Genetic Programming for Finite Algebras

— *GECCO-2008* Presentation —

Lee Spector *
David M. Clark ✣
Ian Lindsay *
Bradford Barr *
Jon Klein *

* Cognitive Science, Hampshire College, Amherst, MA
✣ Mathematics, SUNY New Paltz, New Paltz, NY

Outline

- The domain
- Specific problems
- Methods
- Results
- Significance

Everybody's Favorite Finite Algebra

$$\text{Boolean algebra, } \mathbf{B} := \langle \{0,1\}, \wedge, \vee, \neg \rangle$$

$\wedge$	0	1
0	0	0
1	0	1

$\vee$	0	1
0	0	1
1	1	1

	$\neg$
0	1
1	0

Primal: every possible operation can be expressed by a term using only (and not even) $\wedge$, $\vee$, and $\neg$.

Bigger Finite Algebras

- Have applications in many areas of science, engineering, mathematics
- Can be *much* harder to analyze/understand
- Number of terms grows astronomically with size of underlying set
- Under active investigation for decades, with major advances (cited fully in the paper) in 1939, 1954, 1970, 1975, 1979, 1991, 2008

Goal

- Find terms that have certain special properties
- *Discriminator* terms, determine primality

$$t^A(x,y,z) = \begin{cases} x \text{ if } x \neq y \\ z \text{ if } x = y \end{cases}$$

- *Mal'cev*, *majority*, and *Pixley* terms
- For decades there was no way to produce these terms in general, short of exhaustive search
- Current best methods produce enormous terms

Specific Algebras

$\mathbf{A}_1$ *	0	1	2
0	2	1	2
1	1	0	0
2	0	0	1

$\mathbf{A}_2$ *	0	1	2
0	2	0	2
1	1	0	2
2	1	2	1

$\mathbf{A}_3$ *	0	1	2
0	1	0	1
1	1	2	0
2	0	0	0

$\mathbf{A}_4$ *	0	1	2
0	1	0	1
1	0	2	0
2	0	1	0

$\mathbf{A}_5$ *	0	1	2
0	1	0	2
1	1	2	0
2	0	1	0

$\mathbf{B}_1$ *	0	1	2	3
0	1	3	1	0
1	3	2	0	1
2	0	1	3	1
3	1	0	2	0

Methods

- Traditional genetic programming with ECJ
- Stack-based genetic programming with PushGP
- Alternative random code generators
- Asynchronous islands
- Trivial geography
- Parsimony-based selection
- Alpha-inverted selection pressure
- HAH = Historically Assessed Hardness

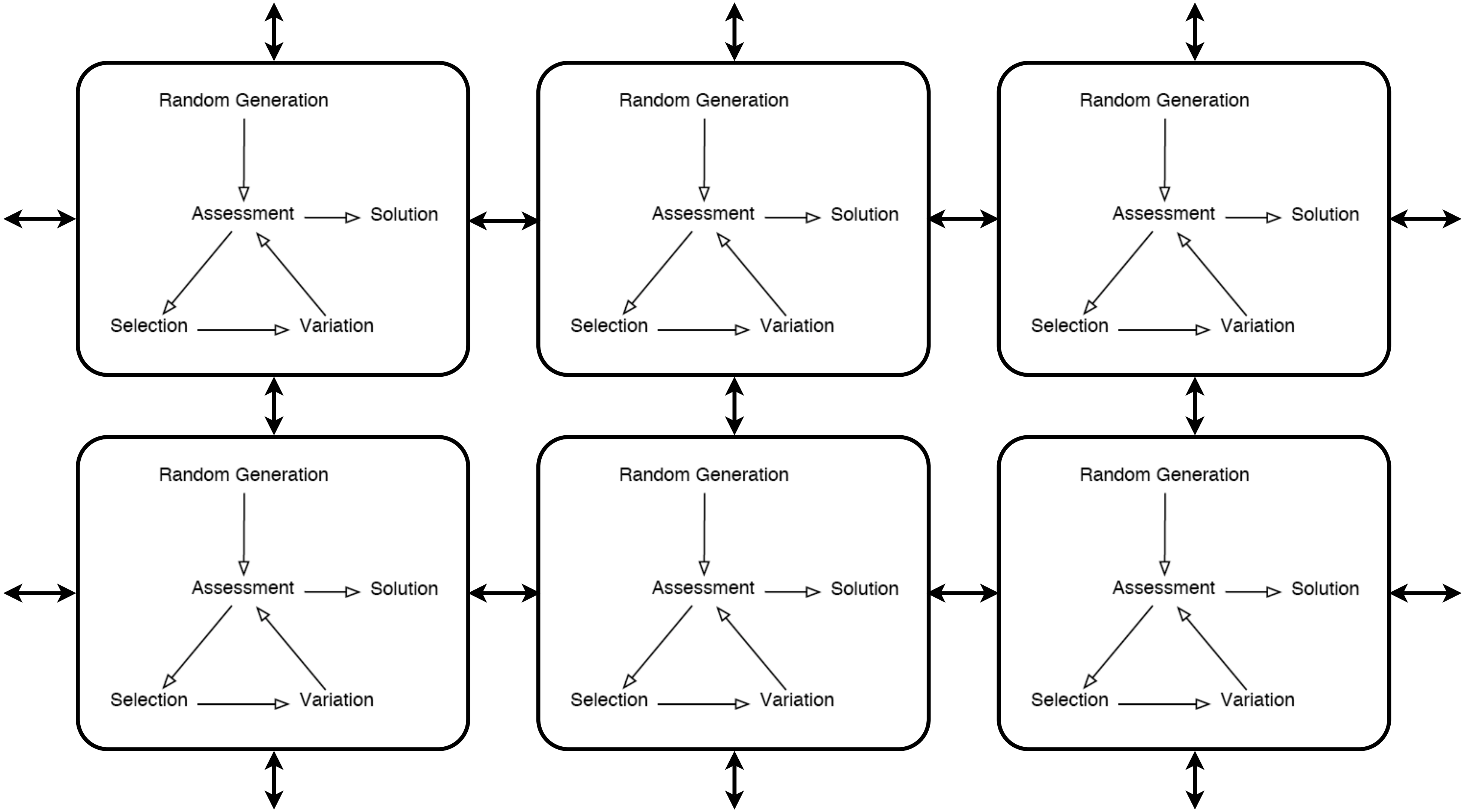
Push/PushGP

- Stack-based postfix language with one stack per type: integer, float, vector, Boolean, name, code, exec, ...
- Syntax-independent handling of multiple data types.
- Code/exec stacks support evolution of subroutines (any architecture), recursion, evolved control structures, and meta-evolutionary mechanisms.
- Several active implementations/projects (Lisp C++, Java: Psh and others, Python: Nudge [a little Push], ...)

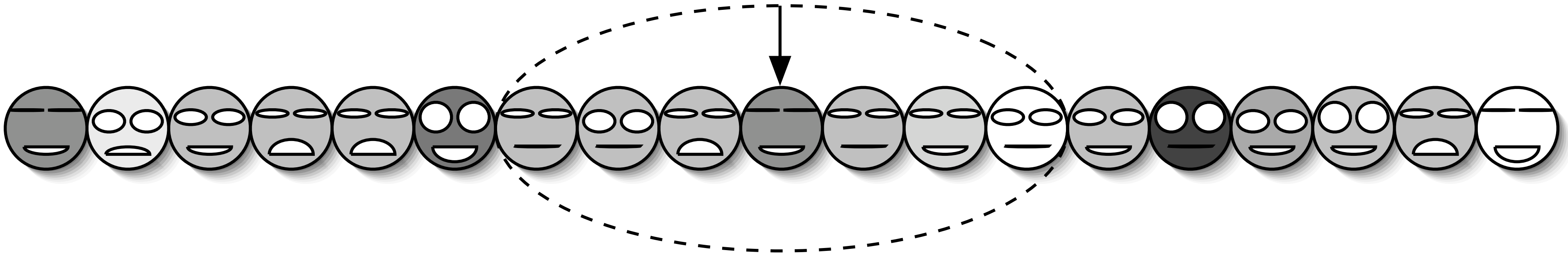
Code Generation/Mutation

- Sean Luke's PTC1, PTC2 used in some runs
- Fair mutation (Langdon, Crawford-Marks, ...) used in some runs

Islands and Migration



Trivial Geography



See *Genetic Programming Theory and Practice III* (2005)

Selection Methods

- ECJ's parsimony-based selection: with some probability select by size rather than fitness
- Alpha-inverted selection: define the "alpha group" to be the group of programs having the population's best fitness. Then use a larger tournament size the smaller the alpha group is (formulae in paper)

HAH

- Historically Assessed Hardness
- Count performance on “hard” fitness cases more more than performance on easy fitness cases, where hardness is based on solution rates over the history of the run
- Formula in paper
- See also *Genetic Programming Theory and Practice VI* (2006)

Results

- Discriminators for A_1, A_2, A_3, A_4, A_5
- Mal'cev and majority terms for B_1
- Parameter tables and result terms in paper
- Example discriminator term for A_1:

$$((((((((x*(y*x))*x)*z)*(z*x))*((x*(z*(x*(z*y))))*z))*z)*z)*(z*((((x*(((z*z)*x)*(z*x)))*x)*y)*(((y*(z*(z*y)))*(((y*y)*x)*z))*(x*(((z*z)*x)*(z*(x*(z*y)))))))))$$

Assessing Significance

Relative to prior methods:

- Uninformed search:
  - Exhaustive: analytical (expected value) and empirical search time comparisons
  - Random: analytical (expected value) and empirical search time comparisons
- Primality method: empirical term size comparisons

Expected Value Analysis

Since $\mathrm{Exp}(X)$ is the weighted sum of the values of X,

$$\mathrm{Exp}(X) = \sum_{j=1}^{\infty} j p_j = \sum_{k=1}^{\infty} \sum_{j=k}^{\infty} p_j = \sum_{k=1}^{\infty} P_k \approx \sum_{k=1}^{\infty} \left(\tfrac{n-1}{n}\right)^{k-1}$$

$$= \frac{1}{1 - \frac{n-1}{n}} = n.$$

We recapitulate this conclusion as follows.

The expected value $\mathrm{Exp}(X)$ *of the number* X *of trials required to find a term representing the function* f *is approximately the size* $n = |A|^{|B|}$ *of the search space* A^B *of all functions from* B *to* A.

- Verified via empirical results with random search and exhaustive search

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

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_I)

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
- Because there were no prior methods for generating practical terms in practical amounts of time, GP has provided the first solution to a previously open problem in the field

Potential Impact

These results are in an foundational area of pure mathematics with:

- A long history
- Many outstanding problems of theoretical significance and quantifiable difficulty
- Applications across the sciences

Conclusions

- Using GP, we have improved significantly on extensive past efforts of both humans and machines to solve problems related to finite algebras
- This is an important and previously unexplored application area for GP, with many open problems and quantitative measures of success