The Semigroups of Order 10

Andreas Distler[1], Chris Jefferson[2], Tom Kelsey[2], and Lars Kotthoff[2]

[1] Centro de Álgebra da Universidade de Lisboa,
1649-003 Lisboa, Portugal
[2] School of Computer Science,
University of St Andrews, KY16 9SX, UK

Abstract. The number of finite semigroups increases rapidly with the number of elements. Since existing counting formulae do not give the complete number of semigroups of given order up to equivalence, the remainder can only be found by careful search. We describe the use of mathematical results combined with distributed Constraint Satisfaction to show that the number of non-equivalent semigroups of order 10 is 12,418,001,077,381,302,684. This solves a previously open problem in Mathematics, and has directly led to improvements in Constraint Satisfaction technology.

Keywords: Constraint Satisfaction, Mathematics, semigroup, Minion, symmetry breaking, distributed search

1 Introduction

An important area of investigation is the determination of the number of solutions of a given finite algebraic problem. It is often the case that we are interested in the number of classes of solutions under some type of equivalence relation, since this gives the number of structural types rather than distinct objects. In certain cases, these numbers can be found by deriving counting formulae. It may also be possible, on an *ad hoc* basis, to derive enumerative constructions of larger objects from smaller ones. In both these cases, no systematic computer search is required – the numbers are calculated from mathematical proofs using the structures of the underlying problem.

There is no guarantee, of course, that the use of formally-proven formulae will work for all problems. It may be that no suitable formulae is available. In this event, the only method left is to carefully search for solutions, ensuring that none is missed and none is counted more than once. Examples include the search for all distinct transitive graphs on n vertices [23, 24], all binary self-dual codes of length 32 [2], all ordered trees with k leaves [32], and all non-equivalent semigroups up to order 9 and monoids up to order 10 [6, 7, 9, 17, 26, 28, 30]. Large-scale studies often involve a combination of enumeration by formula and computer search. The tautomer enumeration problem [18] from Cheminfomatics is an illustrative example. Commercial and academic software packages used to solve this type of problem typically use a suite of transformation

rules that allow enumeration without search, combined with generate-and-test searches for structures not predicted by the rulesets [25].

Whenever computer search is used to solve for types of solutions rather than absolute number of solutions, some method must be employed that ensures that exactly one canonical representative from each equivalence class is returned. This involves breaking the symmetries that allow objects from the same class to be interchanged, and the design and implementation of such methods is an important field of study in its own right [29, chapter 10].

A detailed exposition of search, symmetry-breaking, enumeration, and solution generation is given in [21]. The Constraint Satisfaction methods used in our search are described in [29], details of the specific Constraint Satisfaction solver used – Minion – are in [13], and the computational algebra package – GAP – used to identify and break symmetries is described at [11]. Basics of semigroup theory can be found in [15].

Table 1. Semigroup T of order 10.

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

A semigroup $T = (S, *)$ consists of a set of elements S and a binary operation $* : S \times S \to S$ that is *associative*, satisfying $(x * y) * z = x * (y * z)$ for each x, y, z in S. We call two semigroups $(S, *)$ and $(S', \circ)$ *isomorphic* – respectively *anti-isomorphic* – if there exists a bijection $\sigma : S \to S'$ such that $\sigma(x*y) = \sigma(x) \circ \sigma(y)$ – respectively $\sigma(x * y) = \sigma(y) \circ \sigma(x)$ – for all $x, y \in S$; in this case σ is an *isomorphism* respectively *anti-isomorphism*. An element x of T is an *idempotent* if $x * x = x$. The semigroup T is *nilpotent* if there exists an $r \in \mathbf{N}$ such that $|\{s_1 * s_2 * \cdots * s_r \mid s_1, s_2, \ldots, s_r \in S\}| = 1$, in other words if all products of r elements take the same value. If r is the least value for which this is true, then T is *nilpotent of degree* r. Table 1 is an illustrative example: $T = (\{0, \ldots, 9\}, *)$. By inspection, T is 7-idempotent, i.e. has exactly seven idempotents. T is not nilpotent of degree 3 since, for example, $4 * 5 * 6 = 4 \neq 5 = 2 * 3 * 4$. Given a permutation π of the elements of $\{0, \ldots, 9\}$, it is easy to check that a semigroup isomorphic to T is obtained by permuting the rows, the columns, and finally the

values according to π, and that additionally transposing yields the table of an anti-isomorphic semigroup.

The problem addressed in this paper is finding all ways of filling in a blank table such that multiplication is associative up to symmetric equivalence, i.e. up to isomorphism or anti-isomorphism.

In this paper we report that the hitherto unknown number of semigroups of order 10, up to equivalence, is 12,418,001,077,381,302,684. The size of the search space for this problem is 10^{100} with $2 \times 10!$ symmetries to be broken, making generate-and-test an intractable solution method. A recent advance in the theory of finite semigroups has led to a formula [8] that gives the number of 'almost all' semigroups of given order. Despite this, 718,981,858,383,872 non-equivalent solutions had to be found by a combination of *ad hoc* constructive enumeration proofs and Constraint Satisfaction search, which took 130 CPU years distributed across two local clusters and the Amazon cloud [1].

Our investigations have directly led to improvements in the Minion Constraint Satisfaction Problem solver. Watched constraints were introduced, and a more efficient lexicographic ordering constraint was implemented. These gave orders of magnitude improvements to our search, and have been included in subsequent releases of Minion.

We stress the interdisciplinary nature of our work: the result cannot – to our knowledge – be obtained by mathematical proof alone, nor can current AI search technologies hope to return the exact number of solutions with realistic resources.

In Section 2 we give detailed descriptions of our models and methods, and of the family of Constraint Satisfaction Problems (CSPs) that were solved to provide the main result. Section 3 contains the results divided into the subcases used to overcome computational bottlenecks. In Section 4 we discuss the improvements in CSP solving that we were able to identify and implement, together with a brief analysis of how our solver can often backtrack very early in search, and how distribution of the CSPs across multiple compute nodes is likely to generalise. We give some concluding remarks in Section 5.

2 Methods

We first describe a single CSP for our combinatorial problem (Sec. 2.1), that incorporates *a priori* knowledge regarding the number of semigroups of a certain type. The next stage is the replacement of this CSP by families of CSPs (Sec. 2.2), some of which are both computationally and mathematically difficult, and are solved by distributed search (Sec. 2.3). More computationally tractable families are solved using a single processor, and families having exploitable structure are solved mainly by constructive enumeration (Secs. 2.4, 2.5, and 2.6).

2.1 The single Constraint Satisfaction model

We model semigroups of order 10 as a CSP: a set of variables V each with a discrete and finite domain D of potential values, and a set of constraints C that

either forbid or require certain instantiations of variables by domain values. A full instantiation of the variables in V by values from D is a solution whenever no constraint is violated. We make extensive use of the *element* constraint on variables N, M_i and P having natural number domains

$$N = \langle M_0, \ldots, M_{n-1} \rangle [P]$$

which requires that N is the Pth element of the list $\langle M_0, \ldots, M_{n-1} \rangle$ in any solution. Propagators for this constraint are implemented in many CSP solvers, including Minion.

CSP 1 *Let* $V_1 = \{T_{a,b} \mid 0 \leq a, b \leq 9\}$ *be variables representing the entries in a* 10×10 *multiplication table* T*, and* $V_2 = \{A_{a,b,c} \mid 0 \leq a, b, c \leq 9\}$ *be variables representing each of the products of three elements. Our basic CSP has* $V = V_1 \cup V_2$ *as variables, each with domain* $D = \{0, \ldots, 9\}$*. For each triple* (a, b, c) *of values from* D*, we post the pair of constraints*

$$\langle T_{a,0}, \ldots, T_{a,9} \rangle [T_{b,c}] = A_{a,b,c} = \langle T_{0,c}, \ldots, T_{9,c} \rangle [T_{a,b}] \tag{1}$$

which enforce associativity.

Finding all solutions of CSP 1 would give the number of distinct semigroups of order 10, i.e. all non-identical associative 10×10 multiplication tables. Our task, however, is to search for the number of classes of solutions up to symmetric equivalence, i.e. up to isomorphism or anti-isomorphism. The symmetry group is the set of permutations of $\{0, \ldots, 9\}$ combined with possible transpositions of the tables, which we denote as $S_{10} \times C_2$ using standard group theoretic notation. Let $g = (\pi, \phi) \in S_{10} \times C_2$ be a symmetry and T be a multiplication table. T^g is then the table obtained by

1. permuting the rows and columns of T according to π;
2. permuting the values in T according to π;
3. either
   (a) doing nothing if ϕ is the identity element ϕ_1 of C_2;
   (b) transposing the table if ϕ is the non-identity ϕ_2 element of C_2.

Two multiplication tables T_1 and T_2 are isomorphic if $T_1 = T_2^{(\pi,\phi_1)}$ for some $\pi \in S_{10}$; T_1 and T_2 are anti-isomorphic if $T_1 = T_2^{(\pi,\phi_2)}$ for some $\pi \in S_{10}$. Since $S_{10} \times C_2$ is a group, the set of all multiplication tables can be partitioned into subsets of symmetric equivalents: those tables that are isomorphic or anti-isomorphic to each other form an equivalence class.

Our problem is to find the number of equivalence classes, either by formal enumeration proofs or by solving a variant of CSP 1 that returns exactly one canonical representative from each class. Our general search methodology is to post "lex-leader" symmetry-breaking constraints before search. This is a well-known technique for dealing with symmetries in CSPs [4], made harder to implement in our case because our symmetries involve both variables and values,

and made harder to deploy since we need to post $2 \times 10! = 7,257,600$ symmetry-breaking constraints.

To explain our realisation of "lex-leader" we first introduce of another way to describe a solution of CSP 1. A *literal* of a CSP $L = (V, D, C)$ is an element in the Cartesian product $V \times D$. Literals are denoted in the form $(x = k)$ with $x \in V$ and $k \in D$. An instantiation f corresponds to the set of literals $I_f = \{(x = f(x)) \mid f \text{ is defined on } x\}$, which uniquely defines f (but not every set of literals defines an instantiation). In particular, literals are mapped to literals under the isomorphic and anti-isomorphic transformations described above.

Given a fixed ordering $(\chi_1, \chi_2, \ldots, \chi_{|V_1||D|})$ of all literals in $V_1 \times D$, an instantiation can then be represented as a bit vector of length $|V_1||D| = 100 \times 10$. The bit in the i-th position is 1 if χ_i is contained in the instantiation and otherwise the bit is 0. The bit vector for the instantiation I_f corresponding to the ordering of the literals $(\chi_1, \chi_2, \ldots, \chi_{|V_1||D|})$ is denoted by $(\chi_1, \chi_2, \ldots, \chi_{|V_1||D|})_{I_f}$.

There is one solution in each set of symmetric equivalents for which the corresponding bit vector is lexicographic maximal, which we take to be the property identifying the canonical solution. We denote the standard lexicographic order on vectors by $\geq_{\text{lex}}$.

CSP 2 *Let (V, D, C) be as defined in CSP 1. We extend C by adding for all symmetries $g \in S_{10} \times C_2$ the constraint*

$$(\chi_1, \chi_2, \ldots, \chi_{|V_1||D|}) \geq_{lex} (\chi_1^g, \chi_2^g, \ldots, \chi_{|V_1||D|}^g). \tag{2}$$

The solutions of CSP 2 are canonical representatives of associative multiplication tables, as required.

It is not hard to show that all finite semigroups have at least one idempotent. Our symmetry-breaking constraints ensure that all solution tables will have the value 0 at $T_{0,0}$. Since we ensure that $0 * 0 * 0 = 0$, exactly those solutions that are nilpotent of degree at most 3 will have $a * b * c = 0$ for all values of a, b and c. A formula has recently been derived that gives the number of such solutions without search [8], so we add constraints that rule out these solutions from our CSP. We already have variables that represent each product of three elements, namely V_2.

CSP 3 *Let (V, D, C) be as defined in CSP 2. Form a vector containing the variables $\langle V_2 \rangle = \langle A_{a,b,c} \mid 0 \leq a, b, c \leq 9 \rangle$, and another vector $\langle Z \rangle$ consisting of 10^3 zeros. We post the additional constraint*

$$\langle V_2 \rangle \neq \langle Z \rangle \tag{3}$$

that guarantees that at least one triple product is non-zero in every solution.

Adding the number of solutions of CSP 3 to the formula value for semigroups of nilpotency degree at most 3 will give our full result. It should be noted that no enumeration formula for the solutions of CSP 3 exists: some form of organised search is required to solve the complete problem.

2.2 Case splits

CSP 3 has too many symmetries to be solved as a single entity. In practice, we apply a well-established technique that has been used in the enumeration of semigroups of orders 6 and 8 [28, 30] to subdivide the problem (CSP 4). For each subproblem we prescribe the entries on the diagonal of the multiplication table. There exist 10^{10} distinct diagonals, but only a small number appear in a canonical solution, so that far fewer subproblems are created. A detailed description of the efficient derivation of a set of diagonals containing all canonical ones is given in [5]. This approach heavily influences the symmetry-breaking, as any lex-leader constraint (2) is automatically fulfilled if it corresponds to a symmetry that does not fix the prescribed diagonal. We therefore significantly reduce the overall search space, and also the number of symmetry-breaking constraints needed for most subproblems.

CSP 4 *Let (V, D, C) be as defined in CSP 3, and let $E = \{E_a \mid 0 \leq a \leq 9\}$ be a canonical diagonal. We post the constraints*

$$T_{a,a} = E_a \text{ for } 0 \leq a \leq 9 \tag{4}$$

which prescribe the entries on the diagonal of any solution.

To further reduce the number of instances, we combine certain easy subproblems and prescribe only the number of idempotents, as done in [17]. For more difficult subproblems we do further easily-derived case splits which will be mentioned in Section 3. The choice of method used for a particular subproblem was based on our experience from searching for the semigroups of order 9.

2.3 Distributed CSP search

Our method of solving constraint problems in a distributed way does not require support for distributed architectures in the constraint solver. Instead, we partition and distribute the problem specification itself, with the different partitions of the search space solved independently. The advantage of this approach over techniques that require communication between the compute nodes is that its implementation and deployment are much simpler.

We partition by splitting on the values in the domain of a variable during search [20]. This is done as follows: first the solution process is stopped, then we compute *restart nogoods* as described in [22] and encode them as constraints that can be added to the original problem. When added, the new constraints enable the solver to restart search from where it was interrupted. In addition, we add constraints that split the remaining search space. We partition the domain for the variable currently under consideration into n pieces of roughly equal size. We then create n new models and to each in turn add the constraints ruling out the previously done search and $n-1$ partitions of that domain. As an example, consider the case $n = 2$. If the variable under consideration is x and its domain is $\{1, 2, 3, 4\}$, we generate 2 new models. One of them has the constraint $x \leq 2$

added and the other one $x \geq 3$. Thus, solving the first model will try the values 1 and 2 for x, whereas the second model will try 3 and 4.

It is impossible to predict reliably the size of the search space for each of the splits, and the time needed to search it. This directly affects the effectiveness of the splitting – if the search space is distributed unevenly across the splits, some of the compute nodes will be idle while the others do most of the work. We address this problem by repeatedly splitting the search space during search. In this way we create new units of work whenever a worker becomes idle by simply asking one of the busy workers to stop and generate split models. The search is then resumed from where it was stopped and the remaining search space is explored in parallel by the two workers. Note that there is a runtime overhead involved with stopping and resuming search because the constraints which enable resumption must be propagated and the solver needs to explore a small number of search nodes to get to the point where it was stopped before. There is also a memory overhead because the additional constraints need to be stored.

In practice, we run Minion for a specified amount of time, then stop, split and resume instead of splitting at the beginning and when workers become idle. Initially the utilisation of the workers is suboptimal because not enough work units have been generated. However, after some time the number of work units exceeds the number of workers and the utilisation reaches 100%, and the initial phase of under-utilised resources is negligible compared to the total computation time.

We used the distributed computing system Condor [31] to handle distribution of the work units to the worker nodes. Every time new models are generated, they are submitted to the system which queues them and allocates a worker as soon as one becomes available.

2.4 Construction of certain 2-idempotent semigroups

If $T = (\{1, \ldots, 9\}, *)$ is a 1-idempotent semigroup, 1 being the idempotent, we define four multiplications on $\{0, \ldots, 9\}$:

$$i *_{\mathrm{I}} j = \begin{cases} i * j & \text{if } i, j \in T \\ 0 & \text{otherwise} \end{cases} \qquad i *_{\mathrm{II}} j = \begin{cases} i * j & \text{if } i, j \in T \\ i & \text{if } j = 0 \\ j & \text{if } i = 0 \end{cases}$$

$$i *_{\mathrm{III}} j = \begin{cases} i * j & \text{if } i, j \in T \\ i * 1 & \text{if } i \in T, j = 0 \\ 1 * j & \text{if } j \in T, i = 0 \\ 0 & \text{if } i = j = 0 \end{cases} \qquad i *_{\mathrm{IV}} j = \begin{cases} i * j & \text{if } i, j \in T \\ 0 & \text{if } i = 0 \\ 1 & \text{if } i \neq 0, j = 0 \end{cases}$$

Denote $T_{\mathrm{I}} = (\{0, \ldots, 9\}, *_{\mathrm{I}}), \ldots, T_{\mathrm{IV}} = (\{0, \ldots, 9\}, *_{\mathrm{IV}})$. Then it can be readily verified that $T_{\mathrm{I}}, T_{\mathrm{II}}$, and T_{III} are 2-idempotent semigroups, and so is T_{IV} if the

element 1 is a left-zero in T, i.e. $1 * i = 1$ for all $i \in T$. These four semigroups are pairwise non-equivalent (except that $T_{\mathrm{II}} = T_{\mathrm{III}}$ if T is a group). Moreover, for two non-equivalent semigroups on $\{1, \ldots, 9\}$ the first three constructions lead to non-equivalent semigroups; and two non-isomorphic semigroups on $\{1, \ldots, 9\}$ lead to non-equivalent semigroups under the fourth construction.

We use the above constructions to reduce the search effort for certain 2-idempotent diagonals, with 0 and 1 being the idempotents. It can then be shown that $T_{0,1}, T_{1,0} \in \{0, 1\}$, and that the only solutions not arising from one of the constructions have $T_{0,1} = T_{1,0} = 1$. Hence we fix these variables of the CSP and in addition add constraints to rule out solutions that are of the form T_{II} or T_{III} for some semigroup T on $\{1, \ldots, 9\}$.

2.5 Construction from nilpotent semigroups of order 9

For a 1-idempotent semigroup $T = (\{0, \ldots, 8\}, *)$ with 0 being the idempotent we define a multiplication $\circ$ on $\{0, \ldots, 9\}$ as follows:

$$i \circ j = \begin{cases} i * j & \text{if } i, j \in T \\ 0 & \text{if } i = j = 9 \\ 9 & \text{otherwise} \end{cases}$$

If 0 is a zero element in T then $\circ$ is associative and we define the semigroup $T_\circ = (\{0, \ldots, 9\}, \circ)$. Two semigroups constructed in this manner are equivalent if and only if the two semigroups on $\{0, \ldots, 8\}$ are equivalent.

To rule out all semigroups that are equivalent to a constructed one from the CSP search for a given 1-idempotent diagonal, we cannot assume that 9 is the distinguished element. Every element whose diagonal entry equals the idempotent element 0, and does not appear on the diagonal itself, is a potential candidate. For each such element k we forbid that the entries in the k-th row and k-th column except the diagonal entry are all equal to k. That is we post the constraint

$$\langle K \rangle \neq \langle T_{0,k}, \ldots, T_{k-1,k}, T_{k+1,k}, \ldots, T_{9,k}, T_{k,0}, \ldots, T_{k,k-1}, T_{k,k+1}, \ldots, T_{k,9} \rangle, \quad (5)$$

where $\langle K \rangle$ is the vector of length 18 containing k in each position. It remains to justify that we do not miss any solution by posting these constraints. If T is a multiplication table for which equality holds in (5) for some k, then either T is equivalent to a semigroup as constructed above or there exists $T_{i,j} = k$ with $i, j \neq k$. This gives

$$k * (i * j) = k * k = 0 \text{ and } (k * i) * j = k * j = k$$

showing that such a table T is not associative.

2.6 Construction from nilpotent semigroups of degree at most 3

Let $T = (\{1, \ldots, k\}, *)$ be a nilpotent semigroup of degree at most 3 for some $2 \leq k \leq 9$ with 1 being its idempotent. The structure of T yields a natural

partition of $\{1,\dots,k\}$ into 3 sets: the zero element 1 by itself, the non-zero products, and the remaining elements (for details see [5, Section 2.1]). We denote the latter set by A^- and the set of non-zero products by B^+. The superscripts indicate a sign function, or multiplicative parity, that we introduce on $\{0,\dots,k\}$ with $\mathrm{sign}(0) = -$ and $\mathrm{sign}(1) = +$. We define a multiplication $*_\pm$ on $\{0,\dots,k\}$ as follows:

$$i*_\pm j = \begin{cases} i*j & \text{if } i,j \in T \text{ and } \mathrm{sign}(i) = \mathrm{sign}(j) \\ 1 & \text{if } i = 0, j \in A^- \text{ or } i \in A^-, j = 0 \text{ or } i = j = 0 \\ 0 & \text{if } \mathrm{sign}(i) \neq \mathrm{sign}(j) \end{cases}$$

Then $T_\pm = (\{0,\dots,k\}, *_\pm)$ is a semigroup as all products of three elements equal either 0 or 1 depending only on the parity of the elements in the product. In a second step we define a multiplication on the set $\{0,\dots,9\}$ based on $*_\pm$.

$$i \circ_\pm j = \begin{cases} i *_\pm j & \text{if } i,j \in T_\pm \\ 1 *_\pm j & \text{if } i \in T_\pm, j \geq k+1 \\ i *_\pm 1 & \text{if } i \geq k+1, j \in T_\pm \\ 1 & \text{if } i,j \geq k+1 \end{cases}$$

Two semigroups of order 10 constructed in this way are equivalent if and only if they arise from equivalent semigroups. None of the semigroups are nilpotent as they do not contain a zero and none are equivalent to those constructed in Section 2.5.

In a similar manner to the construction in Section 2.5, given a 1-idempotent diagonal, we have to add constraints for every potential candidate for the distinguished element from the above definition of $*_\pm$. The constraints are similar to (5) but are required for all vectors $\langle K \rangle$ whose entries are the idempotent and the candidate k as in the construction for some semigroup T. Note that this does not result in one constraint for every nilpotent semigroup of degree at most 3, but in one constraint for each of a few specific partitions into sets A^- and B^+ allowed by the diagonal.

In addition we have to search for semigroups not equivalent to one from the construction, in which the idempotent and a candidate for the distinguished element behave as in a construction. This reduces the domains for many table entries to a singleton, and requires one additional constraint that the vector of table entries that would be fixed by the construction does not equal the corresponding vector.

3 Results

For 4 to 7 and for 9 idempotents we added constraints that fixed the idempotents to CSP 3, and were able to solve as single instances on a single computer (Table

Idempotents	Semigroups	Method
3	219,587,421,825	Minion†
4	1,155,033,843	Minion
5	396,258,335	Minion
6	478,396,381	Minion
7	412,921,339	Minion
8	214,294,637	Minion†
9	60,036,717	Minion
10	7,033,090	Minion‡
Total:	222,311,396,167	

Table 2. Semigroups up to equivalence for the cases 3–10 idempotents. Minion† denotes specialised search using 289 sub-cases based on possible diagonals for 3 idempotents, and 4 sub-cases for 8 idempotents. The full methodology is described in [7]. Minion‡ denotes specialised search for bands as described in [5]. The computations took around 920 hours on a machine with 2.66GHz Intel X-5430 processor and 16GB RAM, giving roughly 67,000 solutions per second.

2). The 3- and 8-idempotent cases were solved by case-split into CSP 4 using the appropriate canonical diagonals. A k-idempotent semigroup of order k is known as a *band*, and there is extensive theory for these objects that can be used to refine search. This has been done in [5] for all bands up to order 10.

The 2-idempotent case is strictly harder: there are $2 \times 9!$ symmetries, and from our experience with semigroups of orders 1–9 we predicted about 4×10^{14} solutions. We therefore derived the constructions given in Section 2.4, which give the majority of such solutions without search. We still had to search for roughly 1.2×10^{14} solutions (Table 3).

Case	Semigroups	Method
Based on Section 2.4		
– up to equivalence	158,929,640,752,110	T_I, T_{II} and T_{III}
– up to isomorphism	105,945,136,997,613	T_{IV}
– the rest	226,006,150,622	Minion
Not based on Section 2.4	116,179,193,109,431	Distributed Minion
Total:	381,279,977,009,776	

Table 3. Semigroups up to equivalence having exactly 2 idempotents. The distributed Minion computation (involving machines with varying architectures) took 73 CPU years, returning an average 50,400 solutions per second. The CPU time taken to search for solutions not given by the constructions described in Section 2.4 was comparatively negligible. Methods T_I etc. denote the constructive multiplication defined in Section 2.4. 'The rest' denotes solutions for those instances that are not ruled out by the constraints that forbid the constructions.

The 1-idempotent case can be split naturally into two cases: solutions that are nilpotent of some degree (Table 4), and solutions that are not nilpotent (Table 5). There is exactly one canonical semigroup of nilpotency degree 2 (the zero semigroup) and one of degree 10 (the monogenic, nilpotent semigroup), and there is an enumeration formula for those of degree 3. The cases for degrees 5 through 9 are small enough to be solved by a single computer. The degree 4 case is again sub-divided into 316 canonical diagonals. For 68 pairs of instances, two distinct diagonals lead to CSPs that have the same search variables (the off-diagonal entries), symmetry group and constraints. In these cases we only solve the first CSP, and double the number of solutions to obtain the true value (Table 4). The final case, 1-idempotent but non-nilpotent semigroups, was approached via the constructions from smaller semigroups described in Sections 2.5 and 2.6, with limited CSP search for the remaining solutions (Table 5).

Nilpotency degree	Semigroups	Method
2	1	Zero semigroup
3	12,417,282,095,522,918,811	Formula
4		
– unique	49,304,583,445,962	Distributed Minion
– replicable	91,103,513,956,511	Distributed Minion
– replicas	91,103,513,956,511	No search
5	10,027,051,364	Minion
6	3,395,624	Minion
7	17,553	Minion
8	328	Minion
9	15	Minion
10	1	Monogenic semigroup
Total:	12,417,495,617,164,742,681	

Table 4. Semigroups up to equivalence having exactly 1 idempotent and being nilpotent of some degree. The distributed Minion computation (involving machines with varying architectures) took 60 CPU years, returning an average 74,000 solutions per second. The CPU times for the 5–9 degree cases were negligible by comparison.

3.1 Improved solver performance

As an integral part of the methodology development stage of this investigation, we performed careful profiling of Minion and compared performance against another CSP solver, Gecode [12]. Initial evidence was that Gecode was an order of magnitude faster than Minion on semigroup instances having a large number (over 20,000) of symmetry-breaking constraints, and that the solvers were competitive for other instances. Analysis showed that the speedup was primarily due to the fact that Gecode removes constraints from search once they become

entailed, whereas Minion would leave constraints in place throughout search. We revised Minion to do the same, resulting in two solvers that were broadly comparable in terms of performance.

Through profiling, we found that the revised Minion was spending over 95% of its time in the lexicographic ordering constraint for highly symmetric instances. We identified propagation calls that had a cost but almost always no effect, and as a result were able to design and implement the QuickLex algorithm [16], which looks at only two variables out of the whole constraint at once, leading to massive performance gains for lexicographic ordering constraints. On average, the cost of the QuickLex propagator is 15% of the cost of the hitherto optimal method of Frisch *et al.* [10].

The use of "watched literals" has led to remarkable improvements in SAT solvers, and an implementation for Constraint Satisfaction was proposed in [14]. The careful use of this technique allows efficient maintenance of generalised arc consistency on the element constraints that we use to enforce associativity (Section 2.1).

Our empirical experience for orders smaller than 10 is that the combined use of QuickLex and watched element constraints makes Minion over an order of magnitude faster than Gecode for this class of problems. Minion versions from 0.9 onwards have incorporated these enhancements. We report that solving semigroups appears to be a good stress test for constraint solvers, involving a small number of types of constraints, but large search spaces, large numbers of solutions and large numbers of symmetries.

Case	Semigroups	Method
Diagonal does not admit nilpotent of degree 3 solutions	3,673,835,659	Minion
Diagonal admits nilpotent of degree 3 solutions, and has a sub-diagonal that admits nilpotent of degree 3 solutions		
– construction from order 9	52,972,873,141,621	Construction 2.5
– the rest	12,596,375,843	Minion
– construction from orders up to 9	52,968,071,362,553	Construction 2.6
– the rest	712,828,694	Minion
Diagonal admits nilpotent of degree 3 solutions, and **does not** have a sub-diagonal that admits nilpotent of degree 3 solutions	609,690	Minion
Total:	105,957,928,154,060	

Table 5. Non-nilpotent semigroups up to equivalence having exactly 1 idempotent. The single CPU Minion calculations took about 110 hours, returning an average 44,700 solutions per second. 'The rest' denotes solutions for those instances that are not ruled out by the constraints that forbid the constructions.

4 Discussion

4.1 CSP search analysis

Minion is very successful at solving semigroup instances. We found that instances with no solutions were solved almost immediately, and problems with a large number of solutions had just over twice as many search nodes as solutions, approaching the minimum possible.

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

The displayed example shows how Minion performs early backtracks from a partially filled table when no solution exists. In our example, the diagonal entries have been set before search, and we have assigned $T_{0,1} = 7$. By simple inference from the element constraints, three variables are instantly instantiated (shown with values in italics in the figure). The element constraints in Minion can also remove individual values from the domains of variables. In particular, one of these constraints removes domain values for variable $T_{7,0}$(denoted as ⊞). This constraint gives:

$$\langle T_{7,0}, \ldots, T_{7,9}\rangle[T_{0,1}] = A_{7,0,1} = \langle T_{0,1}, \ldots, T_{9,1}\rangle[T_{7,0}] \tag{6}$$

$$\Rightarrow 7 * 7 = 6 = A_{7,0,1} = \langle 7, 0, T_{2,1}, \ldots, T_{9,1}\rangle[T_{7,0}] \tag{7}$$

$$\Rightarrow T_{7,0} \neq 0 \text{ and } \Rightarrow T_{7,0} \neq 1, \tag{8}$$

contradicting the associativity requirement $7{*}0 = (0{*}1){*}0 = 0{*}(1{*}0) = 0{*}7 = 0$. The value 7 was arbitrary; Minion will backtrack without further assignment for any value between 3 and 9, and search terminates after only 5 search nodes.

Whilst there are common classes of CSP symmetry that can be broken by posting a polynomially-sized subset of the symmetries – typically pure variable or pure value symmetries [3] – this does not seem to be one of those instances. This is because our symmetries permute both variables and values, and since the underlying group consists of all permutations, we have to post a constraint for each member of the subgroup of $S_{10} \times C_2$ determined by the case splits described in Section 2.2. Moreover, partial symmetry-breaking would lead to more than one solution per equivalence class, and we would then have to implement a potentially expensive maximal image post-process in order to obtain the correct result.

Minion has the further advantage that when symmetry-breaking or case-split constraints are added, the reasoning they generate is automatically combined with the reasoning of the associativity constraints, making it quick and easy to try out new ideas.

4.2 Distributed search

Using our knowledge of the numbers of smaller semigroups of various types, our *a priori* estimate of the number of solutions of CSP 3 was somewhere between 5×10^{14} and 1×10^{15}. Our development experience with semigroups of order 9 is that, on average, about 70,000 semigroups are found every second using a single compute node. Assuming a search rate of 50,000 solutions per second for order 10 – since the number and length of the constraints increase with increased order – this equates to between 317 and 634 years of wall clock time on a single machine. We therefore developed a distributed strategy.

Our approach works very well for the extremely large problem we are tackling here. The computations took several CPU decades, making negligible the cost of the under-utilisation of nodes in the first few hours of computation. Similarly, we found that the overheads incurred through splitting, restarting and propagating additional constraints were acceptable, because without the distribution across many computers, we would not have been able to solve the problem at all.

There are several advantages to implementing distributed solving in this way. First, by creating regular "snapshots" of the search done, the resilience against failures increases. Every time we stop, split and resume, our modified models are saved. As they contain constraints that rule out the search already done, we can only lose the work done after that point if a worker fails. This means that the maximum amount of work lost in case of a total failure of all workers is the allotted time T_{max} times the number of workers $|w|$. This is especially important for large-scale computations, and our experience shows this to be extremely useful in practice. The modified models can be stored. We exploited this by moving the solving process to a different set of workers, without losing any work. Since our methods require no communication between the individual workers solving the problem, they only need to be able to receive the problem sub-instances, and send either the solution or split models back. We used compute nodes in two local clusters and the Amazon cloud, and, since some parts of the search space required more memory than the machines initially chosen could provide, we were able to seamlessly move those parts of the computation onto more powerful machines.

Our leveraging of existing software to handle the logistics of distribution led to reductions in both development time and systematic errors. For large problems such as these, our experience is that the number of queued jobs will usually exceed the number of workers, ensuring good resource utilisation.

4.3 Validation

For most of the case-splits, we have run the solver exactly once. It is not inconceivable, therefore, that a miscalculation has occurred. For orders up to 8, the number of solutions is small enough that we can solve CSP 2 with diagonal case splits, using neither enumeration formulae nor constructions. For order 9 we can solve CSP 3, as a family of CSPs using no constructions. We have

performed these calculations multiple times on various architectures, using different choices for search heuristics and different implementations of constraint propagators. The expected totals are returned every time. For orders 7, 8 and 9 we have re-calculated using exactly the same case-splits described in this paper. Again, the computed totals match those in the literature for order 7 [17], 8 [30] and 9 [7]. These checks increase our confidence that (a) there are no systematic errors in our splitting of the problem into smaller instances, and (b) our code for identifying symmetries and solving CSPs is correct.

5 Conclusions

Counting semigroups up to equivalence is not easy, being in some sense near the worst point of the combinatoric tradeoff between counting by reasoning and counting by search. At one end of the scale, the number of all distinct 10×10 multiplication tables is trivial to derive: 100 entries each having one of 10 values gives 10^{100} solutions. This triviality is due to the complete lack of structure. As an example from the other end, finite groups are relatively easy to search for due to their higher level of structure, and there are far fewer of them. Semigroups occupy an intermediate zone, having a small amount of exploitable structure and a large number of solutions.

For many algebraic structures we break symmetries in order to deal with the combinatorial explosion in the number of trivially distinct objects. However for semigroups, the breaking of symmetries – which grow factorially with increasing order – still leaves a super-exponential growth in the number of non-equivalent solutions. The formula for finite semigroups that are nilpotent of degree three gives the vast majority of solutions, but, again, the remainder to be found by search grows super-exponentially as order increases.

It is relatively unusual for Constraint Satisfaction modelling and technology to produce new results in Mathematics. The only examples that we are aware of are new instances of graceful graphs [27] and the monoids of order 10 [7]. Finding the number of semigroups of order 10 has involved advances in both Constraint Satisfaction and abstract algebra. The mathematical constructions described in this paper rule out more than half the search needed and without the enumeration formula for nilpotent of degree 3 solutions the problem is effectively intractable using any known approach. Moreover, both the Constraint Satisfaction technology and the Mathematics are vital – semigroups 10 cannot be solved by researchers from either discipline alone.

Acknowledgments. Parts of the computational resources for this project were provided by an Amazon Web Services research grant. This work was developed within the project PTDC/MAT/101993/2008 of Centro de Álgebra da Universidade de Lisboa, financed by FCT and FEDER. TWK is supported by UK EPSRC grant EP/H004092/1. LK is supported by a SICSA studentship and an EPSRC fellowship.

References

1. Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/ (2008)
2. Bilous, R.T., Van Rees, G.H.J.: An enumeration of binary self-dual codes of length 32. Des. Codes Cryptography 26(1-3), 61–86 (Jun 2002), http://dx.doi.org/10.1023/A:1016544907275
3. Cohen, D.A., Jeavons, P., Jefferson, C., Petrie, K.E., Smith, B.M.: Symmetry definitions for constraint satisfaction problems. In: van Beek, P. (ed.) CP. Lecture Notes in Computer Science, vol. 3709, pp. 17–31. Springer (2005)
4. Crawford, J.M., Ginsberg, M.L., Luks, E.M., Roy, A.: Symmetry-breaking predicates for search problems. In: Aiello, L.C., Doyle, J., Shapiro, S. (eds.) KR'96: Principles of Knowledge Representation and Reasoning. pp. 148–159. Morgan Kaufmann, San Francisco, California (1996)
5. Distler, A.: Classification and Enumeration of Finite Semigroups. Shaker Verlag, Aachen (2010), *also* PhD thesis, University of St Andrews, 2010, http://hdl.handle.net/10023/945
6. 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. Lecture Notes in Computer Science, vol. 5144, pp. 61–76. Springer (2008)
7. Distler, A., Kelsey, T.: The monoids of orders eight, nine & ten. Ann. Math. Artif. Intell. 56(1), 3–21 (2009)
8. Distler, A., Mitchell, J.D.: The number of nilpotent semigroups of degree 3 (2012)
9. Forsythe, G.E.: SWAC computes 126 distinct semigroups of order 4. Proc. Amer. Math. Soc. 6, 443–447 (1955)
10. Frisch, A.M., Hnich, B., Kiziltan, Z., Miguel, I., Walsh, T.: Propagation algorithms for lexicographic ordering constraints. Artificial Intelligence 170, 834 (2006)
11. The GAP Group, (http://www.gap-system.org): GAP – Groups, Algorithms, and Programming, Version 4.4.12 (2008)
12. Gecode: Generic constraint development environment. http://www.gecode.org/
13. Gent, I.P., Jefferson, C., Miguel, I.: Minion: A fast scalable constraint solver. In: Brewka, G., Coradeschi, S., Perini, A., Traverso, P. (eds.) The European Conference on Artificial Intelligence 2006 (ECAI 06). pp. 98–102. IOS Press (2006)
14. Gent, I.P., Jefferson, C., Miguel, I.: Watched literals for constraint propagation in Minion. In: Benhamou, F. (ed.) CP. Lecture Notes in Computer Science, vol. 4204, pp. 182–197. Springer (2006)
15. Howie, J.M.: Fundamentals of semigroup theory, London Mathematical Society Monographs. New Series, vol. 12. The Clarendon Press Oxford University Press, New York (1995), oxford Science Publications
16. Jefferson, C.: Quicklex - a case study in implementing constraints with dynamic triggers. In: Proceedings of the ERCIM Workshop on Constraint Solving and Constraint Logic Programming. CSCLP'11 (2011)
17. Jürgensen, H., Wick, P.: Die Halbgruppen der Ordnungen ≤ 7. Semigroup Forum 14(1), 69–79 (1977)
18. Katritzky, A., Hall, C., El-Gendy, B., Draghici, B.: Tautomerism in drug discovery. Journal of Computer-Aided Molecular Design 24, 475–484 (2010), http://dx.doi.org/10.1007/s10822-010-9359-z, 10.1007/s10822-010-9359-z

19. Klee Jr., V.L.: The November meeting in Los Angeles. Bull. Amer. Math. Soc. 62(1), 13–23 (1956), http://dx.doi.org/10.1090/S0002-9904-1956-09973-2
20. Kotthoff, L., Moore, N.C.: Distributed solving through model splitting. In: 3rd Workshop on Techniques for implementing Constraint Programming Systems (TRICS). pp. 26–34 (2010)
21. Kreher, D., Stinson, D.: Combinatorial Algorithms: Generation, Enumeration, and Search. CRC Press (1998)
22. Lecoutre, C., Sais, L., Tabary, S., Vidal, V.: Nogood recording from restarts. In: Proceedings of the 20th International Joint Conference on Artifical Intelligence. pp. 131–136 (2007)
23. McKay, B.D.: Transitive graphs with fewer than twenty vertices. Math. Comp. 33(147), 1101–1121 (1979), contains microfiche supplement
24. McKay, B.D., Royle, G.F.: The transitive graphs with at most 26 vertices. Ars Combin. 30, 161–176 (1990)
25. Milletti, F., Storchi, L., Sforna, G., Cross, S., Cruciani, G.: Tautomer enumeration and stability prediction for virtual screening on large chemical databases. Journal of Chemical Information and Modeling 49(1), 68–75 (2009), http://pubs.acs.org/doi/abs/10.1021/ci800340j
26. Motzkin, T.S., Selfridge, J.L.: Semigroups of order five. presented in [19] (1955)
27. Petrie, K.E., Smith, B.M.: Symmetry breaking in graceful graphs. In: Rossi, F. (ed.) CP. Lecture Notes in Computer Science, vol. 2833, pp. 930–934. Springer (2003)
28. Plemmons, R.J.: There are 15973 semigroups of order 6. Math. Algorithms 2, 2–17 (1967)
29. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York, NY, USA (2006)
30. Satoh, S., Yama, K., Tokizawa, M.: Semigroups of order 8. Semigroup Forum 49(1), 7–29 (1994)
31. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: The Condor experience. Concurrency – Practice and Experience 17(2-4), 323–356 (2005)
32. Yamanaka, K., Otachi, Y., Nakano, S.I.: Efficient enumeration of ordered trees with k leaves (extended abstract). In: Proceedings of the 3rd International Workshop on Algorithms and Computation. pp. 141–150. WALCOM '09, Springer-Verlag, Berlin, Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-00202-1_13