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

ABSTRACT

We describe the application of genetic programming (GP) to a problem in pure mathematics, in the study of finite algebras. We document the production of human-competitive results in the discovery of particular algebraic terms, namely *discriminator*, *Pixley*, *majority* and *Mal'cev* terms, showing that GP can exceed the performance of every prior method of finding these terms in either time or size by several orders of magnitude. Our terms were produced using the ECJ and PushGP genetic programming systems in a variety of configurations. We compare the results of GP to those of exhaustive search, random search, and algebraic methods.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming—*program synthesis*; I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms—*algebraic algorithms*

General Terms

Algorithms, Experimentation, Performance

Keywords

ECJ, genetic programming, finite algebras, PushGP

1. INTRODUCTION

Genetic programming (GP) has the potential for application to many areas of mathematics. In particular, any area in which open questions can be resolved by discovering relatively small equations, terms, or finite structures is a promising area for the application of GP. For some such questions the very existence of a constraint-satisfying equation, term or structure may settle the issue under study, while for others the specific properties of discovered solutions may have additional implications or provide additional insights.

In this paper we present results from the application of GP to an area of pure mathematics, the study of finite algebras. We are not aware of significant prior results in this area. We document the discovery of particular algebraic terms that have both theoretical significance and quantifiable difficulty, and we argue that the results are human-competitive according to widely promulgated criteria.

In the following section we briefly describe the relevant mathematical context and the specific problems solved. In Section 3 we describe the GP techniques that we used to produce our results. The results themselves are presented in Section 4. Section 5 discusses the significance of the results, including our claims of human-competitive performance. In Section 6 we summarize our findings and discuss prospects for further applications of the presented methods.

2. FINITE ALGEBRAS

For the sake of this paper, and within the over-arching area of mathematics known as *universal algebra*, an *algebra* $\mathbf{A} := \langle A, F\rangle$ consists of an underlying set A and an associated collection F of operations $f : A^r \to A$ on A. The natural number r is called the *arity* of the operation f. Universal algebra is a significant branch of mathematics with a long history (for example see [8]), important sub-disciplines such as *group theory* [20], and applications to several areas of science and engineering.

We use the term *finite algebra* to refer to an algebra in which the underlying set is finite. The finite algebra most familiar to most computer scientists is the ordinary two-element Boolean algebra, $\mathbf{B} := \langle\{0,1\}, \wedge, \vee, \neg\rangle$, in which the underlying set is $\{0,1\}$ and the associated operations are the Boolean operators AND ($\wedge$), OR ($\vee$) and NOT ($\neg$). These operations can be defined by tables:

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

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

	$\neg$
0	1
1	0

A well-known and convenient feature of Boolean algebra is the fact that this small set of operations is sufficient for representing *all possible* operations on $\{0,1\}$. For example, consider the (randomly chosen) ternary operation $q : \{0,1\}^3 \to \{0,1\}$ given by $q(x,y,z)$ is 1 if (x,y,z) is (0,0,1), (1,0,1) or (1,1,1); otherwise $q(x,y,z) = 0$. Then q is represented as a term by

$$q(x,y,z) = (\neg x \wedge \neg y \wedge z) \vee (x \wedge \neg y \wedge z) \vee (x \wedge y \wedge z).$$

More generally, we say that a finite algebra is *primal* if every operation on its underlying set can be represented using a term that is composed only of the operations in the algebra. For example, **B** is a primal algebra.

Over the past fifty years universal algebraists have found that the existence of a number of special kinds of terms on an algebra tell us a great deal about the properties of the class of algebras (called a *variety*) that it generates. For example, a natural generalization of the classical Jordan Holder Theorem and the Krull Schmidt Theorem for groups will hold in the variety generated by an algebra having a *Mal'cev term* [17], that is, a ternary term $m(x, y, z)$ satisfying

$$m(x, x, y) \approx m(y, x, x) \approx y.$$

A natural analog of the Stone Duality Theorem for Boolean algebras holds in the variety generated by any finite algebra having a *majority term* [2, Theorem 3.3.8], that is, a ternary term $j(x, y, z)$ satisfying

$$j(x, x, y) \approx j(y, x, x) \approx j(x, y, x) \approx x.$$

The Chinese Remainder Theorem for the ring of integers has a natural extension in the variety generated by an algebra that has a *Pixley term* [9, Theorem 3.3.1], that is, a ternary term $p(x, y, z)$ satisfying

$$p(x, x, y) \approx p(y, x, x) \approx y \text{ and } p(x, y, x) \approx x.$$

The ternary operation on an algebra **A** given by

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

is called the (ternary) *discriminator operation.* A *discriminator term* for **A** is a ternary term that represents t^A. If **A** has a discriminator term, then every non-trivial finite algebra in the variety generated by **A** is isomorphic to a direct product of subalgebras of **A**, and the variety generated by **A** has a decidable first order theory [28].

Very recent work [3] based on the *primality theorem* provides a recursive method to construct a term representing any desired operation on an algebra already known to be primal. While this is the most time efficient method currently available for constructing such terms, the resulting terms are usually extraordinarily long and often involve millions of operations even in three and four-element algebras.

In order to test the power of GP in this domain we applied it to the three and four element, single-operator algebras in Table 1. All of these algebras are known to be primal as an immediate application of a well known theorem of Rousseau [21]. But the proof of Rousseau's theorem is non-constructive, providing no practical assistance in constructing specific terms. Moreover, previously known methods to construct terms are not adequate to produce the terms we have described in a feasible amount of space and time (see Section 5). We show here that GP can be used to evolve Mal'cev, majority, Pixley and discriminator terms that are orders of magnitude shorter than those that could be produced by prior algebraic methods, and we obtain them in orders of magnitude less time than the expected or empirically discovered times for exhaustive or random search.

3. GP TECHNIQUES

In the following subsections we describe the specific GP techniques that produced the results reported in this paper.

Table 1: Primal algebras in this paper.

$\mathbf{A}_1$ $*$	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

We do this to document our methods and to allow for replication by others.[1] We do not argue here for the superiority of these techniques over any other GP techniques, and we claim no deep justification for our choices of systems or parameters (which vary significantly from run to run). We proceeded in an exploratory mode, trying different techniques and parameters to see if GP could break new ground in the study of finite algebras. We found that it could, in many different configurations, and we describe here the configurations that produced the specific results that we present. The relative efficacy of particular techniques and parameters is a subject for future research.

GP fitness cases for all problems were all of the input combinations for which the target term's definition specifies a particular output. The fitness value for a candidate term, which we sought to minimize, was the sum of the errors across all fitness cases (although in some cases these errors were individually scaled; see section 3.6).

3.1 Traditional GP in ECJ

Most of the results presented in this paper were produced using traditional, "tree-based" GP techniques [13] as implemented in the ECJ evolutionary computation system [29].[2] In this method programs are represented as Lisp-like symbolic expressions, in parenthesized prefix syntax. The mapping between these expressions and the algebraic terms that we seek is direct: algebraic operations are "nonterminals" in the GP representation and variables ranging over members of the underlying sets are "terminals." ECJ supports several advanced GP features, but aside from those described below we did not use such features in the runs described here.

3.2 PushGP

Push is a stack-based programming language designed specifically for code evolution [26]. PushGP is a GP system that produces Push programs rather than Lisp-style expressions. We used Push 3 [25] and the version of PushGP built in to the Breve simulation environment [10].[3] Push includes

[1]Full detail is available in the source code at http://hampshire.edu/lspector/gpfa-gecco-2008.

[2]http://www.cs.gmu.edu/~eclab/projects/ecj/

[3]http://www.spiderland.org/breve

features that support several novel GP techniques but these were not used in the runs reported here.

Both nonterminals and terminals are rendered in Push as instructions that take their arguments, if any, from stacks and do nothing if required arguments are not available. We represented algebraic operations as operations on integers although no arithmetic instructions were included in the instruction set. Push program results are typically read from stack tops after execution, but for the runs reported here we cached and returned the value returned by the last call to the algebraic operation; hence trailing operands had no effect. Results were automatically post-processed, removing ineffectual instructions and converting to infix syntax.

3.3 Code generators and mutation

Some of our runs used non-traditional random code generators. In particular we sometimes used the PTC1 and PTC2 algorithms, which provide enhanced control over the size, shape, and contents of randomly generated programs [15]. These algorithms take several parameters; those not provided here are available in the on-line source code. In some cases we also used a non-traditional mutation operator — *fair mutation* — in which the mutant's new subtree is guaranteed not to differ in size from the subtree that it replaces by more than a specified percentage [14].

3.4 Islands and trivial geography

For some of our experiments we launched runs simultaneously on many nodes in a Beowulf-style computer cluster and allowed the runs to proceed asynchronously with occasional migration of programs between nodes. Each run would export a set of selected programs to a randomly selected other run once per generation. Those programs became available for incorporation into the recipient node's population during that node's next reproduction phase. For related work see, for example, [6].

Some of our runs used a population-structuring scheme called "trivial geography" in which the population is viewed as having a one-dimensional (circular) spatial structure [24]. In this scheme the production of an individual for a particular location in the population is permitted to involve only parents from the neighborhood of that location, where the neighborhood is specified via a radius parameter.

3.5 Alternative selection methods

Some of our runs used ECJ's parsimony-based tournament selection method, in which "The comparison of individuals is based on fitness with probability *prob*, and it is based on size with probability $1 - prob$. For each pairwise comparsion of individuals, [it] randomly decides whether to compare based on fitness or size."[4]

In some of our runs we determined selection tournament sizes dynamically, in inverse relation to the size of the population's "alpha group" in the previous generation, where the alpha group is defined as the set of individuals having the population's best fitness. We call this *alpha-inverted selection.* Specifically, if A is the alpha group, P is the entire population, and s is a scaling factor ≥ 1, we calculate α as:

$$\alpha = \min(1, s\frac{|A|}{|P|})$$

[4]http://www.cs.gmu.edu/~eclab/projects/ecj/docs/

We then calculate the tournament size τ using α and a maximum tournament size parameter m as:

$$\tau = \max(2, (m+1) - \alpha m)$$

This produces tournament sizes ranging from 2 up to m, with the largest tournaments used in the context of the smallest alpha groups and vice versa.

3.6 Historically Assessed Hardness

In some of our runs we scaled the error for each fitness case based on the proportion of individuals that produced the correct answer for that case in the previous generation. This general strategy, of scaling errors by *historically assessed hardness*, is related to prior techniques such as *implicit fitness sharing*; see [11]. In the variant of the technique used here, if the error of an individual on a particular fitness case is e, and if the solution rate for that case across the entire population in the previous generation was r, then the fitness contribution, f, of this case for this individual is calculated (with scaling factor h, normally 1) as:

$$f = e(1 + h(1 - r))$$

The fitness of an individual (for which lower is better) is then the sum of these fitness contributions across all fitness cases. The result is that errors are more "expensive" for rarely-solved cases. As a consequence, individuals that perform better on those cases will have a fitness advantage.

4. RESULTS

We present here the results of several runs that produced algebraic terms of interest (see Section 2) using the techniques described in Section 3. We conducted runs in addition to those reported here, including some that failed to find the target terms and also others that succeeded. Here we present only a representative sample of our successes, including some of our most significant results; we discuss their significance in Section 5.

4.1 $\mathbf{A_1}$

In this run we used ECJ and the parameters shown in Table 2 to evolve a discriminator term for $\mathbf{A}_1$. The system found the following 100% correct term containing 39 operations in generation 3,210:

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

4.2 $\mathbf{A_2}$

In this run we used ECJ and the parameters shown in Table 3 to evolve a discriminator term for $\mathbf{A}_2$. The system found the following 100% correct term containing 51 operations in generation 59,523:

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

4.3 $\mathbf{A_3}$

In this run we used ECJ and the parameters shown in Table 4 to evolve a discriminator term for $\mathbf{A}_3$. The system

Table 2: Parameters used to evolve a discriminator term for $\mathbf{A}_1$ with ECJ.

Population	
Population size	1,000
Processors (islands)	1
Trivial geography	no
Individuals	
Code generator	PTC1
Nonterminals	$*$ (representing $\mathbf{A}_1$)
Terminals	x, y, z
Propagation	
Maximum generations	10,000
Crossover rate	42.5%
Mutation rate	52.5% (standard)
Duplication rate	5%
Tournament size	6 (constant)
Parsimony selection	2% (only for duplication)
Historically assessed hardness	no

Table 3: Parameters used to evolve a discriminator term for $\mathbf{A}_2$ with ECJ.

Population	
Population size	2,000
Processors (islands)	1
Trivial geography	no
Individuals	
Code generator	PTC1
Nonterminals	$*$ (representing $\mathbf{A}_2$)
Terminals	x, y, z
Propagation	
Maximum generations	100,000
Crossover rate	42.5%
Mutation rate	52.5% (standard)
Duplication rate	5%
Tournament size	7 (constant)
Parsimony selection	2% (only for duplication)
Historically assessed hardness	no

found the following 100% correct term containing 20 operations in generation 34:

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

4.4 $\mathbf{A}_4$

In this run we used ECJ and the parameters shown in Table 5 to evolve a discriminator term for $\mathbf{A}_4$. The system found the following 100% correct term containing 68 operations in generation $35,039$:

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

4.5 $\mathbf{A}_5$

In this run we used ECJ and the parameters shown in Table 6 to evolve a discriminator term for $\mathbf{A}_5$. The system found the following 100% correct term containing 33 operations in generation 49:

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

Table 4: Parameters used to evolve a discriminator term for $\mathbf{A}_3$ with ECJ.

Population	
Population size	90,000
Processors (islands)	1
Trivial geography	yes, radius 20
Individuals	
Code generator	PTC2
Nonterminals	$*$ (representing $\mathbf{A}_3$)
Terminals	x, y, z
Propagation	
Maximum generations	51
Crossover rate	10%
Mutation rate	70% (fair ±30%)
Duplication rate	20%
Tournament size	alpha-inverted, max 10
Parsimony selection	no
Historically assessed hardness	no

Table 5: Parameters used to evolve a discriminator term for $\mathbf{A}_4$ with ECJ.

Population	
Population size	3,000
Processors (islands)	1
Trivial geography	no
Individuals	
Code generator	PTC1
Nonterminals	$*$ (representing $\mathbf{A}_4$)
Terminals	x, y, z
Propagation	
Maximum generations	100,000
Crossover rate	42.5%
Mutation rate	52.5% (standard)
Duplication rate	5%
Tournament size	7 (constant)
Parsimony selection	2% (only for duplication)
Historically assessed hardness	no

4.6 $\mathbf{B}_1$

4.6.1 *Mal'cev terms*

ECJ.

In this run we used ECJ and the parameters shown in Table 7 to evolve a Mal'cev term for $\mathbf{B}_1$. The system found the following 100% correct term containing 28 operations in generation 256:

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

Similar Mal'cev terms were found for two other 4-element binary algebras.

PushGP.

In this run we used PushGP and the parameters shown in Table 8 to evolve a Mal'cev term for $\mathbf{B}_1$. The system found the following 100% correct term containing 18 operations (making it the shortest term we have found for this problem) in generation 64 on one of the 39 processors:

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

Table 6: Parameters used to evolve a discriminator term for $\mathbf{A}_5$ with ECJ.

Population	
Population size	70,000
Processors (islands)	1
Trivial geography	yes, radius 100
Individuals	
Code generator	PTC2
Nonterminals	$*$ (representing $\mathbf{A}_5$)
Terminals	x, y, z
Propagation	
Maximum generations	51
Crossover rate	10%
Mutation rate	70% (fair ±50%)
Duplication rate	20%
Tournament size	alpha-inverted, max 10
Parsimony selection	no
Historically assessed hardness	no

Table 7: Parameters used to evolve a Mal'cev term for $\mathbf{B}_1$ with ECJ.

Population	
Population size	70,000
Processors (islands)	1
Trivial geography	yes, radius 70
Individuals	
Code generator	PTC2
Nonterminals	$*$ (representing B_1)
Terminals	x, y, z
Propagation	
Maximum generations	51
Crossover rate	10%
Mutation rate	65% (fair ±50%)
Duplication rate	25%
Tournament size	alpha-inverted, max 10
Parsimony selection	no
Historically assessed hardness	no

4.6.2 *Majority term*

In this run we used ECJ and the parameters shown in Table 9 to evolve a majority term for $\mathbf{B}_1$. The system found the following 100% correct term containing 63 operations in generation 289:

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

5. SIGNIFICANCE

Over the past fifty years algebraists have found that finite algebras which have an associated Mal'cev, majority, Pixley or discriminator term generate varieties of algebras with many interesting properties. This knowledge has motivated an ongoing search for each of these terms. Striking theorems of Davies [5] and Murskiĭ [18] showed that, in an appropriate sense, most finite algebras harbor all of these terms. Yet the proofs of these theorems give no indication as to which algebras these will be or how to construct these terms if they do exist. Practical and effective methods to construct these terms would be recognized as very significant new information by algebraists.

Two approaches encompass the most effective methods previously known to find these terms: uninformed search and construction via the primality theorem. In this section we review these approaches and demonstrate that our GP results are human-competitive because they surpass all previous techniques in significant ways.

Table 8: Parameters used to evolve a Mal'cev term for $\mathbf{B}_1$ with PushGP.

Population	
Population size	50,000/processor
Processors (islands)	39
Trivial geography	yes, radius 100
Individuals	
Code generator	Push standard
Instructions	$*$ (representing B_1), X, Y, Z
Propagation	
Maximum generations	10,000
Crossover rate	40%
Mutation rate	40% (fair ±200%)
Duplication rate	5%
Deletion rate	10%
Migration rate	5%
Tournament size	alpha-inverted, max 10
Parsimony selection	no
Historically assessed hardness	yes

Table 9: Parameters used to evolve a majority term for $\mathbf{B}_1$ with ECJ.

Population	
Population size	100,000
Processors (islands)	1
Trivial geography	yes, radius 20
Individuals	
Code generator	PTC2
Nonterminals	$*$ (representing B_1)
Terminals	x, y, z
Propagation	
Maximum generations	51
Crossover rate	20%
Mutation rate	60% (fair ±30%)
Duplication rate	20%
Tournament size	alpha-inverted, max 10
Parsimony selection	no
Historically assessed hardness	no

5.1 Uninformed search

The problem of finding the different terms we are seeking can be described uniformly as follows. Assume that we are given defining tables for the operations of a finite algebra $\mathbf{A}$, an integer $r > 0$, a subset $B \subseteq A^r$ of the set A^r of all r-tuples of elements of A and a function $f : B \to A$ that is known in advance to be a term operation. Our goal is to find some particular term defining f on B. For example, a Mal'cev term is a term that agrees with the function $f(a, a, b) = f(b, a, a) = b$ on the set $B = \{(a, a, b), (b, a, a) \mid a, b \in A\} \subseteq A^3$.

A direct approach to finding a term for f is to enumerate all r-ary terms in an infinite sequence of increasing size, starting with the smallest terms and testing each one to see if it represents f. We call this *exhaustive search.* As we do the search, each term defines a choice of one of the $n := |A|^{|B|}$ different functions from B to A, where $|A|$ and $|B|$ are the numbers of elements in A and B, respectively. A popular tool that enumerates terms in this way is the **UACalc** program [7].

An alternative is to generate terms in random order, for example by first picking a size and then generating a term of that size using an algorithm such as Knuth's "R" [12]. We call this *random search.* Because exhaustive and random search are equally uninformed — in the sense that we know nothing about how the different functions are distributed in the sequences of terms generated by these methods — our mathematical analysis will apply to both. But exhaustive and random search provide different opportunities for empirical verification (see below).

Our aim is to find a reasonable estimate of the expected number of trial terms we will need to construct and test before finding f. Let $t_1, t_2, t_3, \dots$ be any enumeration of terms produced by either an exhaustive search, a random search, or any other uninformed search. Let X be the random variable whose value is the smallest k such that t_k represents f.

The difficult part of computing the expected value $\mathrm{Exp}(X)$ of X is finding, for each k, the probability that t_k represents f. This is the proportion, among all possible defining tables for the operations of $\mathbf{A}$, of choices which result in t_k representing f. Computing this value appears to be extraordinarily difficult, even – with a few exceptions – for a single term t_k. In order to get a rough estimate of $\mathrm{Exp}(X)$, we make the simplifying assumption that the terms $t_1, t_2, t_3, \dots$ are uniformly distributed among the n possible functions they might represent, that is, that each t_k represents f with probability $\frac{1}{n}$. Thus t_k fails to represent f with probability $\frac{n-1}{n}$. For each $j, k \in \mathbb{N}$, we have

$$p_j := \mathrm{Prob}(X = j) \approx \left(\tfrac{n-1}{n}\right)^{j-1} \tfrac{1}{n}$$

and

$$P_k := \mathrm{Prob}(X \geq k) \approx \left(\tfrac{n-1}{n}\right)^{k-1}.$$

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.

We first apply this result to the search for a Mal'cev term. Notice that a ternary operation $m^{\mathbf{A}} : A^3 \to A$ is a Mal'cev function if and only if it takes the Mal'cev value on the set

$$B := \{(a, a, b), (b, a, a) \mid a, b \in A\}.$$

Here we have $r = 3$ and we count $|B| = 2|A|^2 - |A|$.

Example 5.1. *If* $\mathbf{A}$ *has a Mal'cev term, then the expected number of trials required to find one is* $\mathrm{Exp}(X) \approx |A|^{2|A|^2 - |A|}$.

Similarly, a ternary operation on A is a majority function or a Pixley function if and only if it takes the majority value or Pixley value, respectively, on each triple in the set

$$B := \{(a, a, b), (a, b, a), (b, a, a) \mid a, b \in A\}.$$

Here again $r = 3$ and we count $|B| = 3|A|^2 - 2|A|$.

Table 10: Approximate times required to find terms by uninformed search and by GP.

	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 ($4^{64} \approx 10^{38}$)	?

Example 5.2. *If* $\mathbf{A}$ *has a majority term or a Pixley term, then the expected number of trials required to find one is* $\mathrm{Exp}(X) \approx |A|^{3|A|^2 - 2|A|}$.

If we search for some specific ternary operation, such as the ternary discriminator operation, on a primal algebra $\mathbf{A}$, we have $r = 3$ and $B = A^3$.

Example 5.3. *If* $\mathbf{A}$ *is primal, then the expected number of trials required to find a term representing a given operation (such as the discriminator) is* $\mathrm{Exp}(X) \approx |A|^{|A|^3}$.

The values of $\mathrm{Exp}(X)$ grow dramatically as a function of $|A|$ in each of these examples. In Table 10 we list these values for three and four element algebras. For each case we list a plausible estimate of the time that a 3 GHz computer would require to conduct the search, conservatively estimating that the average term will require $1,000$ computer cycles to generate and test. Within this range we see that searches move from those that can be done quickly to those that are unlikely to ever become computationally feasible. On the right we estimate the times required for single-processor GP runs to find these terms, based on the run times of the specific GP searches documented in this paper.

In favor of exhaustive search we note that it is parsimonious. Since it enumerates the terms from shortest to longest it is guaranteed to produce a shortest term representing the given function. We conducted such an exhaustive search looking for a Mal'cev term for the three-element algebra $\mathbf{A}_1$. We found a minimal size term with 11 operations, and our search revealed exactly 2 such 11-operation terms. Such searches rapidly become impractical, however, and we have conducted exhaustive searches only up to size 14 (which required at least 2 months of CPU time). By comparison GP quickly found $\mathbf{A}_1$ Mal'cev terms of various sizes; one run on a single CPU took one minute to evaluate $840,000$ terms and found a Mal'cev term of size 12, while an independent run quickly found a term of size 24.

To empirically test our analytical results we also conducted random searches for discriminator and Mal'cev terms on $\mathbf{A}_1$. In our first test we generated approximately 10^{11} terms with sizes uniformly distributed from 12 to 50, using Knuth's tree-generation algorithm "R" [12]. We tested each term to see if it was a discriminator term for $\mathbf{A}_1$. Example 5.3 predicts that a successful search would require approximately $3^{27} \approx 10^{13}$ trial terms. Our test of approximately 1% of this number indeed revealed no discriminator term. This search required over a week of CPU time on hardware in the 2.4GHz-3.0GHz range.

Table 11: Sizes of terms for $\mathbf{A}_1$ from the primality theorem and from GP.

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

In our second test of random search, we looked for $\mathbf{A}_1$ Mal'cev terms. Example 5.1 predicts the first success coming after about $3^{15} \approx 1.4 \times 10^7$ trials. We generated and tested approximately 2.2×10^{10} terms, using the same size distribution and generation algorithm as in the first test. This required roughly one day of CPU time. This test produced 64 Mal'cev terms requiring an average of about 3.4×10^8 trials each, one order of magnitude *more* than our theoretical estimate, and suggesting that the Mal'cev function is represented by about 4% as many terms as an average function.

These results are altogether consistent with our predictions. $\mathbf{A}_1$ discriminators are quite hard to find, and their routine and relatively speedy discovery by GP is therefore notable. $\mathbf{A}_1$ Mal'cev terms are comparatively easy to find, and the actual run-times of our tests support our assertion of the conservatism of the time estimates in Table 10.

5.2 Comparison to algebraic methods

Algebraists have put significant effort into finding effective methods to produce a term representing an arbitrary operation on a finite algebra from a small set of building block operations. In 1939 Słupecki did this using unary operations and a surjective binary operation that was not essentially unary [22, 23]. Werner [27](1970) used the ternary discriminator operation together with constants. Other methods were described in [28](1978) and [4](1991). The most time efficient algebraic method found to date is that of Clark, Davey, Pitkethly and Rifqui [3](2008) based on their primality theorem. These authors give an explicit formula to build a term for an arbitrary operation, and draw on UACalc [7] to generate the required building blocks.

While the primality theorem represents a significant theoretical advance, the price it extracts for simple building blocks is very large resulting terms. Using this theorem and the building block terms generated in [3] for our primal algebra $\mathbf{A}_1$, we calculated the sizes of four resulting terms and listed them in Table 11. Juxtaposed to these are the sizes of some of the equivalent terms we generated by GP. Sizes are measured as the number of operation symbols in the term.

5.3 Human-competitiveness

Since the discoveries of the significance of Mal'cev functions [16](1954), majority functions [1](1975), Pixley functions [19](1963) and the discriminator function [27](1970), there has been a keen interest among algebraists to find terms that represent these functions on individual algebras.

Two approaches embody the most effective methods previously developed for finding these terms. The first is uninformed search, of which we explored two variants: exhaustive search, in which terms are enumerated systematically from smallest to largest, and random search, in which terms within a range of sizes are generated in random order. Exhaustive search is guaranteed to produce a smallest term of the required type if such a term exists, but it has the disadvantage of requiring astronomical amounts of time, except for the very smallest algebras or the very simplest terms. Random search has the same abysmal performance but without any guarantees concerning size or success. The second approach is construction via the primality theorem of [3], which gives the most time efficient method known to describe these terms that applies to any primal algebra. But it has the disadvantage that, except for the very smallest algebras, the terms it produces have astronomical length.

We have used GP to significantly expand the range of finite algebras for which we can produce, within a few hours, terms that can be written on a few lines. We have exhibited examples of such algebras for which these terms would require many orders of magnitude more time to find by exhaustive or random search and many orders of magnitude more space to write down the terms produced by the primality theorem. In this respect we have established that GP has surpassed, by a considerable margin, all known methods of generating practical terms for these algebras. 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.

According to the rules[5] for the 2007 "Humies," which are "awards for human-competitive results produced by genetic and evolutionary computation," a result is human-competitive if it satisfies at least one of eight listed criteria. Of these, five appear to be satisfied by our results:

(B) 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.

(D) The result is publishable in its own right as a new scientific result *independent* of the fact that the result was mechanically created.

(E) 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.

(F) 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.

(G) The result solves a problem of indisputable difficulty in its field.

6. CONCLUSIONS AND PROSPECTS

The results presented here suggest that further work in this area will advance knowledge both in algebra and in GP. We have seen that GP can improve significantly on extensive past efforts of both humans and machines to solve the algebraic problems we have posed. Further refinement of our GP techniques will surely improve these solutions. At the same time, these particular algebraic problems allow us to precisely quantify our successes with GP algorithms. This offers the prospect of further developing our GP techniques by providing a critical context in which they can be tested and compared.

We used a few advanced GP techniques in producing the results reported here (see Section 3) but we have not yet

[5]http://www.genetic-programming.org/hc2007/cfe2007.html

experimented systematically with these or with other well-known techniques. Some, like automatically defined functions in traditional GP or the automatic modularization capabilities of PushGP, may allow us to scale up our results significantly if the algebraic domain shares features (in this case, regularity) with those for which these techniques have previously proven useful.

We also expect there to be many other qualitatively different applications of GP to research on finite algebras. The mapping between algebraic terms and GP representations is direct, and we expect that many open problems will be re-framable as searches for algebraic terms having certain specified properties.

7. ACKNOWLEDGMENTS

We thank our anonymous reviewers for thorough and constructive criticism. This material is based upon work supported by the National Science Foundation under Grant No. 0308540. 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.

8. REFERENCES

[1] K. Baker and A. Pixley. Polynomial interpolation and the chinese remainder theorem for algebraic systems. *Math. Z.*, 143:165–174, 1975.

[2] D. Clark and B. Davey. *Topological Dualities for the Working Algebraist.* Studies in Advanced Mathematics Series. Cambridge, UK: Cambridge University Press, 1998.

[3] D. Clark, B. Davey, J. Pitkethly, and D. Rifqui. Primality from semilattices, 2008. Preprint.

[4] B. A. Davey, V. J. Schumann, and H. Werner. From the subalgebras of the square to the discriminator. *Algebra Universalis*, 28:500–519, 1991.

[5] R. O. Davies. On n–valued sheffer functions. *Zeitschr. f. math. Logik und Grundlagen d. Math.*, Bd. 25:293–298, 1979.

[6] F. Fernandez, M. Tomassini, and L. Vanneschi. An empirical study of multipopulation genetic programming. *Genetic Programming and Evolvable Machines*, 4(1):21–51, Mar. 2003.

[7] R. Freese and E. Kiss. UACalc: The universal algebra calculator, 2007. www.cs.elte.hu/~ewkiss/software/uaprog/.

[8] G. Grätzer. *Universal Algebra.* Springer-Verlag, 1982.

[9] K. Kaarli and A. Pixley. *Polynomial Completeness in Algebraic Systems.* New York, NY: Chapman and Hall/CRC, 2001.

[10] J. Klein. BREVE: a 3d environment for the simulation of decentralized systems and artificial life. In *Proc. Artificial Life VIII, the 8th Int. Conf. on the Simulation and Synthesis of Living Sys.*, pages 329–334. The MIT Press, 2002. http://www.spiderland.org/breve/breve-klein-alife2002.pdf.

[11] J. Klein and L. Spector. Genetic programming with historically assessed hardness. In R. Riolo, T. Soule, and B. Worzel, editors, *Genetic Programming Theory and Practice VI.* Springer, in preparation.

[12] D. E. Knuth. A draft of section 7.2.1.6: Generating all trees. In *The Art of Computer Programming*, volume 4. Addison-Wesley, Pre-fascicle 4A. www-cs-faculty.stanford.edu/~knuth/fasc4a.ps.gz.

[13] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.

[14] W. B. Langdon. The evolution of size in variable length representations. In *1998 IEEE International Conference on Evolutionary Computation*, pages 633–638, Anchorage, Alaska, USA, 5-9 May 1998. IEEE Press.

[15] S. Luke. Two fast tree-creation algorithms for genetic programming. *IEEE Trans. on Evol. Comp.*, 4(3):274–283, Sept. 2000.

[16] I. A. Mal'cev. On the general theory of algebraic systems (Russian). *Mat. Sb.*, 35:3–22, 1954.

[17] R. McKenzie, G. McNulty, and W. Taylor. *Algebras, Lattices and Varieties*, volume 1. Belmont, CA: Wadsworth and Brooks/Cole, 1987.

[18] V. L. Murskiĭ. A finite basis of identities and other properties of 'almost all' finite algebras. *Problémy Kibérnétiki*, 30:43–56, 1975.

[19] A. Pixley. Distributivity and permutability of congruence relations in equational classes of algebras. *Proc. Amer. Math. Soc.*, 14:105–119, 1963.

[20] J. Rotman. *An introduction to the theory of groups.* Springer-Verlag, 1994.

[21] G. Rousseau. Completeness in finite algebras with a single operation. *Proc. Amer. Math. Soc.*, 18:1009–1013, 1967.

[22] J. Słupecki. Kryterium pełności wielowartościowych systemów logiki zdań. *Comptes rendus des séances de la Société des Lettres de Varsovie, Classe III*, 32 Année:102–109, 1939.

[23] J. Słupecki. A criterion of fullness of many-valued systems of propositional logic. *Studia Logica*, 30:153–157, 1972.

[24] L. Spector and J. Klein. Trivial geography in genetic programming. In T. Yu, R. L. Riolo, and B. Worzel, editors, *Genetic Programming Theory and Practice III*, volume 9 of *Genetic Programming*, chapter 8, pages 109–123. Springer, 2005.

[25] L. Spector, J. Klein, and M. Keijzer. The Push3 execution stack and the evolution of control. In *Proc. Gen. and Evol. Comp. Conf.*, volume 2, pages 1689–1696. ACM Press, 2005.

[26] L. Spector and A. Robinson. Genetic programming and autoconstructive evolution with the Push programming language. *Genetic Programming and Evolvable Machines*, 3(1):7–40, Mar. 2002.

[27] H. Werner. Eine Charakterisierung funktional vollständiger Algebren. *Archiv d. Math.*, 21:383–385, 1970.

[28] H. Werner. *Discriminator-Algebras: Algebraic Representation and Model Theoretic Properties.* Akademie-Verlag, 1978.

[29] G. C. Wilson, A. McIntyre, and M. I. Heywood. Resource review: Three open source systems for evolving programs–lilgp, ECJ and grammatical evolution. *Genetic Programming and Evolvable Machines*, 5(1):103–105, Mar. 2004.