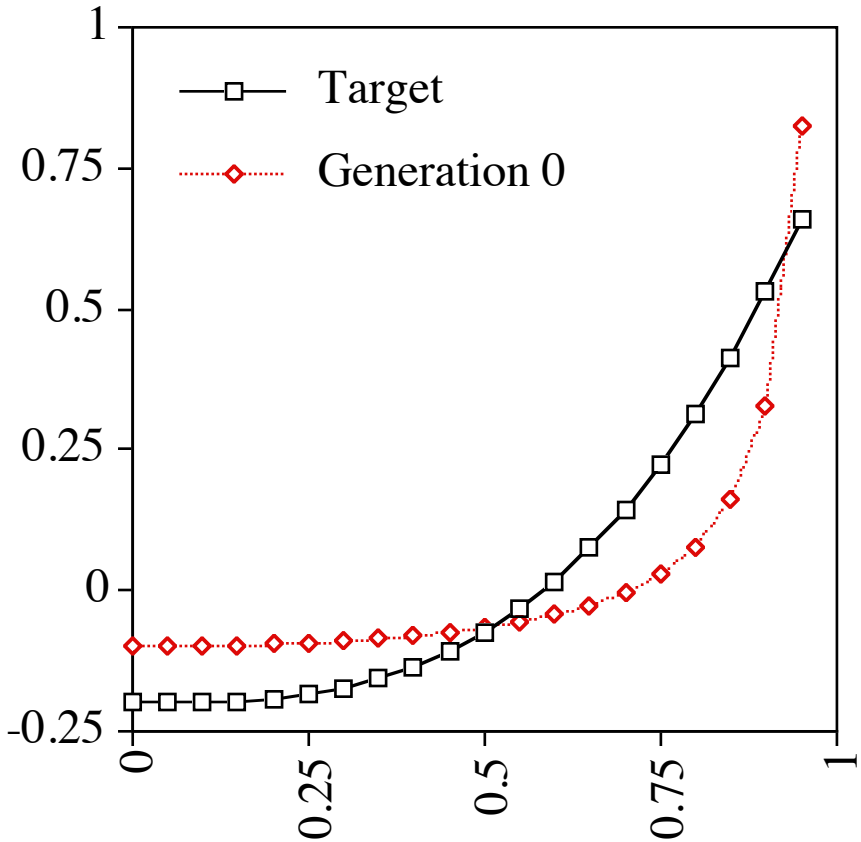
Maximum number of Generations: 51
Size of Population: 1000
Maximum depth of new individuals: 6
Maximum depth of new subtrees for mutants: 4
Maximum depth of individuals after crossover: 17
Fitness-proportionate reproduction fraction: 0.1
Crossover at any point fraction: 0.3
Crossover at function points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

Evolving $y = x^3 - 0.2$



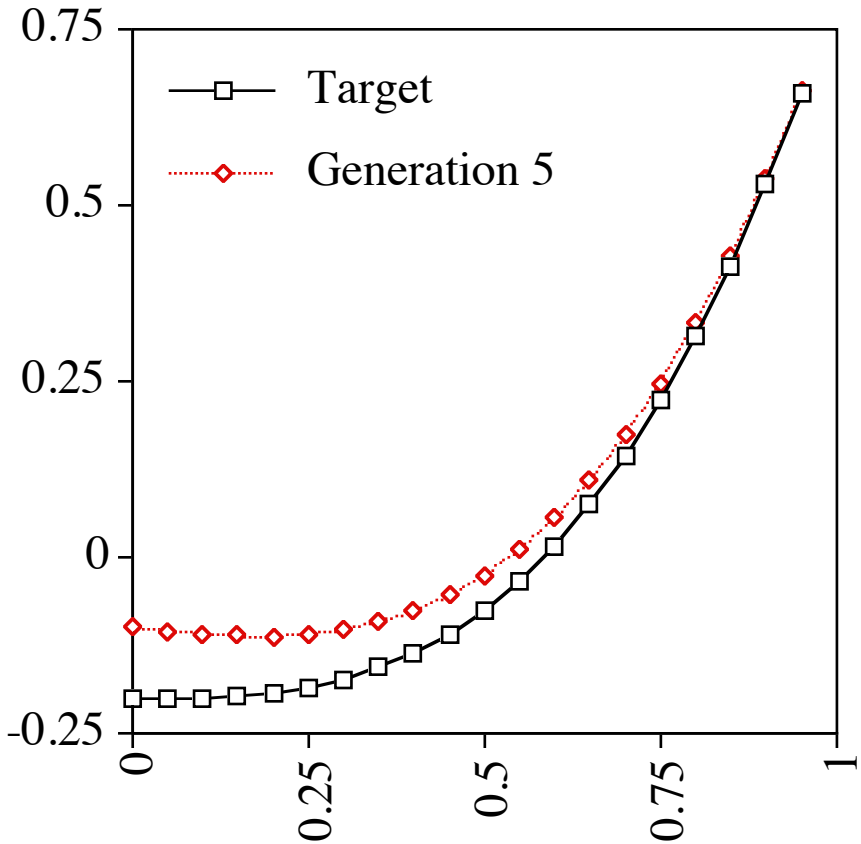
Best Program, Gen 0

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



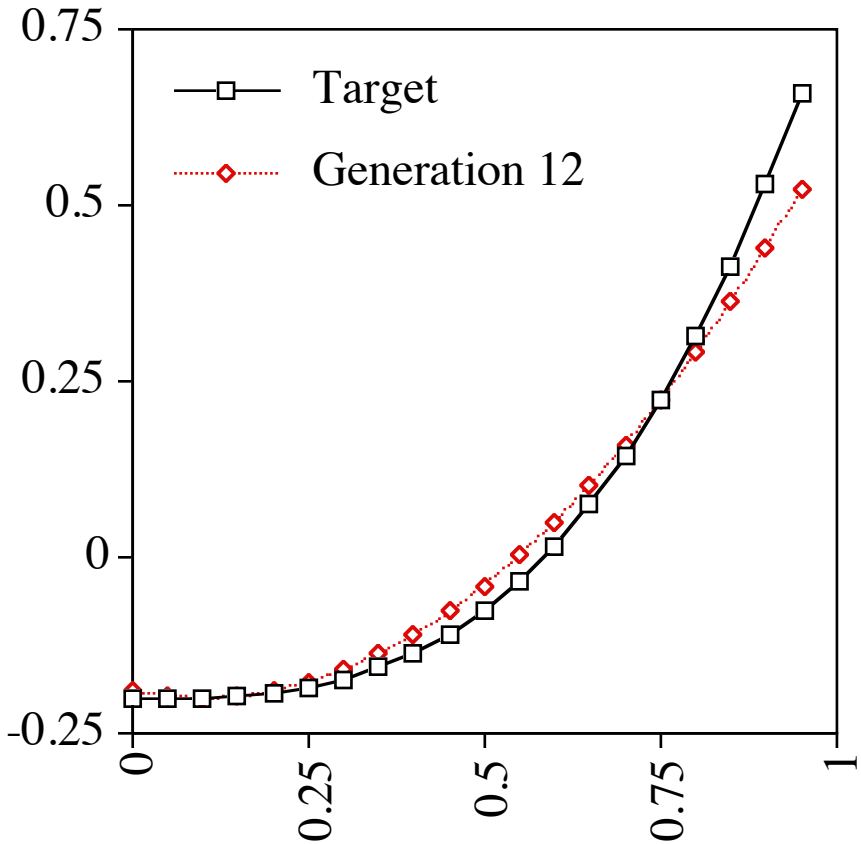
Best Program, Gen 5

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



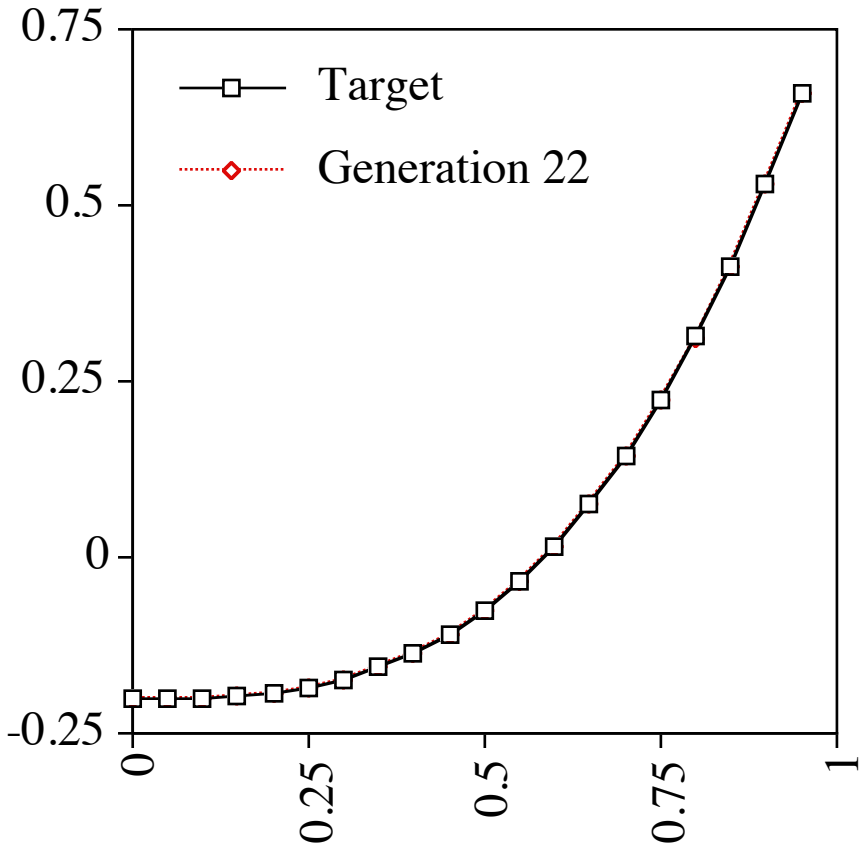
Best Program, Gen 12

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



Push

- A programming language designed for programs that evolve.
- Simplifies evolution of programs that may use:
  - multiple data types
  - subroutines (any architecture)
  - recursion and iteration
  - evolved control structures
  - evolved evolutionary mechanisms
- Primary feature that supports these capabilities is ease of program self-modification.

Push

- Stack-based postfix language with one stack per type
- Types include: integer, float, Boolean, name, code, exec, vector, matrix, quantum gate, [add more as needed]
- Missing argument? NOOP
- Trivial syntax:
  program → instruction | literal | (program*)

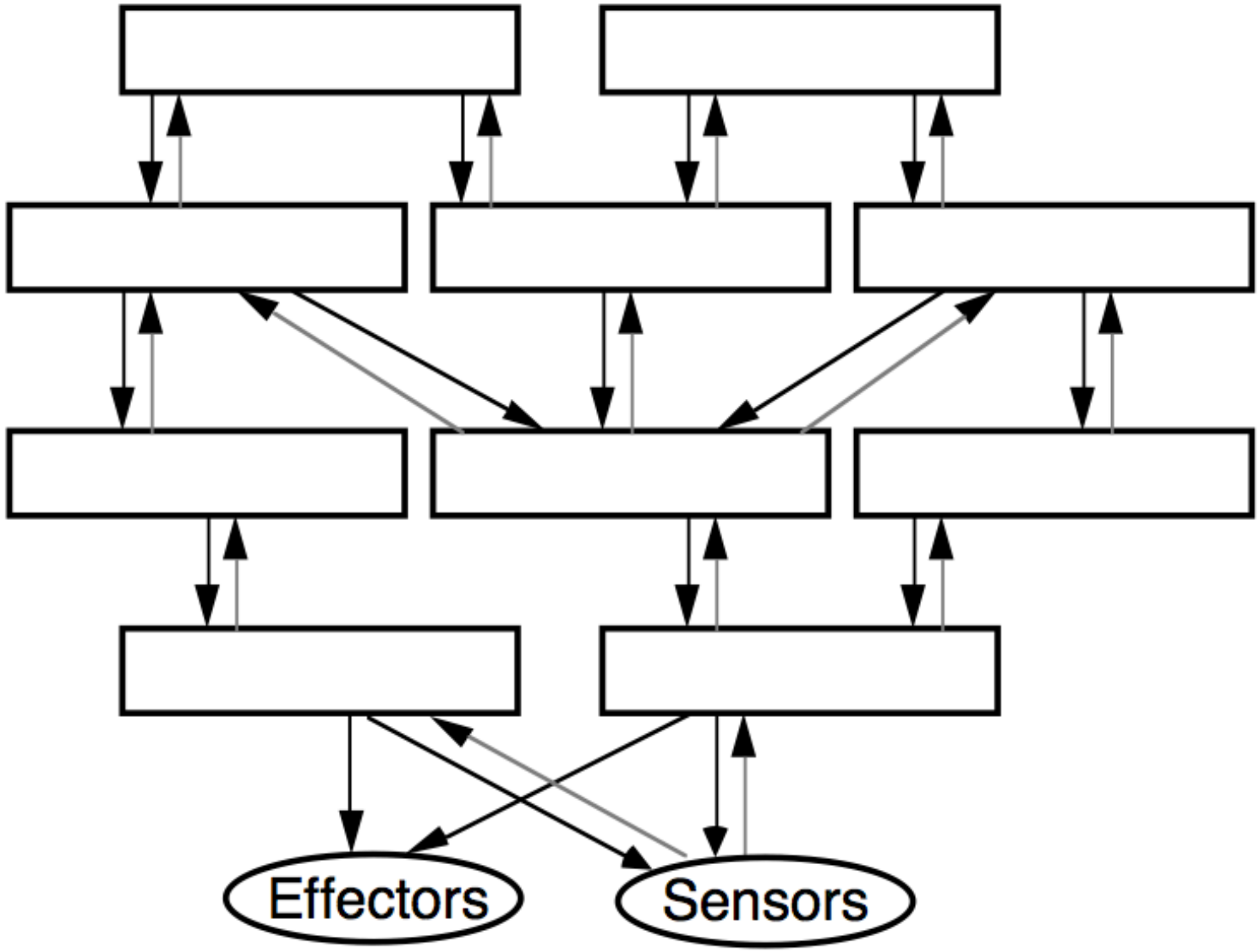
Sample Push Instructions

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

Autoconstructive Evolution

- Individuals make their own children.
- Agents thereby control their own mutation rates, sexuality, and reproductive timing.
- The machinery of reproduction and diversification (i.e., the machinery of evolution) evolves.
- Radical self-adaptation.

Modularity is Everywhere



Modules in GP

- Automatically-defined functions (Koza).
- Automatically-defined macros (Spector).
- Architecture-altering operations (Koza).
- Module acquisition/encapsulation systems (Kinnear, Roberts, many others).
- Push approach: instructions that can build/execute modules with no changes to the system's representations or algorithms.

ADFs

- All programs in the population have the same, pre-specified architecture
- Genetic operators respect that architecture
-
```
(progn (defn adf0 (arg0 arg1) ...)
       (defn adf1 (arg0 arg1 arg2) ...)
  (.... (adf1 ...) (adf0 ...) ...))
```

Modules in Push

- Execution stack manipulation:

  `(3 exec.dup (1 integer.+))`

  Can be more complex, and has produced nice results, but tricky in complex contexts

- Named modules:

  `(plus1 exec.define (1 integer.+)) ... plus1`

  More general but coordinating definitions and references to arbitrarily many names is also tricky ***and this never arises in evolution!***

- How can we do better?

Evolution of Altruism

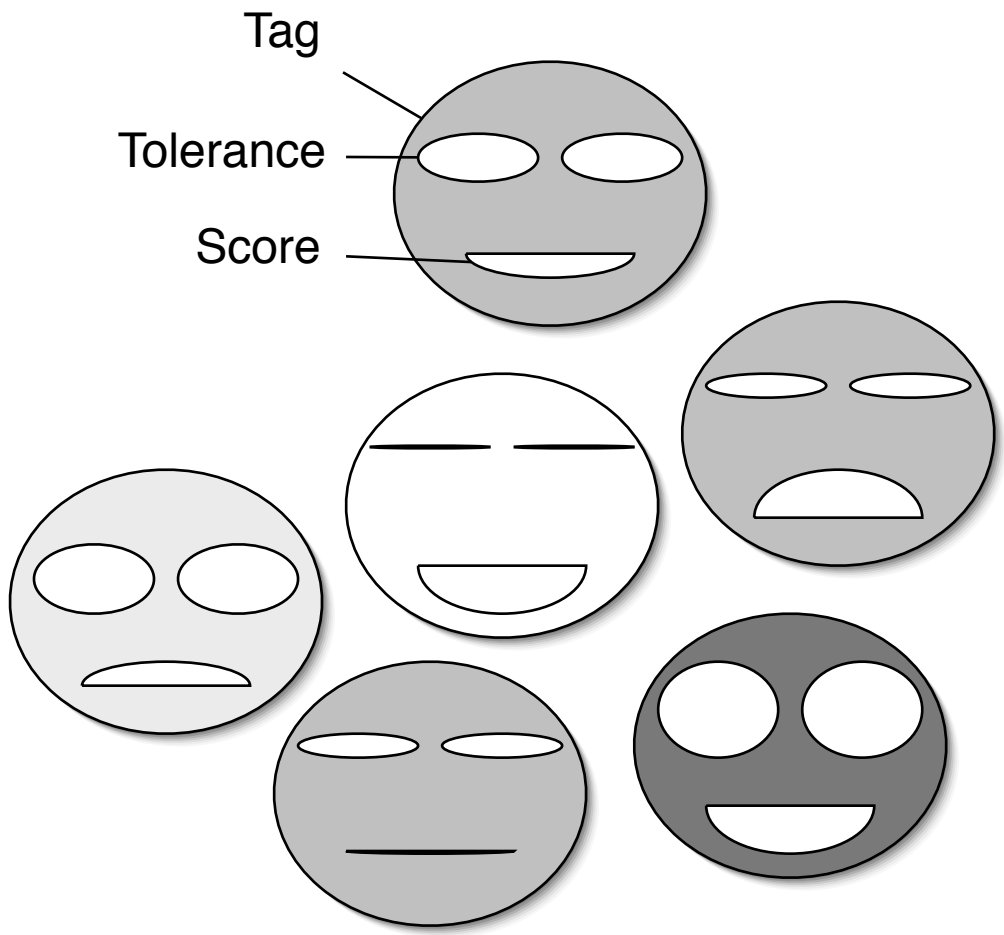
- Puzzles/challenges/results since Darwin
- Explanations of altruism toward:
  - Kin
  - Reciprocating partners
  - Agents with good reputations

Tag-Mediated Altruism

- Tags = arbitrary identifiers (Holland, 1995)
- Riolo *et al.* (*Nature*, 2001) showed that altruism based only on tag similarity can evolve in simple simulations.
- Roberts & Sherratt (*Nature*, 2002) claimed that Riolo *et al.*'s result held only when agents with identical tags were *required* to donate to one another.

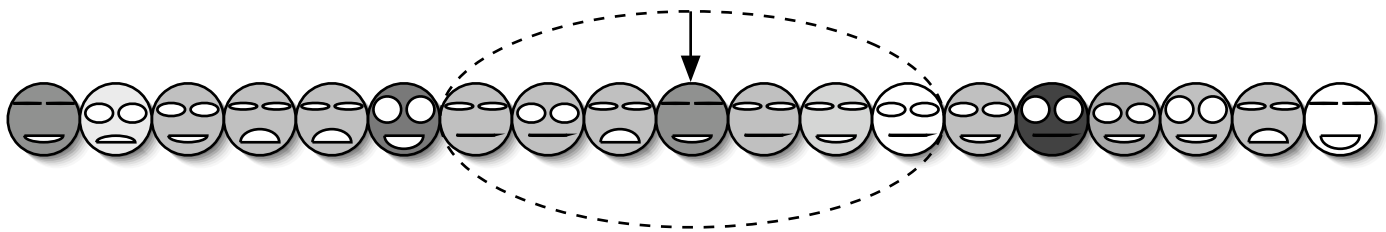
Tags and Tolerances

1. Donations when Δtags $\leq$ tolerance
2. Reproductive Tournaments
3. Mutation

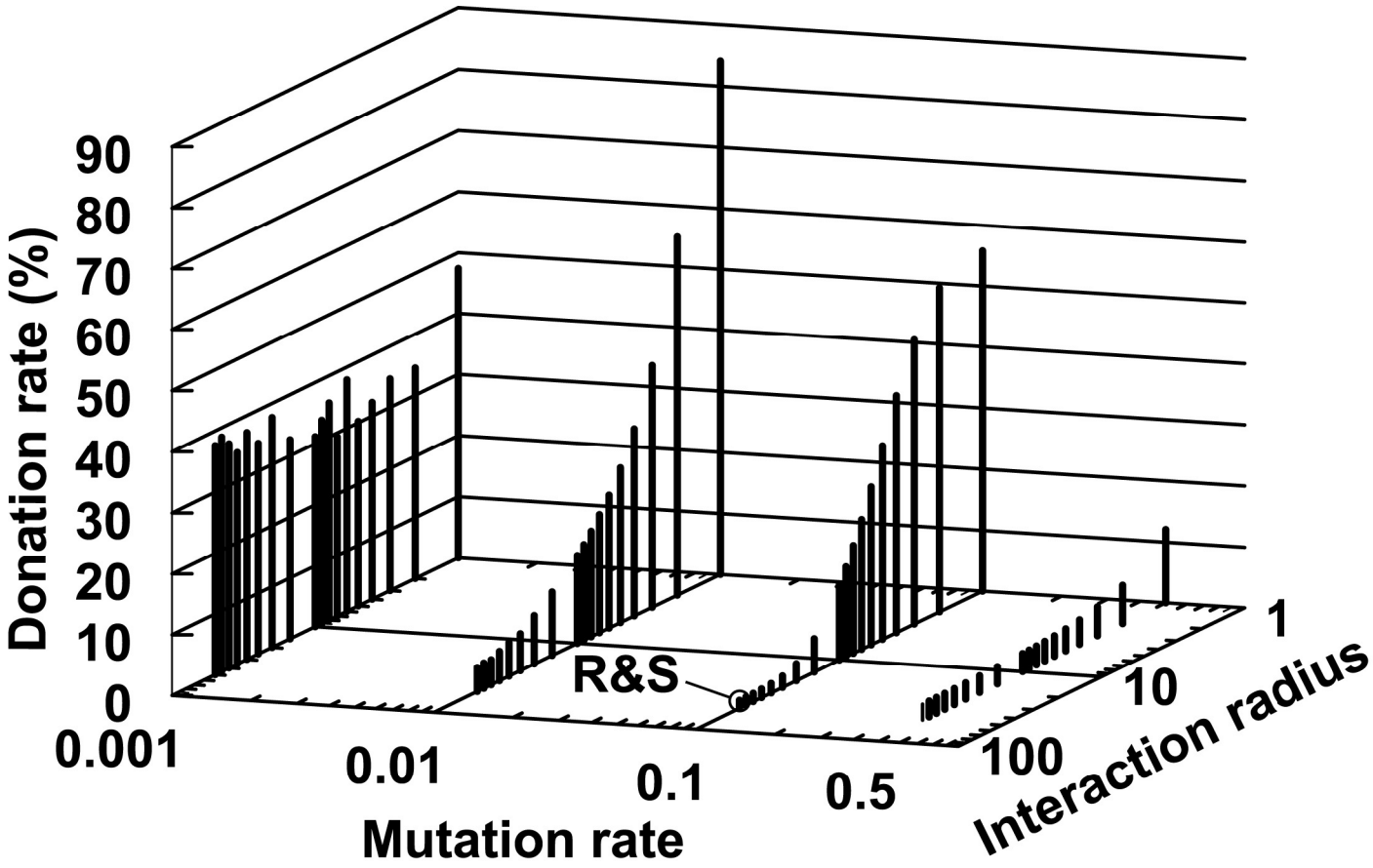


Genetic Stability and Territorial Structure

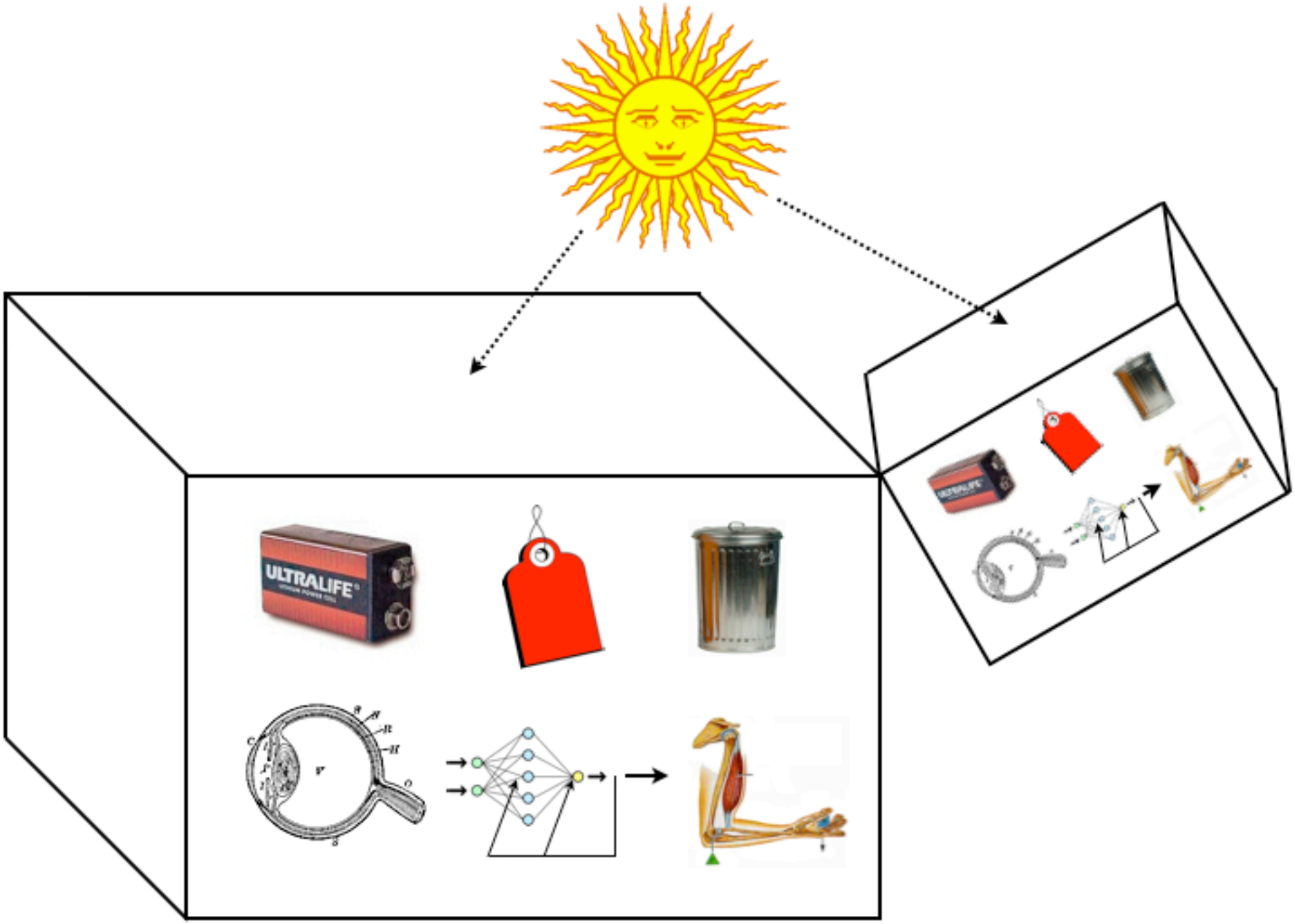
- Varied mutation rate.
- Varied "interaction radius" within a linearly structured population.

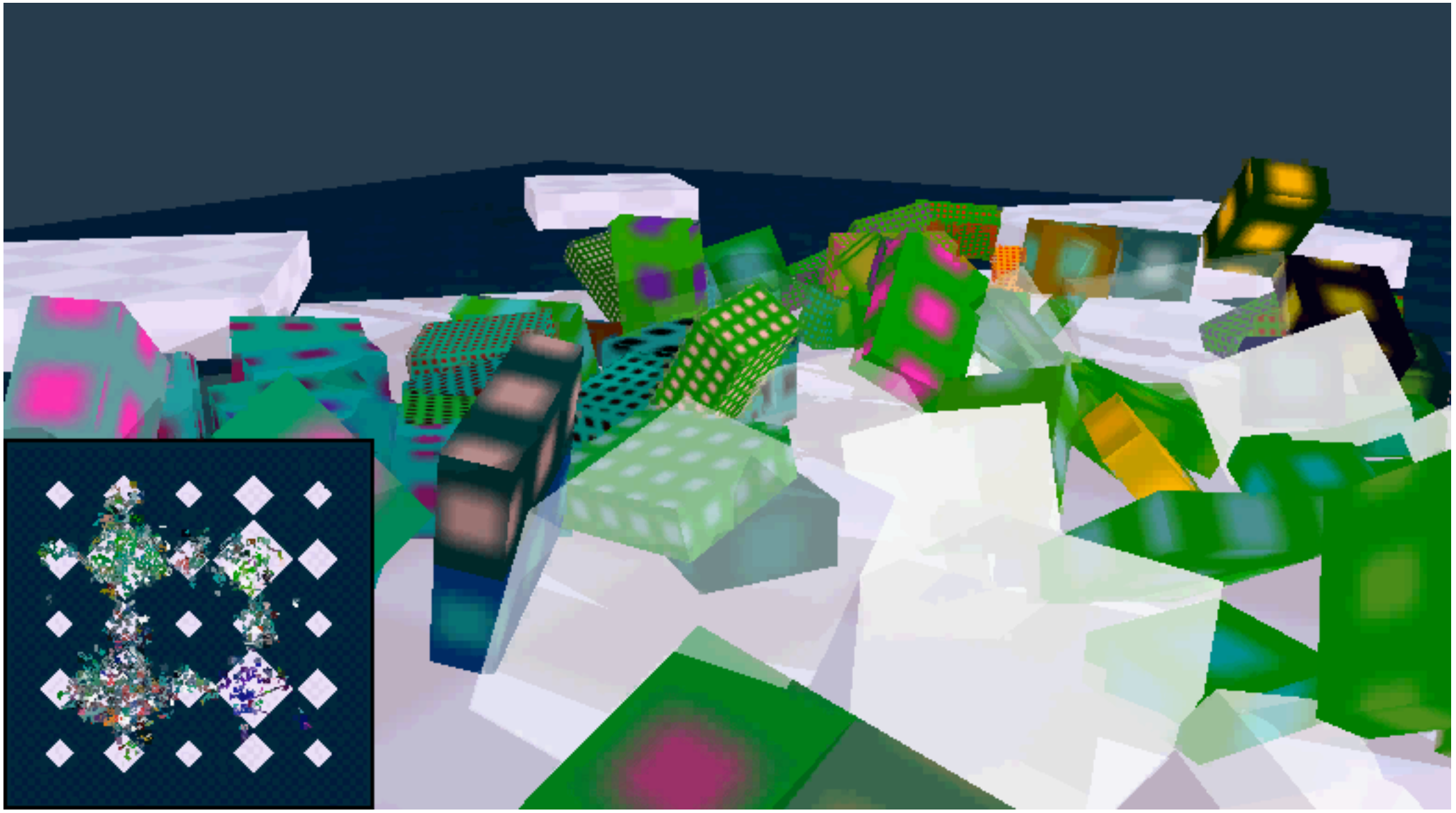


Genetic Stability and Territorial Structure



Spector, L., and Klein, J. Genetic stability and territorial structure facilitate the evolution of tag-mediated altruism. In *Artificial Life*.


ULTRALIFE

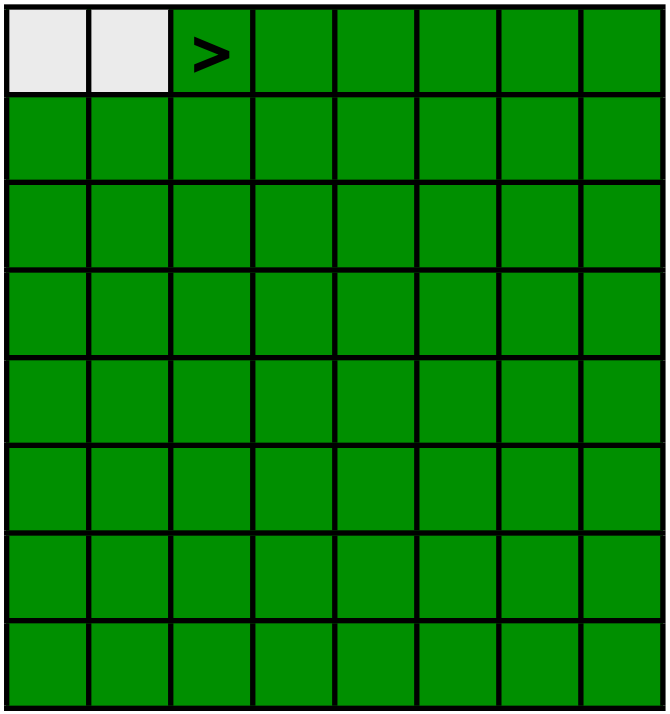


Tags in Push

- Tags are integers embedded in instruction names
- Instructions like tag.exec.123 tag values
- Instructions like tagged.456 recall values by *closest matching* tag
- If a single value has been tagged then all tag references will recall values
- The number of tagged values can grow incrementally over evolutionary time

Lawnmower Problem

- Used by Koza to demonstrate utility of ADFs for scaling GP up to larger problems



Lawnmower Instructions

Condition	Instructions
Basic	left, mow, v8a, frog, $\mathcal{R}_{v8}$
Tag	left, mow, v8a, frog, $\mathcal{R}_{v8}$, tag.exec.[1000], tagged.[1000]
Exec	left, mow, v8a, frog, $\mathcal{R}_{v8}$, exec.dup, exec.pop, exec.rot, exec.swap, exec.k, exec.s, exec.y

Rocks-Cluster Physical View for Wed, 02 Mar 2011 07:28:18 -0500

Get Fresh Data

Full View

Grid > Rocks-Cluster > --Choose a Node

Rocks-Cluster cluster - Physical View | Columns 5

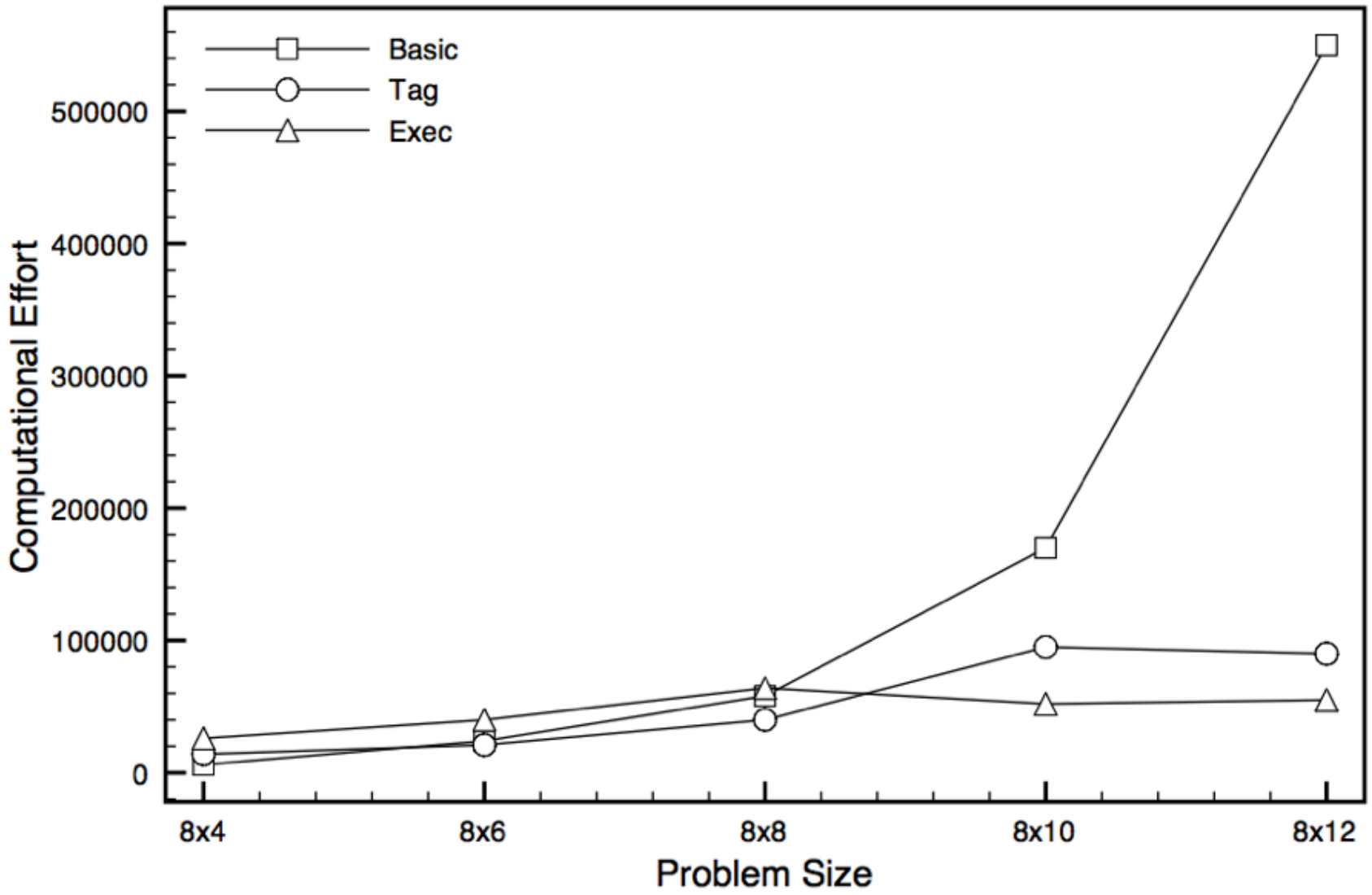
Verbosity level (Lower is more compact): 3 ○ 2 ◉ 1 ○

Total CPUs: **158**
Total Memory: **229.5 GB**

Total Disk: **3798.2 GB**
Most Full Disk: **compute-1-11.local (77.2% Used)**

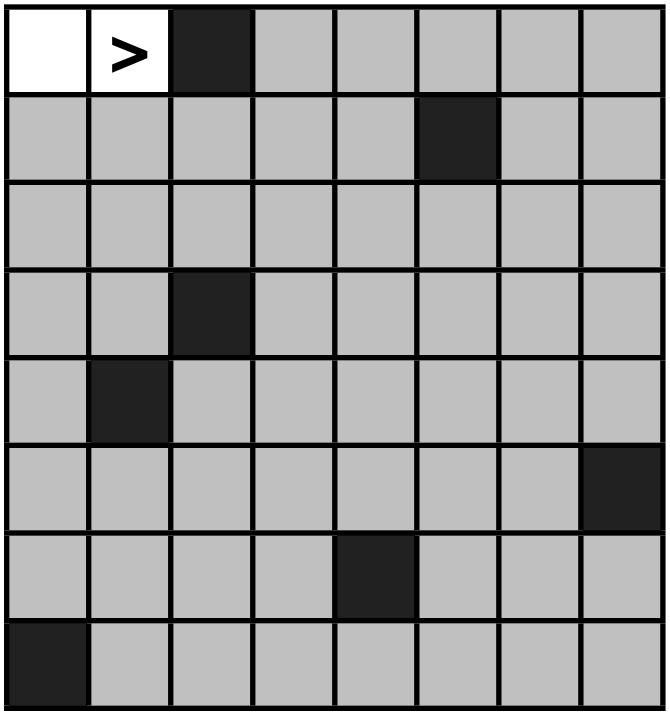
Rack 0	Rack 1	Rack 2	Rack 3	Rack 4
fly.local 0.11 cpu: 1.56G (8) mem: 15.68G	compute-1-16.local 4.00 cpu: 2.93G (4) mem: 7.77G	compute-2-10.local 2.00 cpu: 2.93G (2) mem: 3.86G	compute-3-3.local 2.00 cpu: 1.95G (2) mem: 3.87G	compute-4-2.local 49.12 cpu: 1.86G (48) mem: 31.42G
	compute-1-15.local 3.97 cpu: 2.93G (4) mem: 7.77G	compute-2-9.local 2.00 cpu: 2.93G (2) mem: 3.86G		compute-4-1.local 16.04 cpu: 2.21G (16) mem: 23.53G
	compute-1-14.local 3.99 cpu: 2.93G (4) mem: 7.77G	compute-2-8.local 2.00 cpu: 2.93G (2) mem: 3.86G		
		compute- 2.12		

Lawnmower Effort



Dirt-Sensing, Obstacle-Avoiding Robot Problem

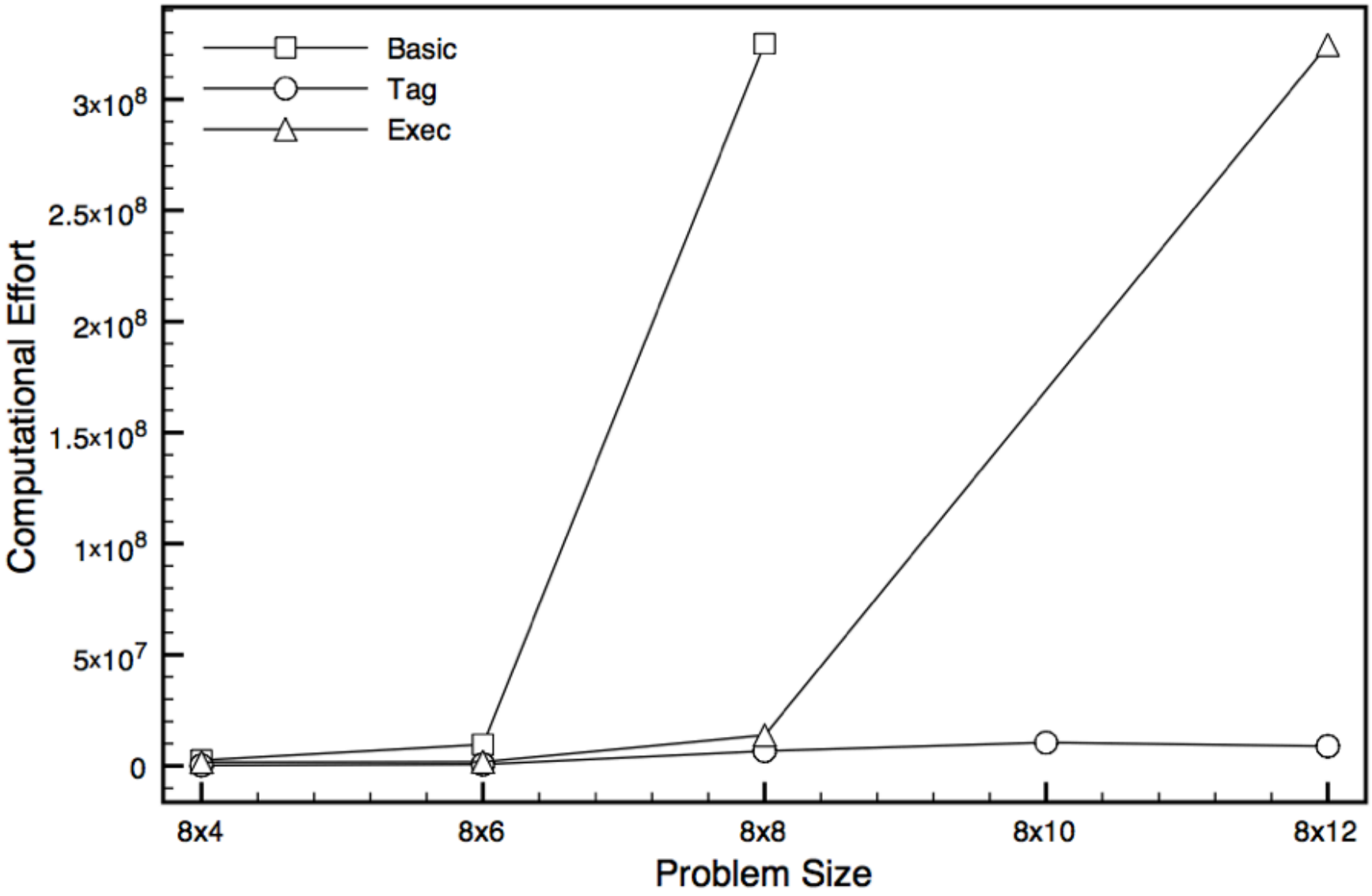
Like the lawnmower problem but harder and less uniform



DSOAR Instructions

Condition	Instructions
Basic	if-dirty, if-obstacle, left, mop, v8a, frog, $\mathcal{R}_{v8}$
Tag	if-dirty, if-obstacle, left, mop, v8a, frog, $\mathcal{R}_{v8}$, tag.exec.[1000], tagged.[1000]
Exec	if-dirty, if-obstacle, left, mop, v8a, frog, $\mathcal{R}_{v8}$, exec.dup, exec.pop, exec.rot, exec.swap, exec.k, exec.s, exec.y

DSOAR Effort



Conclusions

- Execution stack manipulation supports the evolution of modular programs in many situations
- Tag-based modules are more effective in complex, non-uniform problem environments
- Tag-based modules may help to evolve complex software and solutions to unsolved problems in the future

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 #1017817

RI: Small: RUI: Evolution of Robustly Intelligent Computational Systems

NSF Org:	IIS Division of Information & Intelligent Systems
Initial Amendment Date:	August 19, 2010
Latest Amendment Date:	August 19, 2010
Award Number:	1017817
Award Instrument:	Standard Grant
Program Manager:	Sven G. Koenig IIS Division of Information & Intelligent Systems CSE Directorate for Computer & Information Science & Engineering
Start Date:	September 1, 2010
Expires:	August 31, 2013 (Estimated)
Awarded Amount to Date:	$423288
Investigator(s):	Lee Spector lspector@hampshire.edu (Principal Investigator)
Sponsor:	Hampshire College