

The Monoids of Orders Eight, Nine & Ten

Andreas Distler¹ and Tom Kelsey²

¹ School of Mathematics and Statistics, Mathematical Institute,
North Haugh, St Andrews, KY16 9SS, UK
`andreas@mcs.st-andrews.ac.uk`

² School of Computer Science, Jack Cole Building,
North Haugh, St Andrews, KY16 9SX, UK
`tom@cs.st-andrews.ac.uk`

Abstract. We describe the use of symbolic algebraic computation allied with AI search techniques, applied to the problem of the identification, enumeration and storage of all monoids of order ten or less. Our approach is novel, using computer algebra to break symmetry and constraint satisfaction search to find candidate solutions. We present new results in algebraic combinatorics: up to isomorphism and anti-isomorphism, there are 858,977 monoids of order eight; 1,844,075,697 monoids of order nine and 52,991,253,973,742 monoids of order ten.

1 Introduction

The aim of this paper is to find all solutions to a class of problems in algebraic combinatorics. This is a well-known research area: it is natural when discussing, for example, various types of latin squares to try to resolve the question of how many of each type exist. As well as obtaining the correct answer in terms of number of solutions, we aim to store each solution so that they can be analysed in terms of their structure by algebraists. This second aim means that we are not searching for a purely constructive solution to the enumeration problem; we want to generate and store a canonical example from each equivalence class of solutions.

The On-Line Encyclopedia of Integer Sequences [1] contains numerous examples of known initial sequences of enumerations of algebraic and combinatorial structures. The sequence that this paper extends is A058133: the numbers of monoids of order n , considered to be equivalent when they are isomorphic or anti-isomorphic. Currently values for $n \leq 7$ are available online, values for $n = 8$ and 9 were given in [2], the value for $n = 10$ is a new result.

A monoid is an algebraic structure equipped with a closed and associative binary operator, and an identity element. The formal definition used throughout this paper is the following:

Definition 1. A monoid is a tuple $\langle M, *, e \rangle$ where M is a set; $*$: $M \times M \rightarrow M$ satisfies $x * (y * z) = (x * y) * z \quad \forall x, y, z \in M$; and the identity $e \in M$ satisfies $x * e = x = e * x \quad \forall x \in M$.

If $\langle M, *, e \rangle$ is a monoid, and $|M| = n$, then $\langle M, *, e \rangle$ has order n .

Not requiring an identity element gives the definition of a *semigroup* – denoted henceforth as $\langle S, * \rangle$. A *group* – denoted as $\langle G, *, e, {}^{-1} \rangle$ – is a special case of a monoid; each group element has a multiplicative inverse.

Throughout this paper we consider only finite monoids, where M is a finite set. We also simplify denotation of monoid, semigroup and group to M , S and G respectively, whenever the multiplication, identity and/or inverses are either unimportant or clear from the context.

Definition 2. Let $\langle M_1, *_1, e_1 \rangle$ and $\langle M_2, *_2, e_2 \rangle$ be two monoids of the same order. A bijection $\sigma : M_1 \rightarrow M_2$ is an isomorphism if it respects the multiplication – i. e. $(a *_1 b)^\sigma = a^\sigma *_2 b^\sigma$ – and an anti-isomorphism if it inverts the multiplication – i. e. $(a *_1 b)^\sigma = b^\sigma *_2 a^\sigma$. If such a bijection exists the monoids are isomorphic, respectively anti-isomorphic, and are equivalent if either.

Remark 1. Since the identity element of a monoid does not form part of the definition of equivalence, the above definition also applies to semigroups.

In this paper we are interested in monoids up to equivalence. Thus we can choose the underlying set to be $\{1, 2, \dots, n\}$. We then represent monoids by their multiplication tables, with rows and columns indexed from 1 up to n .

Table 1: Example multiplication table, and its image under permutation (1, 4)

$*_1$	1	2	3	4		$*_2$	1	2	3	4
1	1	2	3	4		1	3	1	3	1
2	2	1	3	4		2	1	4	3	2
3	3	3	3	3		3	3	3	3	3
4	4	4	3	3		4	1	2	3	4

Isomorphism of monoids induces an action on tables. Given a permutation g of the members of M , we modify the table by permuting each row according to g , then each column, and finally permuting the values. An anti-isomorphism is the result of an isomorphism action followed by transposing the table. The effect of applying permutation (1, 4) is shown in Table 1. Since permutations of a finite set, permutations of rows and columns of a multiplication table, and table transposition are all invertible, (anti-)isomorphism is a reflexive, symmetric and transitive relation on monoids, and is hence an equivalence relation.

In common with many algebraic enumeration problems, there is a combinatorial explosion as n increases. There are n choices for each of the n^2 positions in a multiplication table. For $n = 9$ and 10 , the number of choices is approximately 2×10^{77} and 10^{100} respectively. This increase in problem size effectively rules out the obvious exhaustive search approach of generating each table, checking if the monoid axioms hold, then checking whether or not an (anti-)isomorphic version of the table has already been found.

Our approach is to develop algebraic results that allow us to devise algorithms for search-space reduction. We implement these algorithms in symbolic computational algebra; formulate the remaining problems in terms of constraints

on solution tables; use advanced AI backtrack search techniques to find solutions; then (in some cases) use computational algebra to decide which canonical representative of (anti-)isomorphic equivalence class to accept as our unique solution.

In the next section we describe the computational algebra software and the constraint solver used; we provide the basic principles of Constraint Satisfaction and formalise notions regarding symmetry breaking in Constraint Satisfaction Problems; at last we present results about finite monoids and their structure. In Section 3 we give a detailed derivation of three algebraic and AI search approaches to the problem, together with proofs of soundness and completeness. We summarise our results in Section 4, and provide concluding remarks and an indication of future avenues of research in Section 5.

2 Tools & Underlying Theory

2.1 Computational Algebra & GAP

Since our problem domain involves binary operators on finite sets, permutations, identity elements, action homomorphisms and symmetry groups, we use specialist software that provides robust, efficient and extensive implementations of algorithms in abstract algebra. GAP [3] (Groups, Algorithms and Programming) is a system for computational discrete algebra with particular emphasis on, but not restricted to, computational group theory. GAP provides a large library of functions which implement algebraic algorithms.

For our purposes, any advanced computational algebra system could be used. However, we rely on efficient GAP code that tests canonicity of an image of a set of points under the action of a permutation group [4]. This allows isomorphic results to be eliminated efficiently.

2.2 Constraint Satisfaction & Minion

Definition 3. *A Constraint Satisfaction Problem (CSP) L is a set of constraints \mathcal{C} acting on a finite set of variables $\Delta = \{A_1, A_2, \dots, A_n\}$, each of which has a finite domain of possible values $D_i = D(A_i)$. A solution to L is an instantiation of all of the variables in Δ such that no constraint in \mathcal{C} is violated.*

The class of CSPs is a generalisation of propositional satisfiability (SAT), and is therefore NP-complete. Solvers typically proceed by building a search tree, in which the nodes are assignments of values to variables and the edges lead to assignment choices for the next variable. If at any node a constraint is violated, then search backtracks. If a leaf is reached, then no constraints are violated, and the assignments provide a solution. A recent advance in constraints programming is the “model and run” methodology, of users building constraint models and then executing them on a solver with few options. This methodology inspired the development of the constraint solver Minion [5]. A major feature of modern SAT solvers is their optimised use of modern computer architecture. Using this approach, Minion has been designed to minimise memory usage. The result of

this is that Minion claims to offer *fast, scalable* constraint solving. Scalability as problem size increases is an important (and also neglected) factor in constraint solver construction. A key aim of our research is to test the claimed scalability of Minion.

2.3 Symmetry Breaking in CSPs

CSPs are often highly symmetric. Given any solution, there can be others which are equivalent in terms of the underlying problem. Symmetries may be inherent in the problem, or be created in the process of representing the problem as a CSP. Without symmetry breaking (henceforth SB), many symmetrically equivalent solutions may be found and, often more importantly, many symmetrically equivalent parts of the search tree will be explored by the solver. A SB method aims to avoid both of these problems.

Permutation groups are the mathematical structures that best encapsulate symmetry. We describe the symmetries of a CSP as a permutation group of the literals (variable-value pairs) of the CSP. Assignments of the form $(Var = val)$ are called *literals*, so partial and full assignments are conjunctions of literals. We denote the set of all literals by χ , and adopt the convention of denoting variables by Roman capitals and values by lower case Greek letters. Clearly any conjunction of literals that involves two values assigned to the same variable can not represent a solution, nor can a conjunction of literals with some variable assigned to no value.

Definition 4. *Given a CSP L , with a set of constraints \mathcal{C} , and a set of literals χ , a symmetry of L is a bijection $f : \chi \rightarrow \chi$ that preserves the set of solutions to L .*

This definition, adapted from the definition for solution symmetries in [6], ensures that solutions are mapped to solutions and non-solutions are mapped to non-solutions.

We denote the image of a literal $(X = \alpha)$ under a symmetry g by $(X = \alpha)^g$. The set of all symmetries of a CSP form a *group*: that is, they are a collection of bijections from the set of all literals to itself that is closed under composition of mappings and under inversion.

Definition 5. *Let G be a group acting on the set Ω . The stabiliser of an element $\omega \in \Omega$ is the set $\{g \in G : \omega^g = \omega\}$. This set is a subgroup of G . The orbit of an element $\omega \in \Omega$ is the set $\{\omega^g : g \in G\}$.*

The orbit of a literal $(X = \alpha)$, denoted $(X = \alpha)^G$, is the set of all literals that can be mapped to $(X = \alpha)$ by a symmetry in G . The stabiliser of a literal $(X = \alpha)$ is the set of all symmetries in G that map $(X = \alpha)$ to itself. We denote

$$\text{Stab}_G(X = \alpha) = \{g \in G : (X = \alpha)^g = (X = \alpha)\}.$$

Given a collection ψ of literals, the *pointwise* stabiliser of ψ is the subgroup of G which stabilises each element of ψ individually. The *setwise* stabiliser of ψ is the subgroup of G that consists of symmetries mapping the set ψ to itself.

There is a general technique, called *lex-leader*, for generating constraints that break symmetries [7]. The idea is to choose for each equivalence class of assignments under the given symmetry group one assignment to be canonical. Then constraints are added before search starts which are satisfied only by canonical assignments. We select canonical assignments by choosing an ordering of the literals, and representing assignments as tuples under this literal ordering. Any permutation of literals g maps tuples to tuples, and the lexicographically least of these is our canonical assignment. This leads to the set of constraints

$$\forall g \in G, V \preceq_{\text{lex}} V^g$$

where G is the symmetry group of the CSP; V is the vector of the literals of the CSP; and \preceq_{lex} is the standard lexicographic ordering relation – for example $AD \preceq_{\text{lex}} BC$ iff either $A < B$ or $A = B$ and $D \leq C$. In theoretical terms, the ordering of the literals in a CSP is unimportant. Operationally, however, it is essential to order the literals such that the search-order is respected. This ensures that lex-leader constraints are violated early in search, leading to improved pruning of the search tree via early backtracking.

Definition 6. *Let P be a symmetry breaking technique for CSPs. P is sound if it does not rule out whole equivalence classes of solutions to a CSP.*

P is complete if it returns exactly one member of each equivalence class of solutions with respect to the symmetry group of the CSP.

Proposition 1. *Given total orderings on the variables and the values of a CSP L , adding lex-leader SB constraints to L is both sound and complete.*

Proof. The proof is given in [7]. □

Another simple way to perform the SB is to leave it as a post process. We test each solution if it is minimal with respect to the same lexicographic ordering as before. If it is, we keep it; if not, we reject it. As this approach uses the same definition for canonical assignments it even becomes possible to compare individual solutions from the two methods. However, this approach only becomes useful in practise when efficient minimality testing algorithms are available that decide for an individual solution whether it is minimal in its orbit. We use an implementation in GAP of the methods presented in [4].

Remark 2. The above discussion applies to the situation where *all* solutions are required for a given CSP. If only the first solution (if any) is sought, then SB is not always an important consideration. In this paper we are always concerned with finding each symmetrically distinct solution to every CSP posed.

2.4 The Structure of Finite Monoids

In this section we examine structural properties of finite monoids. We show that each monoid can be thought of as being composed of a group and a semigroup, and, conversely, that given any group and semigroup one can construct a monoid. We also give an enumeration formula for a certain type of monoids.

Proposition 2. *Let $\langle M, *, e \rangle$ be a finite monoid. Then $M = G \dot{\cup} S$ where G are the units and the elements of S form an ideal.*

This is the unique way to decompose M into two sets that are closed under the induced multiplication such that one forms a group and the other an ideal.

Proof. A proof for the first statement can be found in [8, Lemma 4.7]. We remind the reader that units are the invertible elements (and thus form a group); and that an ideal is a subset of elements, such that every product of elements from the monoid with one factor from the ideal is itself in the ideal (and thus forms a semigroup).

For the second statement consider a decomposition $M = G \dot{\cup} S$ of M with G a group and S an ideal. If one unit were to lie in the ideal it would follow that the identity and thus all elements lie in S and the group would be empty, a contradiction. Thus all units lie in G . On the other hand, putting any not invertible element into G would stop it from being a group. Thus the decomposition is unique. \square

Referring to this proposition we will speak about the decomposition of a monoid into its group part and semigroup part. Their importance for the structure of a monoid is underlined by the following result:

Proposition 3. *If M_1 and M_2 are equivalent monoids, then their group parts are isomorphic and their semigroup parts are equivalent.*

Moreover, let M be a monoid with group part G and semigroup part S , and let G' be a group isomorphic to G and S' a semigroup equivalent to S . Then there exists a monoid equivalent to M with decomposition $G' \dot{\cup} S'$.

Proof. Let σ be an isomorphism (resp. anti-isomorphism) from M_1 to M_2 . Every unit in M_1 is then mapped to a unit in M_2 . As every restriction of σ is still an isomorphism (resp. anti-isomorphism) the group and semigroup parts must be isomorphic (resp. anti-isomorphic). This shows both that the semigroup parts are equivalent and that the group parts are isomorphic (since in a group, an anti-isomorphism composed with taking inverses gives an isomorphism).

For the second statement consider two cases depending on whether the semigroups are isomorphic or anti-isomorphic. In both cases let $\sigma : G \rightarrow G'$ be an isomorphism. If then, in the first case, $\tau : S \rightarrow S'$ is an isomorphism define

$$\pi : M \rightarrow G' \cup S', m \mapsto \begin{cases} m^\sigma & \text{for } m \in G \\ m^\tau & \text{for } m \in S. \end{cases}$$

The multiplication on $G' \cup S'$ given by $x *' y = (x^{\pi^{-1}} * x^{\pi^{-1}})^\pi$ will then both respect the existing multiplication in G' and S' and ensure that π is an isomorphism from M to $G' \cup S'$. In the second case, if $\bar{\tau} : S \rightarrow S'$ is an anti-isomorphism, define

$$\bar{\pi} : M \rightarrow G' \cup S', m \mapsto \begin{cases} (m^{-1})^\sigma & \text{for } m \in G \\ m^{\bar{\tau}} & \text{for } m \in S \end{cases}$$

whence the same multiplication ensures that $\bar{\pi}$ is an anti-isomorphism from M to $G' \cup S'$. And in both cases $G' \dot{\cup} S'$ is the decomposition following the proof for the first statement. \square

As well as decomposing a given monoid into its group and semigroup parts, we can also construct a monoid from a group and a semigroup.

Lemma 1. *Given a group $\langle G, *_G, e, {}^{-1} \rangle$ and a semigroup $\langle S, *_S \rangle$ define a multiplication $*$ on $G \cup S$ by:*

$$x * y = \begin{cases} x *_G y & \text{for } x, y \in G \\ x & \text{for } x \in S, y \in G \\ y & \text{for } x \in G, y \in S \\ x *_S y & \text{for } x, y \in S \end{cases}$$

*Then $\langle G \cup S, *, e \rangle$ is a monoid with group part G and semigroup part S .*

Proof. That $\langle G \cup S, *, e \rangle$ is indeed a monoid can easily be checked from the definition. From the proof for Proposition 2 we know that the group part consists of the invertible elements. From the definition of $*$ it follows that these are precisely the elements from G . \square

We call a monoid constructible in the above way a *trivial-action monoid*.

Lemma 2. *Every monoid of order n with group part of sizes $1, n-1$ or n is a trivial-action monoid.*

Proof. Let M be a monoid with decomposition $G \dot{\cup} S$. It is true in general that $G \times S \rightarrow S, (g, s) \mapsto g * s$ defines a left action and $S \times G \rightarrow S, (s, g) \mapsto s * g$ defines a right action of G on S . If G is the trivial group both these actions have to be trivial as the identity element always acts trivially. If the order of G is $n-1$ there is only one possible image in S and the actions are therefore trivial. There is nothing to prove in the special case that M is already a group. \square

We now want to know the number of all non-equivalent monoids that can be obtained following the construction in Lemma 1.

Proposition 4. *Let n be a natural number, and let \mathcal{G}_i and \mathcal{S}_i denote the non-isomorphic groups and non-equivalent semigroups of order i respectively. The number of non-equivalent trivial-action monoids of order n is then given by*

$$\sum_{i=1}^n |\mathcal{G}_i| |\mathcal{S}_{n-i}|.$$

Proof. From the first part of Proposition 3 it follows that trivial-action monoids can only be equivalent if they have isomorphic group parts and equivalent semigroup parts. Counting the number of combinations it follows that the sum $\sum_{i=1}^n |\mathcal{G}_i| |\mathcal{S}_{n-i}|$ is a lower bound.

Let now $M_1 = G_1 \dot{\cup} S_1$ and $M_2 = G_2 \dot{\cup} S_2$ be two trivial-action monoids with isomorphic group and equivalent semigroup parts. The second part of Proposition 3 says that there exists a monoid equivalent to M_1 with decomposition $G_2 \dot{\cup} S_2$. This, like M_1 , has to be a trivial-action monoid and as such is uniquely defined from the group and semigroup part. Therefore it equals M_2 . It follows that $\sum_{i=1}^n |\mathcal{G}_i| |\mathcal{S}_{n-i}|$ is as well an upper bound, which concludes the proof. \square

3 Methodology

Our underlying methodology is to use GAP to answer algebraic questions related to monoids. These answers allow us to generate suitable constraints for Minion instances, and eliminate (anti-)isomorphic solutions. In operational terms, GAP is the master process with Minion acting as a black box to provide solutions to carefully formulated CSPs. In this section we will describe three approaches for obtaining families of CSPs such that solving the entire family leads to an exact enumeration of the number of monoids of order n . By storing the solutions we can fulfil our secondary aim of providing a database of monoids.

Each CSP family will consist of variations of a generic CSP $L(n)$ consisting of the n^2 variables $\Delta := \{A_{1,1}, A_{1,2}, \dots, A_{1,n}, A_{2,1}, \dots, A_{2,n}, \dots, A_{n,1}, \dots, A_{n,n}\}$ – i.e. each entry in an $n \times n$ table (defining a multiplication $*$ by $i * j = A_{i,j}$) – with each variable having domain $\{1, 2, \dots, n\}$. The constraints are:

1. $a * (b * c) = (a * b) * c$ for all $a, b, c \in \{1, 2, \dots, n\}$;
2. $\exists i \in \{1, 2, \dots, n\}$ such that the i th row = i th column = $[1, 2, \dots, n]$.

The first constraint is associativity, which in Minion is enforced using *element* constraints. The constraint *element*(*vector*, *i*, *val*) specifies that, in any solution, *vector*[*i*] = *val*. We add a new variable $T_{a,b,c}$ for each triple (a, b, c) . The pair of constraints

$$\textit{element}(\textit{row}(a), A_{b,c}, T_{a,b,c}) \textit{ and } \textit{element}(\textit{column}(c), A_{a,b}, T_{a,b,c})$$

then enforce associativity on the triple. Constraint 2 enforces the existence of an identity element.

As mentioned in Section 2.3, modelling a problem often introduces symmetries. In our case this happens because we represent a monoid by its multiplication table. For this we introduced identifiers, $\{1, \dots, n\}$, for the elements that are initially indistinguishable. Moreover, our model fixes the direction to read the multiplication off the table. The symmetries are then isomorphisms and anti-isomorphisms between monoids. In the generic model these are given by the group $S_n \times C_2$ acting on $n \times n$ tables. As described in the introduction, the bijections in S_n permute rows, columns and values of a table while the non-trivial element in C_2 indicates an anti-isomorphism (and thus transposes the table).

To find a single representative from every equivalence class we have to break these introduced symmetries. We therefore define a canonical table in every equivalence class, having the minimal vector of values with respect to the variable order $[A_{1,1}, A_{1,2}, \dots, A_{1,n}, A_{2,1}, \dots, A_{2,n}, \dots, A_{n,1}, \dots, A_{n,n}]$. Thus for the literals $(X = \alpha) \preceq_{\text{lex}} (Y = \beta)$ if either X comes earlier than Y in the vector or $\alpha \leq \beta$. We can then break all symmetries by posting lex-leader constraints, or by post-hoc minimal image testing, as described in Section 2.3.

To obtain the numbers of monoids of order n , it is sufficient to solve the generic CSP $L(n)$, using either of the two SB methods. For large n , this approach is impractical: the search space increases exponentially and the number of symmetries increases factorially with n .

The three approaches presented in the following try to circumvent this explosion in problem size in two ways: by splitting $L(n)$ into a family of instances (most of which having many fewer symmetries), and by the use of enumeration results to reduce the size of the search space.

3.1 Approach 1: The Structure of a Monoid

Table 2: Construction of an instance $L_{G,S}$ from a group and a semigroup

	$m + 1$	$m + 2$	\dots	n
Group Table	*	*	\dots	*
	\vdots	\vdots		\vdots
	*	*	\dots	*
$m + 1$	*	\dots	*	
$m + 2$	*	\dots	*	
\vdots	\vdots		\vdots	
\vdots	\vdots		\vdots	
n	*	\dots	*	
	Semigroup Table			

Legend: Group Table is the multiplication table for a group of order m , with underlying set $\{1, \dots, m\}$. Semigroup Table is the multiplication table for a semigroup of order $n - m$, with underlying set $\{m + 1, \dots, n\}$. * denotes table entries not fixed before search.

Our first approach to splitting the CSP $L(n)$ into a family of instances $\mathcal{L}_1(n)$ involves using as much of the structure of finite monoids – from Section 2.4 – as possible. For a group $G = \langle \{1, 2, \dots, m\}, *_G, 1,^{-1} \rangle$, and a semigroup $S = \langle \{m + 1, m + 2, \dots, n\}, *_S \rangle$ we define $L_{G,S}$ to be a CSP instance having the same set of variables as $L(n)$ with the following constraints:

1. $a * (b * c) = (a * b) * c$ for all $a, b, c \in \{1, 2, \dots, n\}$;
2. $A_{i,j} = i *_G j$ for $1 \leq i, j \leq m$ – i.e. the group part is fixed in the top left part of the table;
3. $A_{i,j} = i *_S j$ for $m + 1 \leq i, j \leq n$ – i.e. the semigroup part is fixed in the bottom right part of the table;
4. all other variables $A_{i,j}$ have domain $\{m + 1, m + 2, \dots, n\}$;
5. the vectors $[A_{m+1,i}, A_{m+2,i}, \dots, A_{n,i}]$ and $[A_{i,m+1}, A_{i,m+2}, \dots, A_{i,n}]$ take distinct values, for all $1 \leq i \leq m$;
6. at least one of these vectors is not equal to $[m + 1, m + 2, \dots, m]$;
7. the first row and column is equal to $[1, 2, \dots, n]$.

Define $\mathcal{L}_1(n)$ to be the family of instances containing $L_{G,S}$ for all pairs of non-isomorphic groups and non-equivalent semigroups such that $|G| + |S| = n$. Table 2 gives an idea of the restrictions on the tables searched for in $L_{G,S}$.

Lemma 3. *The solutions in $L_{G,S}$ are monoids having group part G and semi-group part S .*

Proof. Constraint 1 guarantees associativity and constraint 7 the existence of an identity element. Thus all solutions are monoids. From constraint 4 it follows that S forms an ideal in every solution. As, by assumption, G is a group $G \dot{\cup} S$ must be the unique decomposition from Proposition 2. \square

This lemma shows that constraint 5 is an implied constraint: the units must have all different entries in their row and column in order to be invertible.

Let $l_{G,S}$ be the number of non-equivalent solutions of $L_{G,S}$.

Theorem 1. *The number of non-equivalent monoids of order n is given by*

$$\sum_{i=1}^n |\mathcal{G}_i| |\mathcal{S}_{n-i}| + \sum_{i=1}^n \sum_{G \in \mathcal{G}_i} \sum_{S \in \mathcal{S}_{n-i}} l_{G,S}.$$

Proof. From Proposition 4 we know that the first summand equals the number of trivial-action monoids of order n .

From Lemma 3 and Proposition 3 it follows that the sets of solutions for different instances are disjoint. Constraint 5 ensures that none of the solutions is a trivial-action monoid. On the other hand if M is a non-trivial-action monoid of order n , with decomposition $G \dot{\cup} S$, then there is an instance $L_{G',S'}$ with G' isomorphic to G and S' equivalent to S . Proposition 3 shows that there exists a monoid M' equivalent to M which is a solution of $L_{G',S'}$. \square

Remark 3. From Lemma 2 we know that $l_{G,S} = 0$ if the group size is 1, $n - 1$ or n . Therefore these instances need not be computed.

This approach has the obvious advantage that the search space is greatly reduced: many of the variables are fixed before search, and the domains of the remaining variables are reduced. However, the number of instances is too large for this approach to be practicable: for $n = 10$ and $m = 2$ there is one instance for each non-equivalent semigroup of order 8, and there are 1,843,120,128 of these.

Specifying 1 through m to be the units makes the elements in $\{1, \dots, m\}$ distinguishable from the remaining elements. Thus the symmetry introduced by this model is $S_{\{1, \dots, m\}} \times S_{\{m+1, \dots, n\}} \times C_2$. For the individual instances fixing the entries in the group part and the semigroup part of the multiplication table cuts down the size of the symmetry group: only elements in $S_{\{1, \dots, m\}} \times S_{\{m+1, \dots, n\}} \times C_2$ that stabilise the set of literals corresponding to the fixed group and semigroup parts of the table setwise will map solutions to solutions.

3.2 Approach 2: Restricted Diagonals

We start again from the basic model $L(n)$ and first notice that it does not make use of the fact that the identity element is distinguishable from all the other

elements. To reduce the number of symmetries we restrict our search to monoids that have 1 as identity element.

From the computer search for semigroups we adopt an idea that splits $L(n)$ in a family of instances without resulting in an unmanageable number of them. The idea is to fix the diagonal entries and consider no two equivalent diagonals. This idea was first introduced in computer search for semigroups of order 6 [9] (and subsequently used in the respective problem of order 8 [10]). Observe that every diagonal entry is mapped to a diagonal entry under the action on the table described in Section 1. This yields an induced action on the diagonals and therefore induced equivalence classes of diagonals. Note that fixing 1 to be the identity will cause the equivalence classes of diagonals to differ from the ones in [9, 10]. Our aim is to construct exactly one diagonal from each equivalence class.

Algorithm 1 Construct the connected digraphs with N vertices and a K cycle

Require: $K \leq N$ { cycle has not more than all vertices }

```

1:  $C \leftarrow \emptyset$ 
2: for all  $p$  in PARTITIONS( $N, K$ ) do {the partition specifies the sizes of rooted trees
   at the vertices of the cycle}
3:    $F \leftarrow$  FORESTS( $p$ )
4:   for  $f \in F$  do
5:      $\hat{f} \leftarrow$  set of all arrangements of the elements of  $f$ 
6:     for  $orbit$  in ORBITS( $\langle(1, 2, \dots, K)\rangle, \hat{f}$ ) do {the set of arrangements forms or-
   bits under the cyclic group}
7:        $rep \leftarrow$  representative of  $orbit$  {arbitrary element in the orbit}
8:        $D \leftarrow$  directed cycle of length  $K$ 
9:       for  $i \in \{1, 2, \dots, K\}$  do
10:         $D \leftarrow D$  merged with  $rep_i$  joined at vertex  $i$  {tree at position  $i$ }
11:       end for
12:        $C \leftarrow C \cup \{D\}$ 
13:     end for
14:   end for
15: end for
16: return  $C$ 

```

Proposition 5. *Let C be the family of directed, unlabelled graphs having $n - 1$ vertices, such that the outward degree of any vertex is less than or equal to 1 (and loops are allowed). Then C corresponds to the set of equivalence classes of diagonals with 1 in first position under the full symmetry group on $\{2, \dots, n\}$.*

Proof. For any $c \in C$, every labelling of vertices from $\{2, \dots, n\}$ leads to a diagonal in the following way: the first entry is 1, the entry in position $2 \leq k \leq n$ is the endpoint of the edge from vertex k , and 1 if no such edge exists. Two distinct labellings of c give two diagonals in the same equivalence class, since a re-labelling of c is simply a permutation of elements from $\{2, \dots, n\}$, and hence is an element of the group acting on the diagonals. For any diagonal we can

construct a labelled graph with vertices $\{2, \dots, n\}$, and a directed edge from vertex $2 \leq k \leq n$ to the vertex given by the k th entry of the diagonal, unless the entry is 1. The unlabelled graph is in C . \square

Detailed information regarding the connected components of the members of C is given in [11, 3.4]. They can either be rooted trees with the direction of edges towards the root, or they can be a directed cycle (of length one or more) where every vertex in the cycle is the root of a tree. Algorithm 1 constructs and returns the latter. The former is constructed recursively using the one-one correspondence between rooted trees on n vertices, and forests of rooted trees on $n - 1$ vertices. All sets of forests whose tree sizes are specified by a partition p of $n - 1$ are returned by our function FORESTS(p).

For a given diagonal D we define a CSP L_D with the same variables as $L(n)$ and the following constraints:

1. $a * (b * c) = (a * b) * c$ for all $a, b, c \in \{1, 2, \dots, n\}$;
2. the first row and column is fixed to $[1, 2, \dots, n]$;
3. the diagonal is fixed to D ;
4. there exists an $A_{i,j}$ with value 1 for $2 \leq i, j \leq n$.

Table 3 gives an idea of the restrictions on the tables searched for in L_D .

Table 3: Construction of an instance L_D by fixing the diagonal and identity

1	2	3	\dots	n
2	D_2	*	\dots	*
3	*	D_3	\dots	*
\vdots	\vdots	\vdots	\ddots	\vdots
n	*	*	\dots	D_n

Legend: D_i is the i -th entry in the diagonal; * denotes table entries not fixed before search.

Define $\mathcal{L}_2(n)$ to be the family of instances containing L_D for non-equivalent diagonals D of length n with 1 in first position. Let l_D be the number of non-equivalent solutions in L_D .

Theorem 2. *The number of non-equivalent monoids of order n is given by*

$$|\mathcal{S}_{n-1}| + \sum_D l_D.$$

Proof. The solutions in L_D are monoids having a non-trivial group part, since constraint 4 forces the existence of at least one unit that is not the identity. From

Lemmas 1 and 2 we know that \mathcal{S}_{n-1} is the number of non-equivalent monoids with trivial group part.

Let M be an arbitrary monoid on $\{1, 2, \dots, n\}$ with non-trivial group part and 1 as identity element. If there is no instance L_D for the diagonal D of M , then there must be a permutation $\sigma \in S_{\{2, \dots, n\}}$ such that L_{D^σ} is in $\mathcal{L}_2(n)$. Hence M^σ , a monoid equivalent to M , is a solution of this instance.

If the identity element of M is $e \neq 1$ then $M^{(1,e)}$ is a monoid isomorphic to M with 1 as identity and the above consideration applies to this. \square

Remark 4. We have computed and stored all non-equivalent semigroups for orders n up to 8 [12], using a similar combination of GAP and Minion. We do not report these calculations in detail, since the correct values have already been published.

The family $\mathcal{L}_2(n)$ can be further refined and optimised; details are given in [2].

Specifying 1 to be the identity element makes 1 distinguishable from the other elements. Thus the symmetry introduced by this model is $S_{\{2, \dots, n\}} \times C_2$, already a factor n smaller than for the generic model. Furthermore, a lot of this symmetry is broken by fixing the diagonal entries. Only elements in the stabiliser of the diagonal inside $S_{\{2, \dots, n\}} \times C_2$ will map every solution table to another solution. For many diagonals, the size of this group will be small (i.e. less than a few hundred), indicating that posting lex-leader constraints on images of literals will be efficient. Certain diagonals have large stabilisers, and here post-hoc symmetry breaking may work better. We can refine post-hoc SB by posting additional constraints that are guaranteed not to lose solutions. We define the *signature* of a solution table to be the n -tuple containing the number of occurrences of value k , in position k , for $k = 1, \dots, n$. We use signatures to break more symmetries by posting constraints prior to search. Therefore we analyse the directed graph associated with each diagonal. There are two cases to consider. The first is to identify completely interchangeable vertices in rooted trees. We first look for roots of isomorphic trees, and then recurse down each tree. If we identify symmetric vertices, we post a linear ordering on the signature list of the labels of the vertices: we restrict the numbers of occurrences of these values in any solution. Each level in the rooted tree structure can provide zero or more such constraints. The second is to break any cyclic symmetry by fixing a minimal vertex in a cycle. Again, we recurse through the structure of the graph to identify any symmetry at a given level, posting linear ordering constraints whenever symmetries are found. We use the built-in methods for orbits and stabilisers in GAP to find sets of equivalent values. At each level of the recursion we stabilise the vertices of the levels above.

Proposition 6. *Posting these ordering constraints on signatures is sound.*

Proof. If L_D is empty there are no solutions to lose. Let $M \in L_D$ be a solution of the original problem. If M violates a constraint on the first level then there is a symmetry $g \in \text{Stab}(D)$ – the stabiliser of the diagonal – such that M^g satisfies the constraint. Assume inductively that M satisfies all constraints up to level

$l - 1$. If M violates a constraint on level l then there is a symmetry $g \in \text{Stab}(D)$ which fixes all the constraints of the levels above l such that S^g satisfies the constraint on level l . \square

3.3 Approach 3: Fix Group Part and Diagonal

Table 4: Construction of an instance $L_{G,D}$ from a group and a semigroup diagonal

		$m + 1$	$m + 2$	\cdots	n
Group Table		*	*	\cdots	*
		\vdots	\vdots		\vdots
		*	*	\cdots	*
$m + 1$	* \cdots *	D_{m+1}	*	\cdots	*
$m + 2$	* \cdots *	*	D_{m+2}	*	*
\vdots	\vdots	*	*	\ddots	*
n	* \cdots *	*	\cdots	*	D_n

Legend: Group Table is the multiplication table for a group of order m , with underlying set $\{1, \dots, m\}$. D_i is the $(i - m)$ -th entry of the semigroup diagonal, with m added to this value to ensure that values $\{1, 2, \dots, m\}$ appear only in the group part. * denotes table entries not fixed before search.

We now consider an approach that combines the restrictions on variables and domains of approach 1, with the reduced number of diagonal instances of approach 2. The idea is to fix the group part to be the upper left corner of a solution table, fix the identity to be element 1, and fix the semigroup part of the diagonal to be a canonical representative of an equivalence class of diagonals. Table 4 gives a schema of a resulting CSP instance.

For a group $G = (\{1, 2, \dots, m\}, *_G, 1, ^{-1})$ and a semigroup diagonal D on $\{1, 2, \dots, n - m\}$ we define $L_{G,D}$ to be a CSP instance having the same set of variables as $L(n)$ with the following constraints:

1. $a * (b * c) = (a * b) * c$ for all $a, b, c \in \{1, 2, \dots, n\}$;
2. $A_{i,j} = i *_G j$ for $1 \leq i, j \leq m$ - i.e. the group part is fixed in the top left part of the table;
3. $A_{i,i} = D_i + m$ for $m + 1 \leq i \leq n$ - i.e the remaining diagonal entries are fixed;
4. all other variables $A_{i,j}$ have domain $\{m + 1, m + 2, \dots, n\}$;
5. the vectors $[A_{m+1,i}, A_{m+2,i}, \dots, A_{n,i}]$ and $[A_{i,m+1}, A_{i,m+2}, \dots, A_{i,n}]$ take distinct values, for all $1 \leq i \leq m$;
6. at least one of these vectors is not equal to $[m + 1, m + 2, \dots, m]$;
7. the first row and column is equal to $[1, 2, \dots, n]$.

Define $\mathcal{L}_3(n)$ to be the family of instances containing $L_{G,D}$ for all pairs of non-isomorphic groups and non-equivalent semigroup diagonals such that $|G| + |D| = n$.

Lemma 4. *The solutions in $L_{G,D}$ are monoids having group part G .*

Proof. The proof is identical to that of Lemma 3, with the set $\{m + 1, \dots, n\}$ playing the role of S . \square

Let $l_{G,D}$ be the number of non-equivalent solutions of $L_{G,D}$.

Theorem 3. *The number of non-equivalent monoids of order n is given by*

$$\sum_{i=1}^n |\mathcal{G}_i| |\mathcal{S}_{n-i}| + \sum_{i=1}^n \sum_{G \in \mathcal{G}_i} \sum_{|D|=n-i} l_{G,D}.$$

Proof. Non-equivalent diagonals leave to non-equivalent semigroups as completion of the respective table part. On the other hand there are no restrictions on the semigroup part of a solution of $L_{G,D}$ besides having diagonal D . With this in mind the proof follows the same ideas as that of Theorem 1. \square

Again, from Lemma 2 we know that $l_{G,D} = 0$ if the group size is 1, $n - 1$ or n , and therefore these instances need not be computed.

Specifying 1 through m to be the units makes the elements in $\{1, \dots, m\}$ distinguishable from the remaining elements. Thus the symmetry introduced by this model is $S_{\{1, \dots, m\}} \times S_{\{m+1, \dots, n\}} \times C_2$. For an individual instance, fixing the entries in the group part and the semigroup diagonal of the multiplication table cuts down the size of the symmetry group: only elements in $S_{\{1, \dots, m\}} \times S_{\{m+1, \dots, n\}} \times C_2$ that stabilise the set of literals corresponding to the fixed group and semigroup diagonal of the table setwise will map solutions to solutions.

4 Results

As discussed in Section 3.3, the family of CSPs $\mathcal{L}_1(n)$ was not implemented due to the large number of inequivalent semigroups. Our results for solving the instances in families $\mathcal{L}_2(n)$ are given in Table 5. The isomorph rejection approach suffers from the time required for GAP to decide whether a returned solution is minimal in the stabiliser of the diagonal for the instance. Many diagonals lead to no solutions, whereas other diagonals lead to very many (hundreds of thousands) solutions, each of which has to be tested for minimality. On the other hand, complete symmetry breaking leads to problems with space. Although many stabilisers of diagonals are small, certain diagonals break very little table symmetry. In these cases the number of lex-constraints posted grows factorially with increasing n , leading to large Minion input files and consequently a large space overhead for the constraints that are active (and hence need to be checked for violation) during search. Clearly, both SB methods should return the same number of solutions, and this is the case for all values of n under consideration.

Table 5: Solutions & Timings for Approach 2: Restricted Diagonals

SB Method	n : Diagonals	5	6	7	8	9	10
		27	81	242	699	2,026	5,823
Complete SB	$\mathcal{L}_2(n)$ solutions	30	213	1,757	22,956*	955,569	♠
	GAP cpu (s)	1	14	105	640	4,571	♠
	Minion cpu (s)	0	0	4	41	1,101	♠
	Total cpu (s)	1	14	109	681	5,672	♠
Isomorph rejection	$\mathcal{L}_2(n)$ solutions	30	213	1,757	22,956*	955,569	◇
	GAP cpu (s)	1	14	105	872	117,955	◇
	Minion cpu (s)	0	0	4	47	1,975	◇
	Total cpu (s)	1	14	109	919	119,930	◇

Legend: ♠ denotes spaceout. ◇ denotes timeout. * denotes that the number given for $L_2(8)$ includes the 5 groups of order 8 that were not searched for in our implementation (see [2] for details). All computations were performed using two quad-core 2.66 GHz Intel X-5430 processors, with 16 GB of RAM. GAP cpu times include all non-Minion computation.

The results for the families $\mathcal{L}_3(n)$ are given in Table 6. Only complete symmetry breaking was performed, since better times were obtained for the $\mathcal{L}_2(n)$ families using this approach. All total times are faster than for the respective $\mathcal{L}_2(n)$ families. In particular, solving for $n = 10$ is entirely practicable in terms of time, although significant memory is needed for certain instances.

Table 6: Solutions & Timings for Approach 3: Fixed Group Part and Diagonal

SB Method	n : Instances	5	6	7	8	9	10
		7	22	59	159	426	1,158
Complete SB	$\mathcal{L}_3(n)$ Solutions	5	58	428	5,539	101,082	9,269,715
	GAP cpu (s)	0	1	14	96	606	7,979
	Minion cpu (s)	0	0	0	6	139	40,294
	Total cpu (s)	0	1	14	112	745	48,273

Legend: All computations were performed using two quad-core 2.66 GHz Intel X-5430 processors, with 16 GB of RAM. GAP cpu times include all non-Minion computation.

The number of inequivalent monoids of orders 2 through 10 is given in Table 7. The numbers for orders 8 and 9 were published in [2]. The number for order 10 is a new result in algebraic combinatorics. Our secondary aim was to store the monoids of a given order n . For $n \leq 8$ these are available in the *smallsemi* GAP package [12]; for $n = 9$ we have the solutions, but have not yet incorporated

these into *smallsemi*; for $n = 10$ we have stored the solutions of $\mathcal{L}_3(10)$, but not the trivial-action monoids.

Table 7: Numbers of monoids up to (anti-)isomorphism

n	$ \mathcal{S}_{n-1} $	$\mathcal{L}_2(n)$ solutions	Monoids of order n	$\sum_{i=1}^n \mathcal{G}_i \mathcal{S}_{n-i} $	$\mathcal{L}_3(n)$ solutions
2	1	1	2	2	0
3	4	2	6	6	0
4	18	9	27	25	2
5	126	30	156	151	5
6	1,160	213	1,373	1,315	58
7	15,973	1,757	17,730	17,302	428
8	836,021	22,956	858,977	853,338	5,539
9	1,843,120,128	955,569	1,844,075,697	1,843,974,615	101,082
10			52,991,253,973,742[†]	52,991,244,704,027 [†]	9,269,715

Legend: The number of monoids of order n is the sum of the number of semi-groups of order $n - 1$ and the number of solutions found for all instances in the $\mathcal{L}_2(n)$ family of CSPs (see Theorem 2). It is also the sum of the $\mathcal{L}_3(n)$ solutions and number of trivial-action monoids of order n (see Theorem 3). [†] denotes the inclusion of 52,989,400,714,478 as the number of semigroups of order 9; this number has recently been obtained by the authors using similar methods, and the describing paper is in preparation.

5 Conclusions & Future Work

There is no known enumeration formula for all inequivalent monoids of order n . Therefore, in order to solve this problem, a search strategy is needed. We have combined careful group-theoretic GAP calculations with the speed and efficiency of the Minion CSP solver to obtain the numbers of monoids up to order 10. Two related but distinct families of CSPs were solved, with two related but distinct additions needed to get the total number of monoids for given n . For each method we have proved the correctness of a sum involving an enumeration formula and a family of CSP instances. We have also proved that no solution can be lost in any CSP instance, and that exactly one member of each equivalence class of solutions for any instance will be found. Both sets of solutions lead to the same number of monoids for $n = 2$ through 9.

Our most efficient approach involves the $\mathcal{L}_3(n)$ CSP family, in which the group part is fixed and the diagonal of the semigroup part is fixed. This approach gives the smallest number of instances of any of our models, retains much of the structure, and hence leads to smaller symmetry groups. Moreover, this approach benefits from the fact that nearly all monoids are trivial-action monoids.

5.1 Computational Details

In Section 4 we reported numbers of solutions and total times for each method and each value of n under consideration. As expected, the computational effort is not equally distributed across individual CSP instances. Many instances return no solutions, others very many. A few instances have large numbers of remaining symmetries, whilst most have very few. For $\mathcal{L}_3(n)$, the largest stabiliser occurs when the group is of order 2 and the remaining diagonal entries are $[3, 4, \dots, n]$. Here the stabiliser size is $2(n-2)!$, showing that stabilisers grow factorially with n . For $n = 10$, the resulting 80,640 lex-leader constraints can be dealt with using a compute server having 16 GB RAM, although some 80% of reported Minion time is taken solving this single instance. Nevertheless, Minion scales well with increasing problem size.

5.2 Future Work

The biggest obstacle to obtaining the number of monoids of order 11 is that the number of semigroups of order 10 is both unknown, and is likely to remain unknown for the foreseeable future. Despite this, the number of non-trivial-action monoids of order 11 can be considered independently. To solve this problem using adaptations of the techniques described in this paper, one would have to deal with the explosion in problem size described in Section 5.1. Using current computational resources, our complete SB methods are unlikely to be effective. Possible approaches include revisiting the equivalent $\mathcal{L}_1(11)$ model for those $\mathcal{L}_3(11)$ instances having large stabilisers, and/or deploying dynamic SB, for example GAP-Lex [13].

It is likely that our combined computational algebra and AI search technique can be used to enumerate other classes of algebraic structures that are associative.

Acknowledgements

Our work is supported by EPSRC grant EP/CS23229/1. We thank Ian Gent, Steve Linton, Victor Maltcev, James Mitchell & Nik Ruškuc for their help and comments.

References

1. Sloane, N.J.A.: The on-line encyclopedia of integer sequences. <http://www.research.att.com/~njas/sequences/Seis.html> (2008)
2. Distler, A., Kelsey, T.: The monoids of order eight and nine. In Autexier, S., Campbell, J., Rubio, J., Sorge, V., Suzuki, M., Wiedijk, F., eds.: Artificial Intelligence and Symbolic Computation, 8th International Conference, AISC 2008, Birmingham, July, 2004, Proceedings. Volume 5144 of Lecture Notes in Computer Science., Springer (2008) 61–76

3. The GAP Group: GAP – Groups, Algorithms, and Programming, Version 4.4.10. <http://www.gap-system.org> (2007)
4. Linton, S.: Finding the smallest image of a set. In: ISSAC '04: Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation, New York, NY, USA, ACM (2004) 229–234
5. Gent, I.P., Jefferson, C., Miguel, I.: Minion: A fast scalable constraint solver. In Brewka, G., Coradeschi, S., Perini, A., Traverso, P., eds.: ECAI, IOS Press (2006) 98–102
6. Cohen, D.A., Jeavons, P., Jefferson, C., Petrie, K.E., Smith, B.M.: Symmetry definitions for constraint satisfaction problems. *Constraints* **11**(2-3) (2006) 115–137
7. Crawford, J.M., Ginsberg, M.L., Luks, E.M., Roy, A.: Symmetry-breaking predicates for search problems. In: KR. (1996) 148–159
8. Grillet, P.A.: Semigroups. Volume 193 of Monographs and Textbooks in Pure and Applied Mathematics. Marcel Dekker Inc., New York (1995) An introduction to the structure theory.
9. Plemmons, R.J.: There are 15973 semigroups of order 6. *Math. Algorithms* **2** (1967) 2–17
10. Satoh, S., Yama, K., Tokizawa, M.: Semigroups of order 8. *Semigroup Forum* **49** (1994) 7–29
11. Harary, F., Palmer, E.M.: Graphical enumeration. Academic Press, New York (1973)
12. Distler, A., Mitchell, J.D.: smallsemi – a GAP package. <http://turnbull.mcs.st-and.ac.uk/~jamesm/smallsemi/> (2008)
13. Jefferson, C., Kelsey, T., Linton, S., Petrie, K.: Gaplex: Generalized static symmetry breaking. In Benhamou, F., Jussien, N., O’Sullivan, B., eds.: Trends in Constraint Programming. ISTE, London, UK (May 2007) 191–205