International Journal of Algebra and Computation
Vol. 26, No. 6 (2016) 1141–1176

DOI: 10.1142/S021819671650048X

World Scientific
www.worldscientific.com

Evolution of algebraic terms 2: Deep drilling algorithm

David M. Clark
Mathematics Department, SUNY New Paltz
New Paltz, New York 12561, USA
clarkd@newpaltz.edu

Maarten Keijzer
Chordiant, Utrecht
Amsterdam 3524CX, The Netherlands
mkeijzer@xs4all.nl

Lee Spector
School of Cognitive Science
Hampshire College
Amherst, MA 01002 USA
lspector@hampshire.edu

Received 9 September 2015
Accepted 21 June 2016
Published 12 August 2016

Communicated by K. Keames

The Deep Drilling Algorithm (DDA) is an efficient non-evolutionary algorithm, extracted from previous work with evolutionary algorithms, that takes as input a finite groupoid and an operation over its universe, and searches for a term representing that operation. We give theoretical and experimental evidence that this algorithm is successful for all idemprimal term continuous groupoids, which appear to be almost all finite groupoids, and that the DDA is seriously compromised or fails for most finite groupoids not meeting both of these conditions. See our online version of the DDA at http://hampshire.edu/lspector/dda.

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

0. Introduction

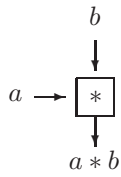
This paper, together with [13, 2], were originally motivated by an intriguing question about primal algebras. Recall that a nontrivial finite algebra is **primal** if every operation on its underlying set is a term operation. The well known characterization

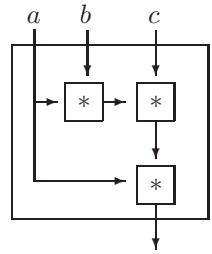
$\vec{d} \in \{0,1,2\}^3$

$\vec{d}$:	000	001	002	010	011	012	020	021	022	100	101	102	110	111	112	120	121	122	200	201	202	210	211	212	220	221	222
$g(\vec{d})$	2	2	1	1	2	2	2	2	1	0	1	1	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0

$*$	0	1	2
0	2	1	2
1	1	0	0
2	0	0	1

$\mathbf{A}_1 := \langle \{0,1,2\}, * \rangle$





$t(x,y,z) = x * ((x * y) * z)$

$t^{\mathbf{A}_1}(a,b,c) = a * ((a * b) * c)$

Fig. 1. A primal groupoid $\mathbf{A}_1$, a target operation g, a term $t(x,y,z)$ representing g, a $*$-gate and a switching circuit built from $*$-gates to calculate g.

of primal algebras given by Rosenberg [10] was shown by Rousseau to considerably simplify in a special case.

Theorem 1 ([11]). *A nontrivial finite algebra that has exactly one fundamental operation which is at least binary is primal if and only if it has no proper subalgebras and no nontrivial automorphisms or congruences.*

Using Rousseau's Theorem it is very easy to produce primal groupoids. For example, $\mathbf{A}_1$ in Fig. 1 is quickly seen to be primal. This means that, if we define an arbitrary operation g on $\{0,1,2\}$, like the one shown at the top of Fig. 1, then we are assured of the existence of a term representing g. Given a primal groupoid $\mathbf{G}$ and an operation g on G, we would like to answer the following question.

Is there a practical method to find a term representing g?

In Sec. 1, we will see that neither of two previously available methods is practical. One will take an unfeasible amount of time while the other will produce an unfeasibly long term. Yet we know from the outset that such a term exists.

For example, the term $t(x,y,z)$ in Fig. 1 represents the operation g defined at the top. The relationship between a term and its term operation has a physical interpretation in electronics that was first recognized by Shannon [12] in his work with Boolean operations. We can think of building physical gates from the $*$-table for $\mathbf{A}_1$ that take inputs a and b from A_1 and output $a * b$. We can then use any groupoid term t as a design for a switching circuit built from these gates that will calculate the term operation $t^{\mathbf{A}_1}$. For the term $t(x,y,z)$ in Fig. 1, we have illustrated the corresponding circuit that calculates the original operation $g = t^{\mathbf{A}_1}$.

In this context, a term can be viewed as a linear notation to describe a switching circuit. If we think of an operation as a performance specification for a switching circuit, then the problem of finding a term to represent an operation

is equivalent to the problem of finding a switching circuit to realize a performance specification.

The Deep Drilling Algorithm (DDA) will solve this problem by solving a more general problem. Fix an integer $k > 1$. A k-ary **array** over a finite groupoid G is a map $A : G^k \to 2^G$, that is, a G^k-indexed sequence of subsets of G. If $\vec{d} \in G^k$, then $A(\vec{d})$ is called the **component** of A associated with $\vec{d}$. A **solution** to a k-ary array A is a k-ary term $t(\vec{x})$ in variables $\vec{x} := (x_0, x_1, \ldots, x_{k-1})$ such that $t(\vec{d}) \in A(\vec{d})$ for all $\vec{d} \in G^k$. The Term Generation Problem (TGP) for **G** is the problem of finding solutions to those arrays over G that do have a solution.

For example, if $g : G^k \to G$ is any k-ary operation, then $\langle g \rangle$ is the k-ary array defined by $\langle g \rangle(\vec{d}) := \{g(\vec{d})\}$ for each $\vec{d} \in G^k$. Here, we see that a term is a solution to the array $\langle g \rangle$ if and only if it is a term representing the operation g. As a different example, suppose we wanted a term t representing a function $f : B \to G$ where $B \subseteq G^k$, such as a Mal'cev term, a Pixley term or a majority term. We would then solve the array A defined for $\vec{d} \in G^k$ by

$$A(\vec{d}) := \begin{cases} \{f(\vec{d})\}, & \text{if } \vec{d} \in B, \\ G, & \text{otherwise.} \end{cases}$$

We give an online version of the DDA at (http://hampshire.edu/lspector/dda).

Since we do not include in the TGP the task of deciding if an operation is a term operation, the TGP is particularly relevant for finite groupoids for which we know in advance which operations are term operations. This is true for an important class of groupoids that includes the primals. An element $e \in G$ of an algebra **G** is an **idempotent** if $\{e\}$ is a subalgebra. If **G** has an idempotent, then it is not primal since an operation that does not preserve $\{e\}$ is not a term operation. A nontrivial finite algebra **G** is **idemprimal** (**IPr**) if every operation that does preserves its idempotents is a term operation. Much of our study will focus on idemprimal groupoids because they have three additional properties that are particularly relevant. Recall that an n-element primal algebra has n^{n^k} different k-ary term operations.

(1) An n-element idemprimal groupoid with i idempotents has $n^{n^k - i}$ different k-ary term operations.
(2) We will see later that success of the DDA usually *requires* the groupoid to be idemprimal.
(3) Almost all finite groupoids are idemprimal.

Item (1) shows that the number of term operations of an idemprimal groupoid is large relative to the maximum possible n^{n^k}, making the task of finding a term representing a particular operation a relatively challenging case of the TGP. Item (2) shows that the question as to whether or not an operation is a term operation will usually be trivial when we use the DDA. Item (3) shows that, in a statistical sense, (2) is a mild constraint. It comes from a well known theorem.

$\mathbf{Pi} :=$
$\langle \{0,1,2,3,4\}, * \rangle$

$*$	0	1	2	3	4
0	1	4	1	0	4
1	2	1	0	3	0
2	3	4	2	4	3
3	2	3	3	4	1
4	2	1	4	3	3

Fig. 2. A 5-element groupoid.

Theorem 2 ([8]). *Let Op be a finite type with at least one operation that is at least binary. Then the proportion of algebras* $\mathbf{G} := \langle \{0,1,\ldots,n-1\}, Op^{\mathbf{G}} \rangle$ *that are idemprimal approaches one as n approaches infinity.*

As an illustration of the TGP, consider the task of deciding if a given groupoid is idemprimal. For example, is the groupoid $\mathbf{Pi} = \langle \{0,1,2,3,4\}, * \rangle$ shown in Fig. 2 idemprimal? This groupoid was chosen as a test case by filling the $*$-table with the first 25 decimal digits of π, each reduced modulo 5. The first step is to decide if we should try to show that it is idemprimal or to show that it is not. Unless there is some deep negative correlation between idemprimality and the digits of π, Murskiĭ's Theorem suggests that we try to show that it is. We begin with the standard characterization of idemprimality.

Theorem 3 ([9, 14]). *A nontrivial finite algebra is idemprimal if and only if it has no nontrivial subalgebras or automorphisms and it has among its term operations the ternary discriminator operation*

$$d(a,b,c) := \begin{cases} c & \text{if } a = b \\ a & \text{if } a \neq b. \end{cases}$$

We can easily check that **Pi** has exactly two idempotents, 1 and 2, and that it has no nontrivial subalgebras or automorphisms. It would just remain to find a discriminator term for **Pi**. But, assuming it does have one, this would bring us back to a frightfully difficult instance of the TGP. It would mean finding a discriminator term among terms representing $5^{5^3-2} \approx 10^{86}$ different ternary term operations!

We will judge our success against the two prior methods for solving the TGP, both of which pose practical difficulties. It is often convenient to use the ternary discriminator operation as a benchmark for testing algorithms. It is uniformly defined on all groupoids, and experience has revealed that it is one of the more difficult terms to find.

1. *Primality Test.* Freese *et al.* have provided a freeware program, *UACalc* [6], that will, among other things, take an n-element algebra $\mathbf{G}$ as input and either report that $\mathbf{G}$ is not primal or list $n+2$ small terms, called **primality terms**, from which a term for an arbitrary operation on G can be recursively built. The recursive algorithm to do this comes with the "Primality Test" of Clark *et al.* [3] and is

reviewed in Sec. 1. For primal groupoids with up to five elements, *UACalc* instantly returns these primality terms. But the recursive process tends to generate from them rather lengthy terms. For example, Rousseau's Theorem tells us immediately that the 3-element groupoid $\mathbf{A}_1$ in Fig. 2 and the 5-element groupoid **NQ** in Fig. 7 are primal. We will see in Sec. 1 that the resulting discriminator term for $\mathbf{A}_1$ can have over 10^{18} variable occurrences, and for **NQ** over 10^{69} variable occurrences.

2. *Random Search.* Another option to find a term for a k-ary operation is to construct k-ary terms in a sequence, testing each one to see if it generates the target operation. If the probability of generating all k-ary term operations approaches one, we will eventually hit the target. If terms are generated in order of non-decreasing length, this method is guaranteed to return a solution of minimal length. But this method tends to take a long time, even for small groupoids. In Sec. 1, we will see that the required search of the $3^{3^3} \approx 10^{13}$-element space of ternary term operations on $\mathbf{A}_1$ for a discriminator term requires an estimated expected time of about one month. For the $5^{5^3-2} \approx 10^{86}$-element search space for **Pi**, the estimate jumps to 10^{72} years(!)

Our goal is to find solutions to the TGP that will produce human scale terms in human scale time.

In 2007, Clark and Spector initiated efforts to use evolutionary computation (EC) methods to find solutions to the TGP. Briefly, EC refers to any computational method that finds solutions to problems by simulating biological evolution on a computer. These methods are extremely diverse, reflecting the great diversity of evolutionary mechanisms occurring in nature. For an overview of EC, see Eiben and Smith [5] or Koza [7].

Since 2004, *the Annual ACM Genetic and Evolutionary Computation Conference* has held a competition for results obtained by applications of EC in any field that were viewed as "human competitive" in the sense that some serious ingenuity would have been required for a human to otherwise obtain them. The first place in the 2008 GECCO competition went to Spector *et al.* [13]. This entry first demonstrated that EC could be used to quickly find small discriminator terms on 3-element primal groupoids in cases where neither the Primality Test nor exhaustive search would be practical.

A typical result of [13] was a discriminator term for $\mathbf{A}_1$. Improving dramatically on 10^{18} variable occurrences and a one month search, they found the following discriminator term for $\mathbf{A}_1$ with 40 variable occurrences in under five minutes.

$$\begin{aligned} t(x,y,z) = {} & (((((((x*(y*x))*x)*z)*(z*x))*((x*(z*(x*(z*y))))*z))*z)*z) \\ & *(z*((((x*(((z*z)*x)*(z*x)))*x)*y)*(((y*\ (z*(z*y))) \\ & *(((y*y)*x)*z))*(x*(((z*z)*x)*(z*(x*(z*y))))))))). \end{aligned}$$

Search space sizes grow dramatically as a function of groupoid size (see Fig. 3 at the end of Sec. 1). As a result, the evolutionary algorithm of [13] was not able to

find a discriminator term in the vastly larger search space of $4^{4^3} \approx 10^{38}$ ternary term operations on any 4-element primal groupoid.

After GECCO 2008 Keijzer joined the effort, and we began to find better evolutionary algorithms that improved on the proof-of-concept results of [13]. These efforts resulted in the co-evolutionary algorithm of [4] that finds terms in much larger search spaces. It draws inspiration from sexual reproduction in nature, co-evolving a population of 'male' terms and a population of 'female' terms whose joint offspring come progressively closer to the target operation.

The present paper illustrates a valuable possible outcome of EC. We are often led to use evolutionary algorithms to find solutions to problems when we are unable, after substantial effort, to find direct algorithms to construct solutions. Through a careful study of those evolutionary algorithms, we can sometimes understand how, why and when they will work well enough to extract from them a more direct non-evolutionary algorithm to reach the same end.

Our DDA is an example of this outcome. Instead of sifting through successive populations of terms in search of a solution, as is done by our evolutionary algorithms, it constructs a single term by drilling deeply into each branch of the term tree until it finds a suitable variable that can be placed at the branch's end. We will see that the DDA and the algorithm of [4] depend on the same properties of the underlying groupoid for their success.

To illustrate the DDA, we checked that **Pi** is indeed idemprimal by finding a discriminator term with 1617 variable occurrences in 12.74 s on our very first run. While this term is large by normal standards, we find it a sizable improvement over both a term of length 10^{69} and a 10^{72}-year search.

$$
\begin{aligned}
t(x,y,z) = {} & (((((x*(x*x))*((y*(y*x))*((((x*x)*(x*z))*((y*z)\\
& *(((((y*x)*x)*x)*((z*z)*((x*((x*x)*x))*(((x*x)*(x*y))\\
& *(((y*z)*(x*x))*((((y*y)*(y*x))*((x*z)*(x*y)))\\
& *((((((z*x)*(y*y))*(((((x*(y*x))*((x*(y*z))*((y*x)*z)))\\
& *((y*x)*(x*y)))*(((y*y)*z)*((((y*x)*(x*x))\\
& *(((((((((x*x)*x)*((((((((z*x)*z)*(((y*y)*(y*x))\\
& *((((y*z)*y)*(z*((x*x)*x)))*((x*x)*x))))*((x*x)*x))\\
& *((x*z)*y))*(z*(x*x)))*((x*y)*x))*((y*y)*(y*y))))\\
& *(z*(z*x)))*(z*(z*y)))*(((x*x)*y)*x))*z)*(y*(z*y))))\\
& *((x*(x*x))*z))))*((y*z)*(x*x))))*((z*y)*y))*(z*(y*y)))\\
& *(y*(z*x)))))))))*(z*((x*x)*x)))))*(((x*(y*y))*(((z*z)*x)\\
& *((x*y)*y)))*((y*x)*z)))))*((((((((((z*x)*(x*y))*(((((((y*z)
\end{aligned}
$$

$$*\,(z*(x*y)))*((((((x*z)*(z*x))*x)*((z*(x*z))*(((x*x)$$
$$*\,(z*x))*(y*((((((z*y)*x)*(((y*(x*y))*((z*((x*x)*x))$$
$$*\,(((y*z)*x)*(y*y))))*((y*z)*x)))*((x*z)*(z*x)))$$
$$*\,(y*(x*x)))*(z*(x*y)))))))*((x*z)*x))*((z*x)*(y*y))))$$
$$*\,(y*z))*(x*(x*x)))*((y*y)*x))*(y*((x*x)*x))))*((((y*z)$$
$$*\,(x*x))*(((z*x)*y)*(z*x)))*((((x*((x*x)*y))*(y*(y*y)))$$
$$*\,((y*x)*(y*x)))*(((x*y)*y)*(((y*y)*y)*(((((y*x)*(y*y))$$
$$*\,(z*(x*y)))*((((((((y*(x*x))*(((y*x)*x)*x))*((z*x)*x))$$
$$*\,(((x*(y*x))*(((x*((x*x)*x))*((((z*x)*(x*x))$$
$$*\,(((x*(z*x))*((y*(z*y))*((((((x*z)*y)*(((x*x)*(y*z))$$
$$*\,((((x*z)*(z*x))*(((z*y)*(x*x))*(((x*x)*(z*y))$$
$$*\,(y*(z*x)))))*(z*(y*z)))))*(y*(x*z)))*((x*x)*(y*z)))$$
$$*\,((y*y)*x))))*((y*x)*(z*y))))*(x*(x*y))))*(y*(x*z))))$$
$$*\,(((x*x)*y)*x)))*(y*(x*z)))*((((((x*y)*(x*z))*((((z*x)$$
$$*\,(x*x))*((y*(y*z))*((x*x)*(y*z))))*(x*(x*(x*x)))))$$
$$*\,(x*(z*y)))*((x*y)*(y*x)))*((x*((x*x)*x))*(((y*y)$$
$$*\,(y*x))*((((((y*(x*y))*(((x*x)*y)*x))*(y*(x*z)))$$
$$*\,(z*(z*y)))*((x*x)*y))*(((((((y*x)*y)*((((y*x)*x)$$
$$*\,((z*((x*x)*x))*((z*(x*(x*x)))*(((x*y)*(z*x))$$
$$*\,(((x*((x*x)*x))*((((((z*(x*(x*x)))*((x*(x*x))$$
$$*\,(((y*x)*(x*x))*(((x*((x*x)*y))*((((z*x)*(y*x))$$
$$*\,(((((y*y)*y)*((z*x)*(x*y)))*(y*(y*z)))$$
$$*\,(x*(x*z))))*((z*y)*x)))*((z*y)*x)))))$$
$$*\,((z*z)*x))*((x*y)*(y*y)))*(y*((x*x)*x))))$$
$$*\,((z*z)*x))))))*(y*(y*x))))*z)*(y*(y*z)))$$
$$*\,((((x*x)*y)*y)*(y*((z*y)*z))))*((x*(x*y))$$
$$*\,(((x*y)*z)*(((y*y)*z)*(((y*z)*(x*x))*((y*(y*y))$$
$$*\,((y*((y*y)*z))*(z*(y*z))))))))))))))*((x*x)*((((y*x)$$

$$
\begin{aligned}
&*(z*y))*(((x*x)*y)*x))*((z*(z*y))*(((z*x)*z)\\
&*((((((x*y)*(y*y))*(((x*x)*(z*x))*((x*((x*x)*x))\\
&*(((((x*x)*(z*y))*(z*(z*x)))*(x*(x*z)))*((z*y)\\
&*(x*x))))))*((y*x)*y))*((x*x)*z))*((y*z)*y)))))))\\
&*((z*(z*x))*((y*(y*y))*(((x*(x*x))*y)*((z*(y*y))\\
&*(((x*x)*(z*y))*((((z*x)*(y*y))*(((y*x)*(x*x))\\
&*(x*((x*x)*y))))*((y*y)*z)))))))))*((((x*y)*z)\\
&*((z*((((y*x)*(z*x))*(((x*x)*y)*y))*(z*(y*z))))\\
&*((y*y)*z)))*(((x*x)*x)*x)))))))*(((x*z)*(y*x))\\
&*((x*(y*x))*(((x*y)*(y*y))*((z*(x*x))*((((((x*(z*x))\\
&*(((y*x)*(x*x))*(((((y*(x*z))*((((y*x)*x)*z)\\
&*(((((z*y)*(y*x))*(((x*(y*x))*((((x*(x*x))*z)\\
&*(((y*((x*x)*x))*(((x*x)*(x*z))*(((y*y)*(z*x))\\
&*(((y*y)*(x*x))*(((((((y*z)*(x*x))*(((((x*x)*(x*x))\\
&*((((x*(x*x))*x)*((x*(y*z))*((((((((((x*x)*((z*(y*x))\\
&*((x*y)*z)))*((x*x)*y))*(y*(z*x)))*(z*y))\\
&*((y*y)*(x*y)))*((x*z)*(x*y)))*(y*z))*(x*y))\\
&*(x*(y*y)))*y)))*(z*(y*z))))*(y*y))*((x*x)*x)))\\
&*((z*x)*(z*x)))*((x*y)*(x*x)))*(y*(x*x)))\\
&*(z*(z*y)))))))*(x*(x*y))))*(y*(y*z))))*(((x*x)*x)*z)))\\
&*((y*y)*x))*((((x*x)*x)*y)*(((((x*y)*z)*((x*x)*(z*x)))\\
&*(((y*x)*x)*z))*((x*x)*y))))))*((x*z)*(z*x)))\\
&*(x*(x*z)))*(z*z))))*((x*(x*x))*z))*((x*z)*x))\\
&*((x*(x*x))*z))*((y*y)*(x*y)))))))*((x*(x*(x*x)))\\
&*((((y*(y*y))*((((y*x)*x)*y)*(((y*(y*y))*(((y*x)*x)\\
&*((((y*x)*x)*y)*((((y*x)*(x*y))*((z*x)*(y*x)))\\
&*(z*(x*x))))))*((x*x)*(y*x)))))*((x*z)*(x*x)))\\
&*((y*x)*(x*y)))))*(((z*(x*y))*((((x*x)*(z*x))
\end{aligned}
$$

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

$$
\begin{aligned}
&*(x*z)))))*((((((y*y)*(y*x))*((((z*(y*y))*((z*x)*y))\\
&*((y*(x*x))*(((x*x)*(z*x))*(((y*(y*y))*((x*x)*(z*x)))\\
&*((z*(x*x))*((z*z)*((((((z*(y*x))*((((z*((x*x)*x))\\
&*((((z*((x*x)*x))*(z*((z*(x*x))*((((x*x)*x)*z)\\
&*(((y*x)*(x*x))*(((x*(x*x))*z)*(((x*(x*x))*z)\\
&*(y*z))))))))*((y*x)*(z*x)))*((z*x)*z)))*(((x*x)*x)*y))\\
&*((x*x)*y)))*(x*(x*(x*x))))*(z*(z*x)))\\
&*(z*(z*x)))*((x*x)*(z*x)))))))))*((x*y)*z)))\\
&*(((x*x)*y)*x))*(((x*x)*y)*x))*(z*(x*(x*x)))))\\
&*((x*y)*(x*z))))*((x*y)*((y*z)*y)))*y))*(((((x*(y*x))\\
&*(((y*y)*(y*x))*((z*x)*((x*((x*x)*y))*(((y*x)*x)*x)))))\\
&*(z*(x*(x*x))))*((x*x)*(y*z)))*((x*y)*z)))))\\
&*(x*(x*z))))*((x*x)*(x*z)))*(z*(z*x)))*((x*x)\\
&*(z*x)))*((x*(x*x))*z))*((x*(x*x))*z)))*(((z*x)*(x*z))\\
&*((((x*y)*(x*x))*(((y*(y*y))*((y*x)*y))\\
&*((x*x)*(x*x))))*(((((y*y)*(y*x))*(((((z*(y*y))\\
&*((((y*x)*x)*x)*((y*z)*x)))*(x*z))*((y*z)*y))*(x*x)))\\
&*(((x*x)*x)*y))*(((z*y)*y)*((y*((x*x)*x))\\
&*(((((x*y)*(y*y))*((((((y*(x*z))*(((y*x)*x)*z))\\
&*((y*z)*z))*(x*z))*(x*(x*y)))*(y*(x*y))))*(x*(y*z)))\\
&*((((((x*x)*y)*(((z*z)*x)*(y*z)))*(x*(x*x)))*((((y*x)\\
&*(x*x))*(y*y))*(((((y*(x*x))*(y*y))*(((y*(x*y))\\
&*(((((z*y)*(y*x))*((x*(z*y))*(z*(x*x))))*((x*x)*(y*z)))\\
&*((((y*x)*(z*x))*((((((((x*(x*x))*x)*((y*x)*(x*y)))\\
&*(y*(x*(x*x))))*((z*x)*y))*((((y*y)*(x*y))*((x*z)\\
&*(x*y)))*((((((x*y)*(y*y))*(((x*x)*(y*y))*(((((y*y)*x)\\
&*((y*(z*x))*((((((((y*z)*(x*x))*((x*y)*(z*x)))\\
&*((((((y*(y*x))*(x*(x*x)))*(((x*y)*(y*x))*((y*z)
\end{aligned}
$$

$$
\begin{aligned}
&* (x * x))))) * (((y * y) * (y * y)) * ((y * (z * x)) * ((((y * (y * y))\\
&* (((x * z) * (x * x)) * (((y * x) * (x * x)) * (x * ((x * x) * (z * y))))))\\
&* ((z * (y * z)) * ((z * x) * (y * x)))) * x)))) * ((x * y) * z)) * (z * y))\\
&* (((y * x) * x) * z))) * (((x * x) * x) * z)) * ((z * y) * x))\\
&* (y * ((x * x) * x))))) * ((z * x) * z)) * (y * (y * x)))))\\
&* ((x * (x * x)) * y)) * (x * (y * x))) * x))) * (((x * x) * (x * x))\\
&* (x * z))) * (y * z)) * ((x * y) * (x * x)))) * ((z * x) * z)))) * x)) * y)\\
&* ((x * x) * x)))) * (((x * x) * y) * (((((z * (x * z)) * ((y * x) * (y * x))) * y)\\
&* (z * y)) * ((x * y) * (x * y))))))))))).
\end{aligned}
$$

This example shows that the DDA is indeed capable of returning powerful results.

In Sec. 1, we will review the two prior methods of solving the TGP described above and justify our statements about their limitations. The DDA itself is described in detail in Sec. 2, where it is proven that it terminates if and only if it has found a correct solution. In Sec. 3, we give detailed illustrations of one application of the DDA in several different forms. The remainder of the paper is devoted to answering two questions.

(1) For which finite groupoids will the DDA efficiently solve the TGP?
(2) Are these groupoids rare or common?

In Sec. 4, we will describe three properties of a finite groupoid that we would expect to help prevent the DDA from failing, and instead lead it to efficiently yield the required terms. In Sec. 5, we will present data from test runs of the DDA on randomly generated groupoids. These data will show a strong correlation between satisfaction of the three properties from Sec. 4 and efficient production of human scale terms in human scale time by the DDA. In Sec. 6, we will give further test data showing how the DDA can fail if any one of our three conditions fails, but can also succeed without these conditions when the search space of term operations is too small. We will also test the DDA on several groupoids with search spaces much smaller and much larger than the controls. Each of the sample groupoids used will be checked for satisfaction of our three conditions, and the results will add to prior evidence from [2] that almost all finite groupoids meet these conditions. In Sec. 7, we will state a number of specific conjectures and open questions based on the experimental results of Secs. 5 and 6.

1. Prior Methods for Term Generation

As a means of putting this work in perspective, we examine here prior solutions to the TGP to provide a baseline as to what has otherwise been achieved. We can

often use our understanding of familiar algebras to construct specific terms. For example, our understanding of finite fields and the ternary discriminator operation allow us to write down the term

$$t(x, y, z) := (x - y)^{p^k - 1}(x - z) + z$$

defining the discriminator on the field $\mathbf{GF}[p^k]$.

In contrast, we are unlikely to have any similar algebraic understanding of randomly generated finite algebras like the groupoids presented in this paper. In those cases, prior methods of term generation reduce to the two discussed in the introduction: the Primality Test of [3] and random search. In this section, we will substantiate our claims that the former generates unacceptably long terms while the later takes an unacceptably long time.

At the core of the Primality Test of Clark *et al.* [3] is an explicit construction of a term representing an arbitrary operation $g : G^k \to G$ on a primal algebra $\mathbf{G}$. To state this theorem, let $\mathbf{G}$ be a finite nontrivial algebra with $G = \{0, 1, \ldots, n-1\}$. The identity operation $\mathrm{id}_G \colon G \to G$ on G is defined as $\mathrm{id}_G(a) = a$ for all $a \in G$. If $a \in G$, then we use $\underline{a}^{\mathbf{G}} \colon G^k \to G$ to denote the constant k-ary operation with value a.

Theorem 4 ([3]). *Let* $\mathbf{G}$ *be a finite algebra with* $G = \{0, 1, \ldots, n-1\}$ *for some* $n \geq 2$. *Then* $\mathbf{G}$ *is primal if and only if*

(i) *there is a binary term* $\wedge$ *such that* $\wedge^{\mathbf{G}}$ *agrees with the usual meet operation on* $\{0, 1\}$,
(ii) *for each* $i < n$ *there is a unary term* χ_i *such that* $\chi_i^{\mathbf{G}}$ *is the characteristic function of* $\{i\}$, *and*
(iii) *there is an* n*-ary term* p *with* $\mathrm{id}_G = p(\chi_0, \chi_1, \ldots, \chi_{n-1})^{\mathbf{G}}$.

Proof. Clearly primality implies (i)–(iii). Conversely assume (i)–(iii) hold. Let $g \colon G^k \to G$, be a k-ary operation for some $k \geq 1$. Define the term operation $\vee^{\mathbf{G}} \colon G^2 \to G$ by $x \vee y := \chi_0(\chi_0(x) \wedge \chi_0(y))$. Then $\vee^{\mathbf{G}}$ agrees with the usual join operation on $\{0, 1\}$. Also, $\underline{0}^{\mathbf{G}}$ is a k-ary term operation with $\underline{0}(x_0, x_1, \ldots, x_{k-1}) := \chi_0(x_0) \wedge \chi_1(x_0)$. For each $j \in G$ with $g^{-1}(j) \neq \varnothing$, define the k-ary term ψ_j by

$$\psi_j(x_0, x_1, \ldots, x_{k-1}) := \bigvee \left\{ \bigwedge_{i<k} \chi_{a_i}(x_i) \,|\, g(a_0, a_1, \ldots, a_{k-1}) = j \right\}, \tag{1}$$

where we understand each $\wedge$ and $\vee$ to be left associated. If $g^{-1}(j) = \varnothing$, then define $\psi_j := \underline{0}$. Then $\psi_j^{\mathbf{G}}$ is the characteristic function of the subset $g^{-1}(j)$ of G^k. It is now straightforward to calculate (as in [3]) that

$$g = p(\psi_0, \psi_1, \ldots, \psi_{n-1})^{\mathbf{G}}, \tag{2}$$

whence g is a term operation of $\mathbf{G}$. □

For example, the algebra $\mathbf{A}_1$ in Fig. 2 is still small enough to compute the required terms by hand. One possible outcome for $\mathbf{A}_1$ is

$$p(x,y,z) := x*(x*y),$$
$$x \wedge y := (x^2)^2 * ((x*y)*y^2),$$
$$\chi_0(x) := x*(x^2)^2, \quad \chi_1(x) := x^2*x, \quad \chi_2(x) := (x^2)^2*x^2,$$

again confirming that $\mathbf{A}_1$ is primal. The authors of [3] reported that UACalc found these terms for sample primal groupoids up to size seven in less than a minute, but memory capacity failed at size eight.

For a term $t = t(\vec{x})$ we define $L(t)$, the **length** of t, as the total number of variable occurrences in t. For example, $L(x^2 * y) = 3$. Our goal is to calculate a lower bound for the length of the term $p(\psi_0, \psi_1, \ldots, \psi_{n-1})$ representing g in the proof of Theorem 4 from the lengths of these $n+2$ primality terms satisfying the conditions of Theorem 4. If $+(x,y) = x+y$ is a groupoid term with binary operation symbol $*$ and variables x and y, we define ℓ_+ to be the number of occurrences of the *left* variable x in that term. For a k-ary operation $g : A^k \to A$, we write $||g^{-1}(j)||$ for the size of $g^{-1}(j)$ and define

$$T_g := \mathrm{Max}\{\|g^{-1}(j)\| \,|\, x_j \text{ occurs in } p(x_0, x_1, \ldots, x_{n-1})\}.$$

Theorem 5. *Let* $\mathbf{G} = \langle\{0,1,\ldots,n-1\},*\rangle$ *be a primal algebra with primality terms* $\wedge, \chi_0, \ldots, \chi_{n-1}, p$. *Let* $\ell := \ell_\wedge$, *let* $d := L(\chi_0)$, *let* $h := \mathrm{Min}\{L(\chi_i) \,|\, i < n\}$ *and let* $g : G^k \to G$ *be a* k*-ary operation for some* $k \geq 1$. *Then*

$$L(p(\psi_0, \psi_1, \ldots, \psi_{n-1})) \geq (d^2\ell)^{T_g - 1}(\ell^{k-1}h).$$

Proof. Let $r \geq 1$ and let $r_0, r_1, r_2, \ldots$ be arbitrary terms all of length greater than r. If $x+y$ is a binary term, then $L(r_0 + r_1) \geq \ell_+ r$. Extending this observation by induction, we can bound the length of the left associated sum as

$$L((\cdots((r_0 + r_1) + r_2) + \cdots) + r_{q-1}) \geq \ell_+^{q-1} r.$$

For $\vec{a} = (a_0, a_1, \ldots, a_{k-l}) \in G^k$, we apply (3) with $+ = \wedge$, with $q = k$ and with $r_i = \chi_{a_i}(x_i)$ to obtain

$$L\left(\bigwedge_{i<k} \chi_{a_i}(x_i)\right) \geq \ell^{k-1}h.$$

Defining $x \vee y := \chi_0(\chi_0(x) \wedge \chi_0(y))$ as in the proof of Theorem 4, we have $\ell_\vee = d^2\ell$. Now let $+ = \vee$, let $q = |g^{-1}(j)|$ and let $r_{\vec{a}} = \bigwedge_{i<k} \chi_{a_i}(x_i)$ for $\vec{a} \in g^{-1}(j)$. From (2) we have

$$\psi_j(x_0, x_1, \ldots, x_{k-1}) := \bigvee\{r_{\vec{a}} \,|\, \vec{a} \in g^{-1}(j)\}.$$

Taking $r = \ell^{k-1}h$ from above, we obtain

$$L(\psi_j(x_0, x_1, \ldots, x_{k-1})) \geq \ell_\vee^{q-1} r = (d^2\ell)^{|g^{-1}(j)|-1}(\ell^{k-1}h)$$

by again applying (3). The result now immediately follows. □

When applying Theorem 5, we must bear in mind that the outcome is very sensitive to the particular primality terms we use for $\mathbf{G}$. For the primal groupoid $\mathbf{A}_1$, we exhibited a term representing the ternary discriminator of length 40 obtained by EC in [13]. Applying Theorem 5 with the primality terms given above, we have $d = 5$, $\ell = 5$, $T_d = 9$, $k = 3$ and $h = 3$. Consequently the discriminator term produced by the Primality Test construction from these primality terms would have length at least $(5^2 \cdot 5)^{9-1}(5^{3-1} \cdot 3) > 10^{18}$, a number considerably larger than 40.

Random search of k-ary terms provides a different approach to solve the TGP. As k-ary terms are generated, we check each one to see if it represents the target operation. The exact method depends on the particular algorithm used to generate terms. For example, all terms could be generated systematically in order of size. It appears that most term generation algorithms offer no practical means to calculate the expected number of trials required for a particular term operation to appear since we have no information about how term operations are distributed among terms. As we only need a rough estimate of the expected number of trials, we will make a simplifying assumption that will allow us to compute it.

By a **uniform random search** for a groupoid $\mathbf{G}$, we mean a random search in which candidate terms are chosen so that all k-ary term operations have equal probability of appearing with each choice. Uniformity guarantees that the probability of generating all k-ary term operations approaches one. It also provides a well-known method to calculate the expected number of trials, a method which does not appear to be feasible to apply without uniformity.

Theorem 6. *Let* $\mathbf{G}$ *be a finite groupoid with* N *distinct* k*-ary term operations. Let* $t_0, t_1, t_2, \ldots$ *be a countable sequence of random variables whose values are* k*-ary terms generated by a uniform random search for a term representing a* k*-ary term operation* g *of* $\mathbf{G}$*. Let* X *be the random variable whose value is the least* $i \geq 0$ *such that* $t_i^{\mathbf{G}} = g$*. Then the expected value* $\mathrm{Exp}(X)$ *of* X *is* N*.*

Proof. Since $t_0^{\mathbf{G}}, t_1^{\mathbf{G}}, \ldots$ are uniformly distributed over the set of k-ary term operations of $\mathbf{G}$, for $i \geq 0$ we have $\Pr(t_i^{\mathbf{G}} = g) = \frac{1}{N}$ and $\Pr(t_i^{\mathbf{G}} \neq g) = 1 - \frac{1}{N}$. Because $t_0, t_1, \ldots$ are also independent, we compute $p_i := \Pr(X = i) = (1 - \frac{1}{N})^{i-1}\frac{1}{N}$ and $P_i := \Pr(X \geq i) = (1 - \frac{1}{N})^{i-1}$. This gives us

$$\mathrm{Exp}(X) = \sum_{i=1}^{\infty} i p_i = \sum_{\ell=1}^{\infty}\sum_{i=\ell}^{\infty} p_i = \sum_{\ell=1}^{\infty} P_\ell = \sum_{\ell=1}^{\infty}\left(1 - \frac{1}{N}\right)^{\ell-1}$$

$$= \frac{1}{1 - (1 - \frac{1}{N})} = N. \qquad \square$$

As an application of Theorem 6, consider the problem of generating a discriminator term in the most challenging case in which $\mathbf{G}$ is primal and therefore has a search space consisting of all $N = n^{n^3}$ of the ternary operations on G. Assume that we use a fast computer with a 3 GHz processor and an efficient algorithm that will generate and test a term t_i in 1,000 clock cycles. We could then search

Primal Groupoid Size (n)	2	3	4	5	6	7
Ternary Operations ($N = n^{n^3}$)	10^2	10^{13}	10^{39}	10^{87}	10^{168}	10^{290}
Expected Time in Years ($10^{-14}N$)	10^{-12}	10^{-1}	10^{25}	10^{73}	10^{154}	10^{276}

Fig. 3. Expected time to target by uniform random search at 3 GHz.

3×10^6 terms per second, or about 10^{14} terms per year of continuous computation. Theorem 6 then gives us an estimate of the expected time to find a discriminator term as $10^{-14}N$ years. In Fig. 3, we see how rapidly this expected time to success would grow with the size of the groupoid. While these time estimates derived from Theorem 6 are rough, Fig. 3 shows that an over estimate of as many as 10 orders of magnitude would not affect our conclusions about computational feasibility in any case except $n = 3$.

2. The Deep Drilling Algorithm

Let $\mathbf{G} := \langle G, * \rangle$ be a finite groupoid with $G = \{0, 1, \ldots, n-1\}$ and let k be a positive integer. By a **term** we mean a groupoid term in the variables $\vec{x} := (x_0, x_1, \ldots, x_{k-1})$. The **depth** of a subterm occurrence in a term refers to the depth of its node in the term tree, with the depth of the root being one. The **height** of a term is the greatest depth of its variable occurrences. For $H \geq 1$, we denote by $\mathcal{T}_H$ the set of all terms of height at most H. Notice that the length of a term is the number of leaves in its term tree. We will use language based on the view that

a tree has its root at the bottom and its leaves at the top.

If $u(\vec{x})$ is a subterm of $t(\vec{x})$ and $v(\vec{x})$ is a subterm of $u(\vec{x})$, then we will say that $v(\vec{x})$ is a "deeper" subterm of $t(\vec{x})$ than $u(\vec{x})$. Thus, "deeper" and "higher" both mean further from the root.

If A is a k-ary array over a finite groupoid $\mathbf{G}$ and $u(\vec{x})$ and $v(\vec{x})$ are terms, we define arrays $LA, [u(\vec{x}), A], RA$ and $[A, v(\vec{x})]$ by

$$\begin{aligned} LA(\vec{d}) &:= \{a \in G \mid ab \in A(\vec{d}) \text{ for some } b \in G\} \\ [u(\vec{x}), A](\vec{d}) &:= \{b \in G \mid u(\vec{d})b \in A(\vec{d})\} \\ RA(\vec{d}) &:= \{a \in G \mid ba \in A(\vec{d}) \text{ for some } b \in G\} \\ [A, v(\vec{x})](\vec{d}) &:= \{b \in G \mid bv(\vec{d}) \in A(\vec{d})\} \end{aligned}$$

for $\vec{d} \in G^k$. Note that if $u(\vec{x})$ is a solution to LA, then $[u(\vec{x}), A]$ has value at each $\vec{d} \in G^k$ consisting of all values b that witness $u(\vec{d}) \in LA(\vec{d})$. Similarly, if $v(\vec{x})$ is a solution to RA, then $[A, v(\vec{x})]$ consists of all b witnessing the fact that $v(\vec{d}) \in RA(\vec{d})$. The following easily verified lemma lies at the core of the DDA.

Lemma 7. *Let* $\mathbf{G}$ *be a finite groupoid and let* A *be a* k*-ary array over* G*. If either*

(i) $u(\vec{x})$ *is a solution to* LA *and* $v(\vec{x})$ *is a solution to* $[u(\vec{x}), A]$ *or*
(ii) $v(\vec{x})$ *is a solution to* RA *and* $u(\vec{x})$ *is a solution to* $[A, v(\vec{x})]$,

then, $u(\vec{x})v(\vec{x})$ *is a solution to* A*.*

Thus, we can solve A if we can either find a solution $u(\vec{x})$ to LA and then solve $[u(\vec{x}), A]$, or find a solution $v(\vec{x})$ to RA and then solve $[A, v(\vec{x})]$. We can now describe the algorithm.

The Deep Drilling Algorithm

Input: a finite groupoid $\mathbf{G}$, a k-ary target array $T : G^k \to 2^G$ and an integer $m > 0$

Output: a k-ary term that is a solution to T

Repeating steps (1)–(7) below builds a binary term tree for a solution to T by associating each node ν with a k-ary array X and then solving X to find the subterm of the solution to T that will be assigned to ν. Begin by constructing the root node τ, associating it with T, and taking T as X and τ as ν in (1). Each time (6) is completed, a new node is assigned a subterm. Then (7) tells when to halt and how to get to the next node and associate it with an array if it is not yet time to halt.

(1) Compute the component sets of X and see if it has a solution in $\mathfrak{T}_m$. If so, define $w(\vec{x})$ to be a randomly chosen solution from $\mathfrak{T}_m$ and go to (6).
(2) If no term of $\mathfrak{T}_m$ is a solution to X, add a left child node λ and right child node ρ above ν. Randomly choose λ or ρ.
(3) If λ is chosen:
    (a) Associate λ with the array LX. Recursively apply the DDA to obtain a solution $u(\vec{x})$ to LX, and assign $u(\vec{x})$ to λ.
    (b) Associate ρ with the array $[u(\vec{x}), X]$. Recursively apply the DDA to obtain a solution $v(\vec{x})$ to $[u(\vec{x}), X]$, and assign $v(\vec{x})$ to ρ.
(4) If ρ is chosen:
    (a) Associate ρ with the array RX. Recursively apply the DDA to obtain a solution $v(\vec{x})$ to RX, and assign $v(\vec{x})$ to ρ.
    (b) Associate λ with the array $[X, v(\vec{x})]$. Recursively apply the DDA to obtain a solution $u(\vec{x})$ to $[X, v(\vec{x})]$, and assign $u(\vec{x})$ to λ.
(5) Define $w(\vec{x})$ to be the product of the left and right child nodes of ν.
(6) Assign $w(\vec{x})$ to ν. If $\nu = \tau$, output $w(\vec{x})$ and halt.
(7) If $\nu \neq \tau$, let Y be the array associated with the parent node π of ν and let σ be the sibling node of ν.
    (a) If a subterm $s(\vec{x})$ has been assigned to σ, go to (5) with π as ν.
    (b) If no subterm has been assigned to σ, associate it with $[Y, w(\vec{x})]$ if it is a left node and with $[w(\vec{x}), Y]$ if it is a right node. Recursively go to (1) with that array as X and σ as ν.

In general, there is no guarantee that the recursive calls of the DDA will ever terminate. This will only happen when a term is assigned to the root node that is then returned as an output in (6). What we do know is that, if the DDA does terminate, then it will output a solution.

Theorem 8. *Let T be an array over a finite groupoid $\mathbf{G}$ and assume that the DDA with input T terminates. Then the term it outputs will be a solution to T.*

Proof. The proof is by induction on the height of the output term. If it has height one, then there must have been no recursions and the output is a variable that is a solution to T, the array associated with the root node.

Assume the output term $t(\vec{x})$ has height $H+1$ with $H \geq 1$. If T has a solution in $\mathfrak{T}_m$, then one such solution was the output. Otherwise, the DDA branched at the root and moved up to the left or right. Assume it went to the left. Then $t(\vec{x}) = u(\vec{x})v(\vec{x})$ where $u(\vec{x})$ is the output of the DDA for the left child node of the root and $v(\vec{x})$ is the output of the DDA for the right child node. By induction, since $u(\vec{x})$ and $v(\vec{x})$ have maximum height H, we conclude that they are solutions to the arrays LT and $[u(\vec{x}), T]$. By Lemma 7, the term $t(\vec{x})$ is a solution to T. □

The DDA could be simplified by eliminating the random choices, for example, by always choosing the first solution in $\mathfrak{T}_m$ and always going up to the left. But this can lead to failure by associating some node with the same array it associated with a lower node. The DDA will then be caught in a loop, drilling ever deeper while endlessly repeating the same sequence of steps. In Sec. 5, we will see many runs of the DDA that terminated within 20 s. Prior to introducing random choices, quite a number of those same runs failed to terminate after 10 h.

3. A Sample Application

Normal use of the DDA requires a computer implementation. However, in small enough examples, it is practical to implement it by hand. As an illustration of this algorithm, we give here such an example. Consider the primal groupoid $\mathbf{A}_1$ in Fig. 2. Being primal, it must have a term representing the operation + of addition modulo three. To find such a term, we apply the DDA with the input array $\langle + \rangle$ whose value at (a, b) is $\{a+b\}$. Because this is a small example, we choose the small value of $m = 2$, using $\mathfrak{T}_2$ as the search terms.

Example 9. An output of the DDA representing addition modulo three over the primal groupoid $\mathbf{A}_1$ is the term

$$t(x, y) := ((x(yx))(y(xy)))((y^2y^2)x).$$

In Table 1, we have recorded the steps taken by the DDA to produce this term. The rows of the table divide the DDA run into discrete iterations. By an **iteration** of the DDA, we mean a full sequence of steps it could execute while

Table 1. Computation of a term representing $+ \bmod 3$ on $\mathbf{A}_1$ with $m = 2$.

Array	00	01	02	10	11	12	20	21	22	Solution	#
$\langle + \rangle$	0	1	2	1	2	0	2	0	1	(#7)(#12)	#13
	0	1	2	1	2	0	2	0	1	$\hookleftarrow$	
$L\langle + \rangle$	12	$*$	0	$*$	0	12	0	12	$*$	$(x(yx))(y(xy))$	#7
	1	1	0	1	0	1	0	1	2	$\hookleftarrow$	
$LL\langle + \rangle$	$*$	$*$	12	$*$	12	$*$	12	$*$	$*$	$x(yx)$	#3
	2	1	2	0	1	1	1	0	0	$\hookleftarrow$	
$RLL\langle + \rangle$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	yx	#1
	2	1	0	1	0	0	2	0	1	$\hookleftarrow$	
A	$*$	$*$	01	$*$	01	$*$	2	$*$	$*$	x	#2
B	2	$*$	01	$*$	12	0	12	$*$	$*$	$y(xy)$	#6
	2	0	1	1	1	0	2	1	0	$\hookleftarrow$	
LB	0	$*$	$*$	$*$	$*$	12	$*$	$*$	$*$	y	#4
$[y, B]$	02	$*$	$*$	$*$	0	01	$*$	$*$	$*$	xy	#5
	2	1	2	1	0	0	0	0	1	$\hookleftarrow$	
C	12	0	02	0	02	12	02	12	2	$(y^2y^2)x$	#12
	1	0	2	0	0	1	0	1	2	$\hookleftarrow$	
RC	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	02	x	#8
$[C, x]$	01	2	02	12	12	0	01	02	0	y^2y^2	#11
	1	2	0	1	2	0	1	2	0	$\hookleftarrow$	
$L[C, x]$	$*$	0	$*$	$*$	$*$	12	$*$	$*$	12	y^2	#9
	2	0	1	2	0	1	2	0	1	$\hookleftarrow$	
$[y^2, [C, x]]$	$*$	02	12	2	$*$	12	$*$	02	12	y^2	#10

$A = [LL\langle + \rangle, yx] \quad B = [x(yx), L\langle + \rangle] \quad C := [(x(yx))(y(xy)), \langle + \rangle] \quad * := 012$

working at a particular node before moving on to another node. The numbered columns of the table display the arrays on the left whose solutions will be subterms of the required term. The 11th column gives the solution that was eventually found to each array. The last column indicates the numerical order (#) in which these solutions arose. Following the arrays, we listed the term operation values ($\hookleftarrow$) if the terms that were not a single variable. The symbol $*$ represents 012, indicating that every term is a solution for that case. The table was generated as follows:

(1) We listed the values of $\langle + \rangle$ in the first row.
(2) We computed the arrays $L\langle + \rangle, LL\langle + \rangle$ and $RLL\langle + \rangle$, randomly choosing between L and R, until an array was found having a solution in $\mathfrak{T}_2$. In this case, every term was a solution to $RLL\langle + \rangle$, and we chose the solution yx (#1).
(3) Computing the values of $A = [LL\langle + \rangle, yx]$, we found that it has solution x and therefore, by Lemma 7, $x(yx)$ is a solution to $LL\langle + \rangle$.
(4) Computing values of $B = [x(yx), L\langle + \rangle]$, we saw that it has no solution in $\mathfrak{T}_2$.
(5) Going left, we saw that LB has the solution y in $\mathfrak{T}_2$. Computing the values of $[y, B]$, we found that it has solution xy. By Lemma 7, the term $y(xy)$ is a solution to B and $(x(yx))(y(xy))$ is a solution to $L\langle + \rangle$.

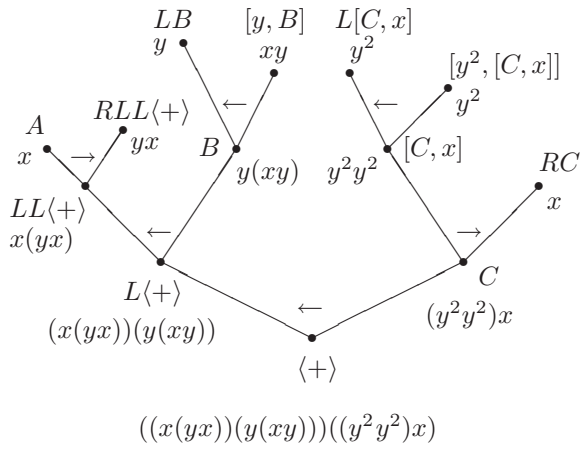


Fig. 4. Term tree for a term representing + on $\mathbf{A}_1$.

It remained to similarly solve the array $C = [(x(yx))(y(xy)), \langle+\rangle]$, obtaining the solution $(y^2y^2)x$. By Lemma 7, the product $((x(yx))(y(xy)))((y^2y^2)x)$ gave us a solution to $\langle+\rangle$.

In Fig. 4, we illustrate the resulting term tree, each node with its associated array and its eventually assigned subterm. Figure 5 gives a different illustration of this DDA run where its successive states are described by partial terms. Adding

$\Diamond$	$\langle+\rangle$
$\Diamond-$	$L\langle+\rangle$
$(\Diamond-)-$	$LL\langle+\rangle$
$((-\Diamond)-)-$	$RLL\langle+\rangle$
$((\Diamond(yx))-)-$	$A = [LL\langle+\rangle, yx]$
$((x(yx))\Diamond)-$	$B = [x(yx), L\langle+\rangle]$
$((x(yx))(\Diamond-))-$	LB
$((x(yx))(y\Diamond))-$	$[y, B]$
$((x(yx))(y(xy)))\Diamond$	$C = [(x(yx))(y(xy)), \langle+\rangle]$
$((x(yx))(y(xy)))(-\Diamond)$	RC
$((x(yx))(y(xy)))(\Diamond x)$	$[C, x]$
$((x(yx))(y(xy)))((-\Diamond)x)$	$L[C, x]$
$((x(yx))(y(xy)))(((\Diamond y^2)x)$	$[y^2, [C, x]]$
$((x(yx))(y(xy)))((y^2y^2)x)$	

Fig. 5. $\Diamond$: solution to array to right. $-$: solution to yet unspecified array.

a pair of child nodes to the term tree adds a pair of parentheses to the partial term. The lozenge ◊ indicates the node in the tree where the algorithm is currently looking for a solution to the array on the right. The dashes indicate nodes whose associated arrays are yet to be determined.

4. Termination

From Theorem 8, we know that the DDA is successful if and only if it terminates. In the remainder of this paper, we will examine the conditions under which it will and will not terminate. We will describe three properties of finite groupoids whose presence we believe might help to ensure that the DDA will terminate with a solution: no separating relations (NSRs), asymptotic completeness (AC) and idemprimality (IPr). We will prove a theorem which shows how IPr precludes an obvious obstacle to termination. Conditions NSR and AC arose in [2] as sufficient conditions to guarantee term continuity (TC), the property that the term to term operation function of a groupoid is continuous. In evolutionary algorithms for the TGP, TC can be thought of as meaning that small mutations are survivable.

4.1. *Solvable arrays*

The DDA proceeds by recursively constructing and solving many arrays. Once it reaches a node with a particular array assignment, it will not leave the branch growing from that node until if finds a solution to its array. Consequently a necessary condition for it to terminate is that every array it generates and assigns to a node does indeed have a solution.

It is here that we call on idemprimality. We say that a k-ary array A over a finite groupoid **G preserves idempotents** if $e \in A(\vec{e})$ for each idempotent $e \in G$ and the k-tuple $\vec{e}$ with constant value e.

Lemma 10. *Let* **G** *be an idemprimal groupoid and let A be a k-ary array over G. Then A has a solution if and only if it preserves idempotents and $A(\vec{d}) \neq \varnothing$ for all $\vec{d} \in G^k$.*

Proof. Let $t(\vec{x})$ be a solution to A. Then $t(\vec{d}) \in A(\vec{d})$ for all $\vec{d} \in G^k$, showing that $A(\vec{d}) \neq \varnothing$. If $e \in G$ is an idempotent, then $e = t(\vec{e}) \in A(\vec{e})$, showing that A preserves idempotents.

Now assume A preserves idempotents and that its component sets are non-empty. Let $g : G^k \to G$ be any operation for which $g(\vec{d}) \in A(\vec{d})$ for all $\vec{d} \in G^k$ and $g(\vec{d}) = e$ if $\vec{d} = \vec{e}$ for some idempotent e. Since **G** is idemprimal, g is a term operation that is a solution to A. □

Theorem 11. *Let* **G** *be an idemprimal groupoid (IPr). If T is a k-ary array that has a solution, then the array associated with* every *node of the term tree generated by the DDA with input T also has a solution.*

Proof. It is sufficient to show that if A is a k-ary array over G that has a solution which is not a variable, then LA has a solution and, for *every* solution $u'(\vec{x})$ to LA, the array $[u'(\vec{x}), A]$ also has a solution. (By virtually the same argument, RA has a solution and, for every solution $v'(\vec{d})$ to RA, the array $[A, v'(\vec{x})]$ has a solution.)

Let $u(\vec{x})v(\vec{x})$ be a solution to A that is not variable. Then $u(\vec{x})$ is a solution to LA. Now let $u'(\vec{x})$ be any solution to LA. To show that $[u'(\vec{x}), A]$ has a solution, we apply Lemma 10. Let $\vec{d} \in G^k$. Then $u'(\vec{d}) \in LA$ so there is a $b \in G$ such that $u'(\vec{d})b \in A(\vec{d})$. Thus $b \in [u'(\vec{x}), A](\vec{d}) \neq \varnothing$. If $e \in G$ is an idempotent, then $e = u'(\vec{e})e \in A(\vec{e})$ since A preserves idempotents. Thus $e \in [u'(\vec{x}), A](\vec{e})$, showing that $[u'(\vec{x}), A]$ preserves idempotents. By Lemma 10, we see that $[u'(\vec{x}), A]$ has a solution. □

It is natural to ask if Theorem 11 extends to semiprimal groupoids. Recall that a finite algebra is **semiprimal** if every operation that preserves its subalgebras is a term operation. We say that a k-ary array A over $\mathbf{G}$ **preserves subgroupoids** if, for each subgroupoid $\mathbf{H}$ of $\mathbf{G}$ and $\vec{d} \in H^k$, the set $A(\vec{d})$ contains an element of H. It is easy to see that Lemma 10 extends to semiprimal groupoids with the same proof. It is also easy to see that the proof of Theorem 11 does not in any straightforward way extend to semiprimal groupoids. Example 16 in Sec. 6 will show that the DDA, applied to an array over a semiprimal groupoid that has a solution, can generate an array that has no solution.

4.2. *Term continuity*

While the DDA is not an evolutionary algorithm, it was extracted from evolutionary algorithms. As we will demonstrate in the next two sections and the upcoming subsequent work [4], the conditions associated with success of the DDA are the same as those associated with success of the co-evolutionary algorithm of [4]. It is the thesis of [2] that is a fundamental property of a finite groupoid needed for any of these algorithms. To succeed in general, the groupoid satisfy the following condition.

Continuity Condition. For $k \geq 1$, a finite groupoid $\mathbf{G}$ is ***k-term continuous*** if, in a probabilistic sense, mutations deep in k-ary terms generally result in correspondingly small changes in their term operations.

This condition is defined precisely in [2], where it is interpreted to mean that the k-ary term to k-ary term operation map is in a sense "continuous" relative to appropriate metrics on the spaces of k-ary terms and k-ary term operations.

While the definition of "term continuity" given in [2] makes this notion precise, it does not offer any practical method to tell if a particular groupoid is or is not term continuous. The central result of [2] is to show that a finite groupoid is k-term continuous if it has two testable properties, one of which is necessary.

Continuity Theorem ([2]). *Let $k > 1$ and let $\mathbf{G}$ be a finite groupoid.*

(i) *If some subgroupoid of $\mathbf{G}$ has a separating relation, then $\mathbf{G}$ is not k-term continuous.*
(ii) *If no subgroupoid of $\mathbf{G}$ has a separating relation and $\mathbf{G}$ is asymptotically k-complete, then $\mathbf{G}$ is k-term continuous.*

In order to make this theorem understandable, we review the definitions of "separating relation" and "asymptotically k-complete" introduced in [2].

Let $\mathbf{G}$ be a finite groupoid. A non-empty, irreflexive, symmetric, binary relation $\sigma \subseteq G^2$ is a **separating relation on G** if, for all $a, b, c \in G$,

$$(a, b) \in \sigma \text{ implies } (ac, bc) \in \sigma \quad \text{and} \quad (ca, cb) \in \sigma.$$

We say that $\mathbf{G}$ has NSRs if there is no separating relation on any subgroupoid of $\mathbf{G}$. For example, if θ is a congruence on $\mathbf{G}$ for which $\mathbf{G}/\theta$ is a nontrivial quasi-group, then it is easy to see that $\sigma := G^2 \backslash \theta$ is a separating relation on $\mathbf{G}$. In particular, if $\mathbf{G}$ is a nontrivial quasi-group, then the $\neq$ relation is a separating relation on $\mathbf{G}$. A simple paper-and-pencil algorithm is given in [2] (the Separating Relations Test 9) that takes a finite groupoid $\mathbf{G}$ as input and returns either the unique maximal separating relation on $\mathbf{G}$ or the statement that $\mathbf{G}$ has NSR. For example, it is quick to check that **Pi** in Fig. 2 has NSR. In Secs. 5 and 6, we will add to prior evidence that almost all finite groupoids have NSR.

Fix a choice of a positive integer $k > 1$ representing the arity of the operations we will study. For each positive integer H, let $\mathcal{T}_H$ denote the set of k-ary terms of height at most H, where $\mathcal{T}_1 = \{x_0, x_1, \ldots, x_{k-1}\}$ is the set of k variables. If $\vec{d} \in G^k$, let $\mathrm{sg}(\vec{d})$ denote the **subgroupoid generated by** $\{d_0, d_1, \ldots, d_{k-1}\}$. For each $\vec{d} \in G^k$ and $a \in G$, let $\beta_{\vec{d},a}(H)$ be the probability that $t(\vec{d}) = a$ for $t(\vec{x}) \in \mathcal{T}_H$. We say that $\mathbf{G}$ is asymptotically complete (AC) at $\vec{d} \in G^k$ if, for each $a \in \mathrm{sg}(\vec{d})$, the sequence $\beta_{\vec{d},a}$ is eventually bounded away from zero. (Note that $\beta_{\vec{d},a}$ has constant value zero if and only if $a \notin \mathrm{sg}(\vec{d})$.) We say that $\mathbf{G}$ is **asymptotically k-complete** if $\mathbf{G}$ is AC at each $\vec{d} \in G^k$. We say that $\mathbf{G}$ is AC if it is asymptotically k-complete for each $k > 1$.

We refer to the sequences of probability distributions

$$\{(\beta_{\vec{d},0}, \beta_{\vec{d},1}, \ldots, \beta_{\vec{d},n-1}) \mid \vec{d} \in G^k\}$$

as the **probability sequences for G**. It is shown in [2] (Theorem 16) that probability sequences can be recursively generated on a spreadsheet. Our experience is that these spreadsheet runs always offer compelling evidence either that the groupoid is AC or that it is not. Very often we find that $\beta_{\vec{d},a}$ converges to a positive value for each $\vec{d} \in G^k$ and $a \in \mathrm{sg}(\vec{d})$. In this case $\mathbf{G}$ is clearly AC.

As a typical example, the groupoid **Pi** in Fig. 2 has exactly two proper subgroupoids, $\{1\}$ and $\{2\}$. Taking $k = 10$, the probability sequence for $\vec{1} := (1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$ has constant value $(0, 1, 0, 0, 0)$ and the probability sequence

Table 2. Asymptotic completeness of **Pi**.

H	$\beta_{\vec{d},0}$(H)	$\beta_{\vec{d},1}$(H)	$\beta_{\vec{d},2}$(H)	$\beta_{\vec{d},3}$(H)	$\beta_{\vec{d},4}$(H)
1	0.2	0.1	0.1	0.5	0.1
2	0.127273	0.127273	0.145455	0.254545	0.345455
4	0.098817	0.182526	0.088076	0.368477	0.262102
8	0.103371	0.195561	0.091358	0.365539	0.244171
16	0.103372	0.195275	0.091609	0.365434	0.244311
32	0.103372	0.195275	0.091609	0.365434	0.244310
64	0.103372	0.195275	0.091609	0.365434	0.244310
500	0.103372	0.195275	0.091609	0.365434	0.244310

for $\vec{2} := (2, 2, 2, 2, 2, 2, 2, 2, 2, 2)$ has constant value $(0, 0, 1, 0, 0)$. As one check for AC, we entered the values $\vec{d} = (3, 4, 3, 0, 2, 3, 3, 0, 1, 3)$. Since only x_3 and x_7 are assigned value 0, we have, for example, $\beta_{\vec{d},0}(1) = 0.2$.

Our spreadsheet computation shows all five $\beta_{\vec{d},a}$ sequences are stable to six decimal places from $H = 32$ to $H = 500$, as indicated by sample outputs in Table 2. Many other choices of $\vec{d}$ other than $\vec{1}$ or $\vec{2}$ indicate convergence to the same limit. From this, we tentatively conclude that **Pi** is also AC, and therefore is term continuous.

5. DDA Control Data

We will now report on tests of the DDA with randomly generated groupoids in order to assess its ability to solve the TGP for groupoids satisfying our conditions NSR, AC and IPr.

In order to test the DDA we focused our study on ternary operations on 5-element groupoids for several reasons. The search space of k-ary term operations on an n-element groupoid has size up to n^{n^k}, the size realized in the primal cases. If n and k are too small, the DDA can often find terms inadvertently, without effectively engaging the algorithm. For example, it is hard to find any 3-element groupoid at all on which it will fail to find a term for some ternary term operation. In this and the next section, we will exhibit examples in which small search space sizes lead to incongruous test results. In contrast, this problem seems to be rare for ternary operations on 5-element groupoids. With search spaces up to size $5^{5^3} > 10^{87}$, we find that 5-element groupoids are large enough to offer significant tests of the algorithm. But they are still small enough to allow us to record many test runs and easily recognize failures.

Our implementation of the DDA was written in the Clojure programming language (which compiles to Java bytecode) and was run on a MacBook Pro using the OS X 10.9.5 operating system with a 2.2 GHz Intel Core i7 processor. We found that $m = 4$ seemed to be the most efficient choice of the parameter m for our hardware, and we have used it for all of our runs. With $m = 5$, the searches of $\mathcal{T}_5$ took longer than the additional iterations of the DDA with $m = 4$. With $m = 3$ the additional iterations of the DDA took longer than the search of $\mathcal{T}_4$. We used

P$_1$

$*$	0	1	2	3	4
0	3	2	0	4	0
1	2	4	1	1	1
2	1	1	3	0	1
3	1	3	4	1	0
4	0	1	0	1	0

P$_2$

$*$	0	1	2	3	4
0	2	1	2	4	1
1	0	4	4	1	4
2	0	4	4	1	3
3	1	1	2	1	3
4	1	1	0	0	3

I$_1$

$*$	0	1	2	3	4
0	1	3	0	4	1
1	1	0	3	3	0
2	3	4	3	0	1
3	4	4	0	1	2
4	0	3	1	1	4

I$_2$

$*$	0	1	2	3	4
0	2	3	2	2	1
1	2	1	2	3	3
2	1	3	4	3	3
3	0	4	0	1	1
4	3	2	4	1	1

I$_3$

$*$	0	1	2	3	4
0	1	3	4	3	3
1	3	2	1	4	0
2	3	1	0	2	2
3	4	2	3	3	1
4	2	2	4	1	1

I$_4$

$*$	0	1	2	3	4
0	0	3	2	1	3
1	0	4	1	1	2
2	3	4	3	2	0
3	0	0	0	1	1
4	0	4	1	3	1

H$_1$

$*$	0	1	2	3	4
0	3	2	4	4	2
1	0	2	2	1	0
2	1	0	4	4	4
3	4	4	3	0	0
4	2	3	4	4	4

H$_2$

$*$	0	1	2	3	4
0	4	1	0	3	1
1	4	4	2	1	2
2	2	4	4	4	4
3	3	0	2	4	4
4	1	4	1	3	3

H$_3$

$*$	0	1	2	3	4
0	0	2	1	4	1
1	2	3	1	3	2
2	2	1	4	0	1
3	0	1	1	1	0
4	3	1	1	2	4

H$_4$

$*$	0	1	2	3	4
0	0	3	0	0	0
1	1	0	3	2	1
2	2	2	1	4	3
3	3	1	1	3	1
4	0	2	0	1	2

Fig. 6. Ten randomly generated control groupoids.

⟨www.random.org⟩ to create a baseline of randomly constructed control groupoids. A sequence of 250 random digits from $\{0, 1, 2, 3, 4\}$ generated tables for the 10 5-element controls shown in Fig. 6.

Our next step was to determine which of the conditions NSR, AC and IPr each of the ten controls met.

No Separating Relations. Applying the Separating Relations Test from [2] we easily saw that all 10 have NSR.

Asymptotic Completeness. To test for AC, we did spreadsheet runs of the probability sequences for each of the controls, displaying the results in Table 3. Based on our experience, it appears that the asymptotic behavior of these sequences is not sensitive to the value of $k \geq 2$, and depends only on the subgroupoid $\mathrm{sg}(\vec{d})$ generated by $\vec{d}$ (see Conjecture 20). We took $\vec{d} = (0, 1, 2, 3, 4)$, thereby starting at $H = 1$ with a uniform distribution generating the entire groupoid. Table 3 gives, for each groupoid except $\mathbf{P}_1$ and $\mathbf{I}_4$, the value of H at which the probability sequence

Table 3. AC stabilization for 10 controls.

	H: Stable to 6 Places	$\beta_{\vec{d},0}(H)$	$\beta_{\vec{d},1}(H)$	$\beta_{\vec{d},2}(H)$	$\beta_{\vec{d},1}(H)$	$\beta_{\vec{d},2}(H)$
$\mathbf{P}_1$	640	0.214617	0.421448	0.112999	0.092938	0.157998
	641	0.160072	0.352963	0.180900	0.097998	0.208067
$\mathbf{P}_2$	47	0.096406	0.407246	0.012507	0.160493	0.323348
$\mathbf{I}_1$	46	0.223009	0.237552	0.056348	0.196862	0.286230
$\mathbf{I}_2$	26	0.067649	0.327946	0.157361	0.300648	0.146396
$\mathbf{I}_3$	17	0.106159	0.231534	0.266755	0.244381	0.151170
$\mathbf{I}_4$	(65,015)	0.993588	0.003196	0.000010	0.003196	0.000010
$\mathbf{H}_1$	17	0.000000	0.000000	0.000000	0.000000	1.000000
$\mathbf{H}_2$	42	0.044221	0.121108	0.097690	0.329468	0.407513
$\mathbf{H}_3$	211	0.090064	0.424037	0.121105	0.319300	0.045494
$\mathbf{H}_4$	25	0.311505	0.227259	0.154695	0.265474	0.041067

becomes stable to six decimal places, followed by that stable value $\beta_{\vec{d}}(H)$. In these eight cases, we take this as an indication that the sequence is converging to a point close to that $\beta_{\vec{d}}(H)$ for the listed value of H.

In all cases except $\mathbf{P}_1, \mathbf{I}_4$ and $\mathbf{H}_3$, the results are similar to those shown in Table 2 for **Pi**, with each sequence stabilizing to six decimal places at some value of H below 50. The sequences for $\mathbf{H}_3$ appear to similarly converge, but they take a bit longer. The groupoid $\mathbf{P}_1$ is an example of AC that does not converge. Rather, the even terms converge to one limit and the odd terms converge to another, with all limits being positive. The groupoid $\mathbf{I}_4$ is particularly unusual. It apparently converges monotonically in each coordinate to $(1, 0, 0, 0, 0)$. But it stands out from the others as converging excessively slowly. At $H = 65{,}015$, the limit of our hardware, it is just short of being stable to only two decimal places.

Because $\mathbf{I}_4$ and $\mathbf{H}_1$ have nonzero coordinate sequences converging to zero, we conclude that they are not AC. Groupoids $\mathbf{H}_2, \mathbf{H}_3$ and $\mathbf{H}_4$ each have a 2-element subgroupoid which we also verified to be AC, but have not displayed the probability sequences. Consequently $\mathbf{P}_1, \mathbf{P}_2, \mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3, \mathbf{H}_2, \mathbf{H}_3$ and $\mathbf{H}_4$ are all AC.

Idemprimality. Looking at our controls in Fig. 6, we can quickly check that $\mathbf{P}_1$ and $\mathbf{P}_2$ are both primal by Rousseau's Theorem 1, and therefore satisfy IPr. An examination of $\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3$ and $\mathbf{H}_4$ shows that none of them have a discriminator term since each one has a 2-element subgroupoid that does not have one. By Pixley and Werner's Theorem 3 these last four do not satisfy IPr.

The remaining four, $\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3$ and $\mathbf{I}_4$, were a bit problematic. Each has exactly one idempotent and no nontrivial subalgebras, congruences or automorphisms. Consequently, by Pixley and Werner's Theorem 3, each one satisfies IPr if and only if it has a discriminator term. Murskiĭ's Theorem [8] suggests that it is likely that they all do. But showing that they really do raised the same problem that we had with the groupoid **Pi** in the Introduction. Our best strategy at that point appeared to be hoping that they do all have discriminator terms and seeing if the DDA was able to find them. Fortunately that strategy paid off.

Table 4. Stars indicate satisfaction of NSR, AC and IPr for 5-element controls.

	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{I}_1$	$\mathbf{I}_2$	$\mathbf{I}_3$	$\mathbf{I}_4$	$\mathbf{H}_1$	$\mathbf{H}_2$	$\mathbf{H}_3$	$\mathbf{H}_4$
NSR	*	*	*	*	*	*	*	*	*	*
AC	*	*	*	*	*			*	*	*
IPr	*	*	*	*	*	*				

As a first application of the DDA, we applied it to a search for these four discriminator terms. Each of the first three came in under 20 s on their first run. Given their search space sizes of $5^{5^3-1} > 10^{86}$, this appeared to be a good first test of the DDA. But it was only after 3 failed 10 h runs and 12 runs capped at 10 min each that $\mathbf{I}_4$ finally yielded a discriminator term, in fact one remarkably similar in run time and length of the others. We give here run time in seconds and term length for each successful run:

$$\mathbf{I}_1 : 13.29/2090,\ \mathbf{I}_2 : 14.28/2233,\ \mathbf{I}_3 : 18.26/2773,\ \mathbf{I}_4 : 14.78/2384.$$

Consequently $\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3$ and $\mathbf{I}_4$ all satisfy IPr.

Table 4 summarizes the conditions satisfied by these 10 controls.

We now report on systematic tests of the performance of these controls under the DDA. In Sec. 6, we will then use these data as a baseline of expected performance of the DDA to judge its performance on other groupoids. Our goal in both sections will be to further demonstrate the efficacy of the algorithm itself and to examine the role of the three conditions, NSR, AC and IPr, in its success.

Our first task was to determine the target operations to use to test the DDA. The ternary discriminator provided a good uniformly defined target in [13]. But we will want to be able to test the DDA on any groupoid, including some which do not have a discriminator term. For this purpose, we generated 10 random ternary terms in variables x, y, z, each of length 100, as follows. Begin with a random choice of t_1 in $\mathcal{T}_2 \setminus \mathcal{T}_1$. Given t_j, randomly choose a subterm s of t_j and a new random term $u \in \mathcal{T}_2 \setminus \mathcal{T}_1$. Obtain t_{j+1} from t_j by replacing s in t_j by either (su) or (us). Then t_j will have length $2j$ for each positive integer j, and, by induction, every term of length $2j$ can be obtained in this way. The process ends at $j = 50$ and we use t_{50} as the required random term of length 100.

With each control and test groupoid, we used these same 10 random terms to calculate 10 ternary operations that we knew in advance to be term operations of that groupoid. We then did 10 DDA runs with each, one with each of the 10 targets. In this way, we were guaranteed that each of our DDA runs sought a term that did indeed exist. In Table 5, we summarize the results, giving the *Max*imum, *Mean* and *Min*imum of the ten run times and giving the *M*ean *L*engths of outputs for the successful runs. By "Failed" we mean no output after 10 h.

The groupoid $\mathbf{I}_4$ succeeded on the first three runs in a mean time of 7.76 s and a mean term length of 1363. On the fourth run, it failed to give an output after 10 h. The groupoid $\mathbf{H}_1$ succeeded on the first three runs in a mean time of 179.23 s

Table 5. Control data: time (s) and mean length for 10 runs on each of 10 random targets.

Ctrl:	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{I}_1$	$\mathbf{I}_2$	$\mathbf{I}_3$	$\mathbf{I}_4$, $\mathbf{H}_1$	$\mathbf{H}_2, \mathbf{H}_3, \mathbf{H}_4$
Max	11.72	9.77	14.50	12.91	16.71	Mixed	Failed
Mean	8.70	6.19	10.57	8.92	13.82	Success	
Min	4.11	1.94	7.34	4.79	10.13		
M L	1522	1077	1938	1549	2415		

and a mean term length of 29,793. On the fourth run, it failed to give an output after 10 h.

To summarize, five of our 10 control groupoids satisfied all three conditions: NSR, AC and IPr. These five, and only these five, succeeded in each of 10 runs; each under 20 s. The other five each failed at least one run after 10 h.

6. DDA Test Data

Tables 4 and 5 for our random sample of control groupoids show a perfect correlation between consistent and efficient success of the DDA and satisfaction of conditions NSR, AC and IPr. We have not yet seen a single example in which these three conditions hold but the DDA fails. In this section, we present further tests of this correlation by looking at groupoids which do not satisfy all three conditions, groupoids with small search spaces and groupoids with fewer or more than five elements. These tasks confront us with the problem that many of the groupoids required for these tests are either hard to find or cumbersome to use. Consequently, we needed to do a qualitative analysis through case studies of particular examples we could find rather than a quantitative analysis with randomly generated controls.

In this section, we will see examples showing that if any one of the conditions NSR, AC and IPr fails, the DDA can fail even with the other two being satisfied. The mixed success with $\mathbf{I}_4$ will show how, in the presence of NSR and IPr, marginal failure of AC can still allow occasional successes. We will also see that the DDA can succeed when the search space of ternary term operations is too small, regardless of which of the conditions NSR, AC and IPr hold. The mixed success with $\mathbf{H}_1$ will show how a smaller than normal search space and satisfaction of only NSR can still lead to some successes. We have used the same fixed sample of 10 randomly constructed terms to generate targets to test the DDA. Our last examples will look for evidence that these samples are indeed representative.

Tables for several new test groupoids are shown in Fig. 7. All of the test groupoids we will need are listed in Table 6, along with the conditions they satisfy. These examples will be discussed as they are presented.

6.1. *Failure of DDA without NSR, AC and IPr*

NSR. The groupoid $\mathbf{Q}$ in Fig. 7 is a quasigroup and therefore fails NSR because the $\neq$ relation is a separating relation. It satisfies IPR because it is primal by Rousseau's

#2393

∗	0	1	2
0	1	0	0
1	0	0	2
2	0	2	0

#571

∗	0	1	2
0	0	0	1
1	0	2	1
2	2	1	1

Q

∗	0	1	2	3	4
0	1	4	0	2	3
1	3	2	4	0	1
2	2	0	3	1	4
3	0	3	1	4	2
4	4	1	2	3	0

NQ

∗	0	1	2	3	4
0	1	4	0	2	3
1	3	2	4	0	1
2	2	0	3	2	4
3	0	3	1	4	2
4	4	1	2	3	0

Fig. 7. Tables for new test groupoids.

Table 6. Satisfaction of NSR, AC and IPr for 12 test groupoids.

	Q	**NQ**	$\mathbf{H}_2$	$\mathbf{H}_3$	$\mathbf{H}_4$	$\mathbf{I}_4$	**C**	$\mathbf{Z}_5$	$\mathbf{H}_1$	**#2393**	**#571**	**PI**
NSR		∗	∗	∗	∗	∗	∗		∗	∗	∗	∗
AC	∗	∗	∗	∗	∗			∗				∗
IPr	∗	∗				∗						∗

Theorem 1. Taking $k = 10$ and $\vec{d} = (1, 4, 4, 0, 2, 1, 3, 1, 4, 2)$, the spreadsheet shows that $\beta_{\vec{d}}$ quickly stabilizes to the uniform distribution to six decimal places with

$$\mathbf{Q} : (0.200000, 0.200000, 0.200000, 0.200000, 0.200000) \text{ at } H = 5,$$

suggesting that **Q** is asymptotically complete. Consequently **Q** satisfies AC and IPr but not NSR. Three DDA runs were done on **Q**, two using the discriminator operation and one using the random operations as targets, with no successes after running 10 h each.

To see just how sensitive the DDA is to the NSR condition, we created the new groupoid **NQ** from **Q** by changing only the value of $2 * 3$ from 1 to 2. It is easy to check that **NQ** is also primal but has NSR, therefore satisfying IPr and NSR. Calculating $\beta_{\vec{d}}$ for **NQ** with the same choice of $\vec{d}$, we find that it also quickly converges to a distribution tipped slightly away from 1 and toward 2:

$$\mathbf{NQ} : (0.196952, 0.149112, 0.253748, 0.202897, 0.197291) \text{ at } H = 12.$$

Running the DDA on **NQ**, we found a discriminator term in 10.84 s. We found terms for all 10 random operations in minimum/mean/maximum time of 8.72/10.03/12.70 s with a mean length of 1973.

This performance of **NQ** is altogether consistent with that of the five controls that fulfill conditions NSR, AC and IPr. Together **Q** and **NQ** show how sharply the NSR condition can separate failure from success for the DDA.

AC. We found but one example that fails AC but satisfies NSR and IPr. That is $\mathbf{I}_4$, and an illuminating example it is. It fails AC, so we need an explanation as to why the DDA was able to find an $\mathbf{I}_4$ discriminator term on the 16th run after it had failed in 15 prior runs. Being idemprimal with one idempotent, its search space size is $5^{5^3-1} > 10^{86}$, way too large for anything to be found by accident.

The answer is at the core of the proof of the Continuity Theorem of [2]. The Central Lemma 20 uses NSR to deduce the existence of a sequence of elements of the groupoid that will collapse two computations to the same value. It is then argued that this sequence has a positive probability of occurring since, by AC, each element has a positive probability of occurring. Our Table 3 shows that the probabilities of 1, 2, 3 and 4 occurring each descends monotonically to zero — leading to failure of AC. But they descend to zero *extremely slowly*. That could mean that, on some relatively short runs, those probabilities remain high enough to force the required sequence to occur. The result would be that the DDA applied to $\mathbf{I}_4$ would fail on normal long runs when the probabilities fell too low, but would occasionally succeed if the computation went quickly in the right direction.

This is exactly what we saw: failure of the DDA on the searches for random targets and failure on many discriminator runs; then a 14.78 s run producing a discriminator term. It is a fine illustration of the important role of AC. Although $\mathbf{I}_4$ is technically not AC, the DDA on $\mathbf{I}_4$ can occasionally finish its work before the failure of AC becomes an impediment.

IPr. Theorem 11 tells us one way the DDA might fail if the groupoid is not idemprimal. Without idemprimality, the DDA might create an array that has no solution, and would search that branch indefinitely without finding one. Such a failure is illustrated in Example 16. Our controls $\mathbf{H}_2$, $\mathbf{H}_3$ and $\mathbf{H}_4$ appear to also all fall into this trap. None of them satisfy IPr, but all satisfy NSR and AC (Table 4). The DDA spent 10 h with each of them unsuccessfully looking for a target term that was known to exist. This performance contrasts sharply with the controls that consistently succeeded in under 20 s.

6.2. *Success of DDA with small search spaces*

The examples of the last subsection suggest that NSR, AC and IPr may be necessary for the DDA to succeed. But this is not the case. We will now look at a variety of examples in which a small search space allows the DDA to succeed without NSR, AC and IPr. By an **iteration** of the DDA, we mean the full sequence of consecutive steps it makes at one node before arriving at the next node, that is, one cycle of the steps (1)–(7) (or (5)–(7)). Thus a run of the DDA consists of a sequence of iterations, the first taking place at the root node. Our first example is simple enough to demonstrate a failure of AC and success of the DDA by hand.

Theorem 12. *Let* $\mathbf{C} = \langle C, * \rangle$ *be a nontrivial finite groupoid for which* $*$ *has constant value* $c \in C$.

(i) *The groupoid* $\mathbf{C}$ *satisfies NSR but neither AC nor IPr.*
(ii) *For every* $k > 1$, *the search space of* k*-ary term operations has size* $k + 1$.
(iii) *For every* $k > 1$, *the DDA will solve every* k*-ary array that has a solution over* $\mathbf{C}$ *on the first or fourth iteration.*

Proof. (i) For NSR, let σ be a non-empty binary relation on C with $(a, b) \in \sigma$. Then $(a * c, b * c) = (c, c) \in \sigma$ so σ is not a separating relation. From the definition of idemprimality, it immediately follows that **C** does not satisfy IPr.

For AC we show, for every $k > 1$, that **C** is not asymptotically k-complete at every k-tuple other than $(c, c, \ldots, c)$. Choose $\vec{d} \neq (c, c, \ldots, c)$ in C^k and let b be a coordinate of $\vec{d}$ other than c. Then $\beta_{\vec{d},b}(1) > 0$. Since the only terms with value b at $\vec{d}$ are single variables, $\beta_{\vec{d},b}$ converges to zero so **C** is not AC.

(ii) The only k-ary term operations on **C** are represented by the $k + 1$ terms $x_0, x_1, \ldots, x_{k-1}$ and x_0^2.

(iii) Let $k > 1$ and let T be a k-ary array that has a solution over **C**. Assume the DDA does not find a solution in the first iteration, that is, $\mathfrak{T}_m$ does not contain a solution to T. Since $\mathfrak{T}_m$ does not contain terms representing all $k+1$ term operations, it must be that $m = 1$ and x_0^2 is a solution to T. Thus $c \in T(\vec{d})$ for all $\vec{d} \in C^k$. The first iteration ends by adding two nodes above the root and choosing one for the second iteration. Assume the left node is chosen.

Let A be the array with $A(\vec{d}) = C$ for all $\vec{d} \in C^k$. Since $c \in T(\vec{d})$ for all $\vec{d}$, we have $LT = A$. Every term is then a solution to LT, so the DDA will randomly chose some solution $x_i \in \mathfrak{T}_1$. It will then calculate the array $[x_i, T] = A$ and assign it to the right node. On the third iteration, it will again randomly choose a solution $x_j \in \mathfrak{T}_1$ to $[x_i, T] = A$. On the fourth iteration, it will return to the root, assigning it the solution $x_i x_j$ to T. □

Example 13. Let $\mathbf{Z}_5 = \langle \{0, 1, 2, 3, 4\}, + \rangle$ be the groupoid of integers modulo five and take $k = 3$.

(i) The groupoid $\mathbf{Z}_5$ satisfies AC but neither NSR nor IPr.
(ii) The search space of ternary operations on $\mathbf{Z}_5$ has size 125.
(iii) The DDA found terms for all ten random term operations in a total of 1.2 s.

Each ternary term operation on $\mathbf{Z}_5$ is expressible as $ax + by + cz$ for some $a, b, c < 5$, giving us (ii). No such term defines the discriminator operation, so $\mathbf{Z}_5$ does not satisfy IPr. Like **Q**, it is a quasigroup so it does not satisfy NSR. And like **Q**, its probability distributions converge to the uniform distribution, so it is AC and we have (i).

Example 14. The groupoid $\mathbf{H}_1$ satisfies only one of our three conditions: NSR. Even so, the DDA produced terms for four random operations on $\mathbf{H}_1$ before it failed. How can these successes be explained?

While we are not in a position to make a conclusive argument, Table 3 suggests that the search space of term operations for $\mathbf{H}_1$ is particularly small relative to the other control groupoids. There we see that the computation of probability sequences for $\mathbf{H}_1$ required only 17 iterations to reach 6-place convergence to the limit (0,0,0,0,1). Thus almost all terms of height 17 or more have the same term operation, namely, the operation with constant value 4.

Table 7. Discriminator terms from the DDA: time (in seconds) and length.

	Pi	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{I}_1$	$\mathbf{I}_2$	$\mathbf{I}_3$	$\mathbf{I}_4$	**NQ**
Time	12.74	10.45	14.23	13.29	14.28	18.26	14.78	10.84
Length	1617	1598	2266	2090	2233	2773	2384	1820

6.3. *Extrapolation beyond random samples*

The intention of our use of randomly generated term operations was to obtain a sample of all term operations, but it is hard to know just how representative this sample is. As an example of a possibly different operation, we considered the ternary discriminator. From [9, 14] we know that a nontrivial finite set with the ternary discriminator operation and a constant term for each element in the set is primal. This suggests that the ternary discriminator is a particularly complex operation, as it can generate all operations with the help of only constants. As a last test on 5-element groupoids, we collected here the results of discriminator runs on our control and test groupoids that satisfy NSR, AC and IPr to see if the results were reasonably similar to those of the random term operations in Table 5. Table 7 shows that they were indeed reasonably similar, with only slightly higher run times and lengths. (Note here that this is all first run data except for $\mathbf{I}_4$ where it is run 16 after 15 failed.)

We would like to see if the performance characteristics of the DDA with 5-element groupoids extends to smaller and larger examples as well. Conducting experiments either way leads to new challenges. Smaller groupoids have smaller search spaces, and we have seen how small search spaces can lead to misleadingly positive results since they tend to circumvent the important difficulties. Figure 3 illustrates the very rapid growth in search space size as a function of groupoid size. Larger groupoids with larger search spaces will take the DDA much longer, making experiments impractical to do.

Failure of the DDA on a small groupoid that fails one or more of our three conditions would provide a correspondingly more convincing demonstration of the importance of those failed conditions. But finding small examples on which the DDA fails is a challenge. Known small examples on which NSR fails are associated with quasi groups and tend to have very small search spaces, like $\mathbf{Z}_5$. So we need an example with NSR. Examples with large enough search spaces to fail tend to have a discriminator term. But we want it to fail IPr, so we cannot use Rousseau's Theorem to identify ones with a discriminator term. And we cannot use the DDA to find a discriminator term since we want to find examples for which the DDA fails.

We overcame this dilemma by turning to the Berman and Burris Catalog [1] of all 3-element groupoids. Among the 3,330 isomorphism types it identifies, it lists 62 that are semiprimal. Among those 62 we found two, shown in Fig. 7, that are not AC. Groupoid **#571** has a 2-element primal subalgebra with the third element idempotent, and therefore has a ternary search space size of $2^8 3^{18} \approx 10^{11}$.

Table 8. Failed DDA run to solve array T over **#571** with $m = 1$.

Array	0	1	2	Solution
T	0	2	2	
LT	01	12	12	x
	0	1	2	$\hookleftarrow$
$[x, T]$	01	1	0	None!

The other, **#2393**, has only a single 2-element primal subalgebra and therefore a ternary search space size of $2^3 3^{19} \approx 3 \times 10^{11}$. Both are smaller than the search spaces of 3-element primals, where the DDA consistently finds discriminator terms in approximately 0.10 s.

Example 15. Both of the semiprimal groupoids **#2393** and **#571** have NSR but are neither IPr nor AC. The DDA failed to produce a discriminator term for either one within 10 h.

The semiprimal groupoid **#571** can also be used to prove that Theorem 11 does not extend from idemprimality to semiprimality. The unary array (operation) T in Table 8 preserves both subgroupoids $\{0\}$ and $\{1, 2\}$ of **#571** and therefore has a solution. For example, $(xx^2)^2$ is a solution.

Example 16. Table 8 illustrates a run of the DDA with the semiprimal groupoid **#571**, the array T and $m = 1$ as inputs. On its third iteration (defined in Example 9), this run generates the array $[x, T]$ which has no solution since $[x, T](2) = \{0\}$ but 0 is not in the subgroupoid $\mathrm{sg}\{2\} = \{1, 2\}$. Consequently this DDA run will never terminate.

As a larger example, we constructed a table for the 10-element groupoid **PI** shown in Fig. 8 from the first 100 digits of π. Seeing no recognizable connection between the DDA and the number π, we expect that **Pi** is a good example of a

PI

$*$	0	1	2	3	4	5	6	7	8	9
0	1	4	1	5	9	2	6	5	3	5
1	8	9	7	9	3	2	3	8	4	6
2	2	6	4	3	3	8	3	2	7	9
3	5	0	2	8	8	4	1	9	7	1
4	6	9	3	9	9	3	7	5	1	0
5	5	8	2	0	9	7	4	9	4	4
6	5	9	2	3	0	7	8	1	6	4
7	0	6	2	8	6	2	0	8	9	9
8	8	6	2	8	0	3	4	8	2	5
9	3	4	2	1	1	7	0	6	7	9

Fig. 8. Groupoid table of the first 100 decimal digits of π.

randomly chosen 10-element groupoid for our purposes. It illustrates what the DDA can do within a 24-h period.

Example 17. The groupoid **PI** of Fig. 8 has a single idempotent, 9, and otherwise no nontrivial subalgebras, automorphisms or congruences. In 22 h and 11 s, the DDA found a discriminator term for **PI** of length 202,612 (not displayed). By Theorem 3 it is idemprimal, and therefore has a ternary search space of size $10^{10^3-1} = 10^{999}$.

7. Conjectures and Open Questions

As stated at the end of the introduction, we are interested in knowing for which finite groupoids will the DDA efficiently solve the TGP and whether these groupoids are rare or common. In this section, we will state a number of conjectures that are supported by the evidence we have collected. We will also give a number of interesting open questions that we cannot yet answer.

In order to state these conjectures and questions, we need to start by saying exactly what it means for the DDA to "solve" the TGP for a finite groupoid **G**. A first attempt might be to say that the DDA "solves" the TGP for **G** if every run of the DDA for a solution to an array over **G** that has a solution will eventually terminate with a solution. For example, Theorem 12 shows that this is true when the operation on **G** is constant. However, the discussion immediately following Theorem 8 shows why this condition is way too strong. The DDA requires the frequent random choices to minimize the number of failed runs, even in the presence of NSR, AC and IPr.

This means that we can only ask that the DDA "solve" the TGP for **G** in a probabilistic sense. Given the target array assigned to a node, there are only finitely many different iterations it could make at that node, resulting from the different possible random choices. Consequently, given an original target array T, there are, for each positive integer J, only finitely many sequences of iterations it could make starting with target T that either have length J or terminate prior to J iterations. We will call these J-**runs** for T.

We will say that the DDA **solves the TGP** for a finite groupoid **G** if, for every target array T over **G** that has a solution, the proportion of J-runs that terminate approaches 1 as J approaches infinity. Based on this definition, we can state our primary conjecture.

Conjecture 18. *The DDA solves the TGP for every finite groupoid that satisfies NSR, AC and IPr.*

The main justification for this conjecture comes from the results of our control experiments. Of 10 randomly constructed 5-element groupoids, five satisfied all three of our conditions NSR, AC and IPr (Table 4). Ten runs of the DDA were done on each control groupoid with 10 different random targets. The five satisfying

our three conditions, and only those five, succeeded on all 50 runs in less than 20 s each (Table 5).

Proving this conjecture would be a significant step forward. Theorem 11 tells us how IPr can help to prevent the DDA from failing. The Continuity Theorem 4.2 says that NSR and AC together imply TC. A first step toward proving Conjecture 18 ought be some kind of analog of Theorem 11 for TC.

Problem 1. *Prove some theorem that explains how TC* (*or the combination of NSR and AC*) *helps to prevent the DDA from failing, perhaps in combination with IPr.*

Even if Conjecture 18 is not exactly correct as it stands, there is surely some good explanation as to why NSR and AC help to prevent failure of the DDA, as IPr does, in so many of our tests.

Example 16 showed that Theorem 11 can not be extended in the natural way to all semiprimal groupoids. But perhaps it can be extended to some subclass of semiprimals. This would lead to a strengthened version of Conjecture 18 by weakening IPr to include these semiprimal groupoids.

Problem 2. *Are there groupoids that are semiprimal but not idemprimal which, under the DDA starting with a target array that has a solution, will only produce auxiliary arrays that have a solution*?

If Conjecture 18 is true, its significance will depends on how frequently these three conditions hold. Murskiĭ's Theorem 2 tells us that the proportion of n-element groupoids satisfying IPr approaches 1 as n approaches infinity. This statement is commonly expressed as, "Almost all finite groupoids are idemprimal." We have seen ample informal evidence that the same is true of NSR, but we have not seen a proof.

Conjecture 19 (Clark [2]). *Almost all finite groupoids have NSR.*

We have by no means the same evidence for the ubiquity of AC. For all but the very simplest groupoids, we have relied on spreadsheet computations to guess whether or not they were AC, but this evidence is never by itself conclusive. In contrast to the efficient algorithm of [2] to test for NSR, we do not know if AC is decidable at all.

Problem 3. *Is AC a decidable property of finite groupoids*?

Although we have no answer, we have not seen an example in which our spreadsheet data did not make a convincing case one way or the other for a particular substitution sequence $\vec{d}$. Part of the difficulty is that AC asks that we test *all* choices of $\vec{d} \in G^k$ for *all* choices of k, yet we can only test finitely many. Informal experiments we have done are consistent with the following conjecture. If it is

true, it would tell us that we only need to test one choice of $\vec{d}$ for each nontrivial subgroupoid of $\mathbf{G}$. If $\mathbf{G}$ is idemprimal, that would reduce to only one test!

Conjecture 20. *Let $k, k' > 1$ be integers, let $\mathbf{G}$ be a finite groupoid, let $a \in G$ and assume that $\vec{d} \in G^k$ and $\vec{d'} \in G^{k'}$ with $\mathrm{sg}(\vec{d}) = \mathrm{sg}(\vec{d'})$.*

(i) *If $\beta_{\vec{d},a}$ is eventually bounded away from zero, then $\beta_{\vec{d'},a}$ is eventually bounded away from zero.*
(ii) *If $\beta_{\vec{d}}$ converges, then $\beta_{\vec{d'}}$ converges to the same limit.*

Programming spreadsheet computations is quite tedious for $n > 5$, so we have not tested AC beyond that limit. Yet the evidence we do have suggests that AC is common, and more common with larger groupoids.

Problem 4. *Demonstrate in some computational or theoretical way that the proportion of n-element groupoids that satisfy AC grows large as n approaches infinity.*

Quasigroups form one class of groupoids for which the AC problem might be tractable. Groupoids $\mathbf{Z}_5$ and $\mathbf{Q}$ offer evidence of the next conjecture.

Conjecture 21. *If $\mathbf{G}$ is a finite quasigroup, then it satisfies AC with the probability distributions for each $\vec{d} \in G^k$ converging to the uniform distribution over $\mathrm{sg}(\vec{d})$.*

What can be said about the converse of Conjecture 18? Looking at our control results in Tables 4 and 5, we see that the five controls not satisfying all three of NSR, AC and IPr, and only those five, were not able to find terms for all 10 random operations in under 20 s. In fact, all five were terminated after unsuccessfully working on one of them for 10 h. In spite of this *evidence*, the converse of Conjecture 18 is not in general true. This is suggested by all of the examples in Sec. 6.2 where DDA runs were successful on groupoids that failed at least one of these three conditions. The fact that the converse of Conjecture 18 is false follows from Theorem 12.

Corollary 22. *If a nontrivial groupoid is constant, then the DDA solves its TGP, yet it does not satisfy IPr.*

All of these counterexamples to the converse of Conjecture 18 are attributable to search spaces that are too small to require NSR, AC and IPR for success of the DDA. This observation suggest a final overarching conjecture. If Conjectures 18 and 19 were true and Problem 4 were solved by proving that almost all finite groupoids are AC, then the following would be true.

Conjecture 23. *The DDA solves the TGP for almost all finite groupoids.*

Acknowledgments

The authors would like to give a special thanks to the anonymous referee whose careful reading and thoughtful suggestions significantly improved this exposition.

References

[1] J. Berman and S. Burris, A computer study of 3-element groupoids, *Logic Algebra* (the Proceedings of the Magari conference), Aldo Ursini and Paolo Aglian, eds. (Marcel Dekker Inc., New York, NY, USA, 1996), pp. 379–430.

[2] D. Clark, Evolution of algebraic terms 1: Term to term operation continuity, *Int. J. Algebra Comput.* **23**(5) (2013) 1175–1205.

[3] D. Clark, B. Davey, J. Pitkethly and D. Rifqui, Flat unars: The primal, the semiprimal and the dualisable, *Algebra Universalis* **63**(4) (2010) 303–329.

[4] D. Clark, M. Keijzer and L. Spector, Evolution of algebraic terms 3: A co-evolutionary algorithm (in preparation).

[5] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing* (Springer-Verlag, New York, NY, USA, 2003).

[6] R. Freese, E. Kiss and M. Valeriote, UACalc, a Universal Algebra Calculator, Available at: www.uacalc.org (2011).

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

[8] V. L. Murskiĭ, Konéčnaá baziruémost′ toždéstv i drugié svojstva "počti vséh" konéčnyh algébr (A finite basis of identities and other properties of "almost all" finite algebras), *Problémy Kibérnétiki* **30** (1975) 43–56.

[9] A. Pixley, Functionally complete algebras generating distributive and permutable classes, *Math. Z.* **114** (1970) 361–372.

[10] I. Rosenberg, Über die funktionale Vollständigkeit in den mehrwertigen Logiken (Struktur der Funktionen von mehreren Veränderlichen auf endlichen Mengen), *Rozpravy Československé Akad. Věd Řada Mat. Přírod.* **80** (1970) 1–93.

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

[12] C. E. Shannon, A symbolic analysis of relay and switching circuits, *Transactions of the American Institute of Electrical Engineers* **57**(12) (1938) 713–723, doi:10.1109/T-AIEE.1938.5057767.

[13] L. Spector, D. Clark, B. Barr, J. Klein and I. Lindsay, Genetic programming for finite algebras *Genetic and Evolutionary Computation Conference* (GECCO) 2008 Proceedings, Atlanta GA (July 2008), Editor-in-Chief Maarten Keijzer, Association for Computing Machinery (ACM), ISBN: 978-1-60558-130-9, pp. 1291–1298. [Paper won first place in the GECCO 2008 Human Competitive competition.] ⟨www.genetic-programming.org/hc2011/combined.html⟩.

[14] H. Werner, Eine Charakterisierung funktional vollständiger Algebren, *Arch. Math.* (Basel) **21** (1970) 381–385.