International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 16, Number 2, pages 147–165, June 2026

Computing Embeddings and Isomorphisms of Finite Semigroups (extended version)[1]

James East
Centre for Research in Mathematics and Data Science,
Western Sydney University,
Sydney, Australia
0000-0001-6112-9754

Attila Egri-Nagy
Human & AI Center, Akita International University
Akita, Japan
https://orcid.org/0000-0001-7861-7076

Andrew R. Francis
School of Mathematics and Statistics,
University of New South Wales,
Kensington, Australia,
0000-0002-9938-3499

James D. Mitchell
Centre for Interdisciplinary Research in Computational Algebra,
University of St Andrews, St Andrews, Scotland
0000-0002-5489-1617

Received: February 18, 2026
Revised: May 4, 2026
Accepted: June 1, 2026
Communicated by Sayaka Kamei

Abstract

Semigroup theory is a branch of abstract algebra, and it provides mathematical tools for the theory of computation. Finite semigroups can describe state transition systems and thus they model physically realizable computers. Engineering questions like *What is the minimal number of states to realize a particular computation?* and *Which type of computation is more capable?* translate into the algebraic tasks of constructing isomorphisms and embeddings between semigroups of different representations. The underlying problem is (sub)graph isomorphism, which is computationally difficult in general. We describe variations of backtrack search algorithms that exploit the algebraic properties of semigroups, various improvements and optimizations. We carry out computational experiments to extend our algebraic knowledge. In particular, we report new computational results on transformation semigroups and on the more general family of diagram semigroups. We study the minimal degree representation problem, count distinct embeddings for

[1]This paper is an extended version of the paper with the same title presented at The Thirteenth International Symposium on Computing and Networking CANDAR 2025.

three types of diagram semigroups and work on an open problem of embedding into 2-generated subsemigroups.

Keywords: algebraic automata theory, minimal degree transformation representation of diagram semigroups, isomorphisms and embeddings, optimized backtrack search, computer algebra

1 Introduction

Semigroups are abstract algebraic structures with a single associative binary operation, often called multiplication or composition. Since only the associativity condition is required, semigroup elements come in a great variety (numbers, functions, matrices, relations, diagrams, etc.). Some models of computation, e.g., finite state automata, are based on the composition of total functions and thus automata are in essence transformation semigroups.

Deciding whether or not a semigroup S embeds into another semigroup T (T contains a copy of S), or whether they are isomorphic (S and T are essentially the same) is a non-trivial task. It can be reduced to the graph-isomorphism GI problem of undirected graphs [36], which is one of the leading problems in theoretical computer science. It is an open question whether GI can be decided in polynomial time or not [22]. The more general problem, the subgraph isomorphism problem, is NP-complete [7].

Interpreting semigroups as models of computation, the embedding problem is the question of whether one computer can emulate another one. This provides a framework for discussing engineering questions like *What is the minimal number of states needed to realize a computation?*, *How can we compare the computational power of different forms of computing?* [15]. This practical interest, combined with the inherent difficulty of the problem, justifies an experimental computational approach.

Here we describe a customized backtrack search algorithm that hugely improves the current computational capabilities in semigroup theory. By finding embeddings into full transformation semigroups efficiently, we can solve the minimal degree representation problem up to degree $n = 8$ (at least) and find all distinct embeddings. The computational implementation allowed us to solve open problems about embeddability into 2-generated subsemigroups.

In this section we give the basic definitions of semigroup theory and introduce diagram representations. Section 2 describes how the backtrack search algorithm can be customized for computing embeddings and deciding isomorphisms. Section 3 goes through techniques for improving the basic algorithm. Section 4 summarizes existing computational implementations and how the software packages developed here advance the state of the art. Section 5 is about applying the developed algorithms to problems about diagram semigroups.

1.1 Semigroups – Basic Definitions and Notation

Using the traditional mathematical approach, we first give the abstract definition of semigroups and then proceed to concrete representations. The state-based transformation representation is the most relevant in computer science, but in the next section we also discuss generalized diagram representations. The interplay between the abstract structure and its combinatorial representations is central to this paper. Utilizing features of concrete representations often allows more efficient algorithms, since we have more information about the semigroup elements and on how they can be composed.

1.1.1 Semigroups and Homomorphisms

A *semigroup* is a set S together with an associative binary operation $S \times S \to S$. We can denote the binary operation explicitly, like $s \cdot t$, or we can simply use concatenation st for denoting the product of semigroup elements $s, t \in S$.

A semigroup element e is an *idempotent* if $ee = e$. In other words, an idempotent is an operation that executed twice (or more times) yield the same effect as executing it once. This is an important property of several real-world processes, like pressing elevator button, or sending an online request.

A *subsemigroup* of S is a subset U of S that is closed under the operation, denoted by $U \leq S$. For $A \subseteq S$, $\langle A \rangle$ denotes the least subsemigroup of S containing A, the semigroup *generated* by A. In other words, $\langle A \rangle$ is

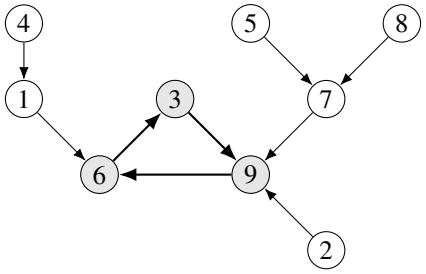

Figure 1: The graph of the transformation $[6, 9, 9, 1, 7, 3, 9, 7, 6]$. It has a single component. The cycle is highlighted.

the semigroup that arises from taking products of elements from A. If $\langle A \rangle = S$, then A is a *generating set* for S.

Given two semigroups $(S, \cdot)$ and $(T, \star)$, a *homomorphism* is a function $\varphi : S \to T$, such that $\varphi(u \cdot v) = \varphi(u) \star \varphi(v)$ for all $u, v \in S$. We call S the *source*, and T the *target* semigroup. An *isomorphism* is an invertible homomorphism; when an isomorphism exists we write $S \simeq T$. Isomorphic structures are essentially the same. An *embedding* is an injective homomorphism, denoted by $S \hookrightarrow T$. The image of an embedding of S into T is an isomorphic copy of S inside T.

1.1.2 Multiplication Tables

For a finite semigroup S with $|S| = n$, we fix an order of its elements $s_1, \ldots, s_n$, and refer to the elements by their indices. The *multiplication table*, or *Cayley table* of S is an $n \times n$ matrix $S = (S_{i,j})$ with entries from $\{1, \ldots, n\}$, such that $S_{i,j} = k$ if $s_i s_j = s_k$, and we can simply write $ij = k$. The ith row is $S_{i,*}$ and the jth column is $S_{*,j}$. A multiplication table is a complete description of a given semigroup: it contains the result of the multiplication for any pair of elements. However, it does not contain any internal details of the semigroup elements. It does not explain why $st = v$, it merely records this fact. That's why it is an *abstract representation*.

1.1.3 Transformation Semigroups

For a finite set X, a *transformation* is a function $f : X \to X$. We refer to the elements of X as *points* or as *states* when emphasizing the automata theory connection. In the finite case, when $|X| = n \in \mathbb{N}$, the standard labeling of states is done by the positive integers $\{1, \ldots, n\}$. A transformation t is denoted by simply listing the images of the points: $[t(1), t(2), \ldots, t(n)]$.

A transformation $t : X \to X$ can be represented as a directed graph, the so-called *functional digraph*, defined as $G(t) = (X, E)$, where the edge set E is $\{(x, t(x)) \mid x \in X\}$. In general, a functional digraph consists of several *components* (connected parts). A single component has a cycle with trees joining into the states in the cycle. For an example see Figure 1. In other words, we have a discrete dynamical system [18].

A *transformation semigroup* (X, S) of *degree* n is a collection S of transformations of an n-element set X, closed under function composition. The semigroup of all transformations of n points is the *full transformation semigroup* $\mathcal{T}_n$.

Transformations give a concrete representation of semigroups. From the perspective of automata theory, we can consider the set of states to be X and the semigroup elements the transitions of these states. In this way, semigroups are abstract representations of automata (without initial and accepting states).

1.1.4 Permutation Groups

Groups are semigroups with an identity (unique do-nothing operation) and with unique inverses for all elements. Groups model reversible, and semigroups model irreversible processes. The group consisting of all *permutations* (bijective transformations) of degree n is the *symmetric group* $\mathcal{S}_n$. For permutations we use

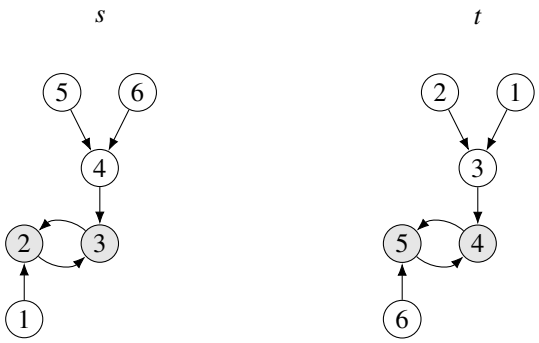

Figure 2: The two transformations, s and t, are essentially the same. We can verify this visually, since their graphs are drawn in identical layouts. As transformations they may look different: $s = [2, 3, 2, 3, 4, 4]$ and $t = [3, 3, 4, 5, 4, 5]$. However, we can find a conjugator $g = (1, 6)(2, 5)(3, 4)$ (in transformation notation $[6, 5, 4, 3, 2, 1]$) to relabel one to the other: $g^{-1}sg = t$. Note that we used the same canonical state set for both, thus the relabeling is not that apparent. We move between the sets of ordered states $1, 2, 3, 4, 5, 6$ and $6, 5, 4, 3, 2, 1$.

cycle notation. For instance, $[2, 3, 1]$ is written as $(1, 2, 3)$. The *cyclic* (one-generated) group of order n is denoted by $\mathbb{Z}_n$. In computing terms, $\mathbb{Z}_n$ is an n-counter.

1.1.5 Conjugation

We often need to decide whether two transformations are essentially the same or not. Intuitively, if we can get transformation t by relabeling the points in transformation s, then they are doing the same computation on different states. We could use s as substitute for t by changing the state set, performing s, and finally converting the results back. The definition of *conjugation* describes exactly this process.

Transformations $s, t \in \mathcal{T}_n$ are *conjugate* if there exists a permutation $g \in \mathcal{S}_n$ such that $t = g^{-1}sg$. We call such a g a *conjugator*. For an example see Figure 2.

Conjugation can be extended to sequences and sets of transformations. As conjugation is tied to symmetries, it can be used for other representations of semigroup elements. Symmetries enable compression, thus they are essential for efficient enumerations of combinatorial structures. Often we are interested in structures only up to conjugation.

1.2 Diagram Semigroups

From the point of view of theoretical computer science and automata theory, transformation semigroups are the most important representations of semigroups. However, they are just one special case of *diagram semigroups*, where elements can be composed by adjoining diagrams. These are described by fundamental mathematical objects such as relations and functions. The original interest came from algebras with a basis whose elements can be multiplied diagrammatically (e.g. [4,28]) and diagram semigroups can model various situations in theoretical physics [33].

Partitioned binary relations are the most general type of diagrams we consider, although historically it was the last to be defined [34]. They are directed graphs of binary relations, where the vertices are partitioned into two nonempty sets, the "interfaces" for diagram composition. Here we consider partitioning only into equal-sized parts. For a finite set X a *diagram* is a subset of $(X \cup X') \times (X \cup X')$ where $|X| = |X'| = n$, the *degree* of the diagram, and $X \cap X' = \emptyset$. Pictorially, we draw the points from X on an upper row with those from X' below, and we draw a directed edge $a \to b$ for each pair (a, b) from the diagram. The product $\alpha\beta$ of two diagrams α and β (on the same set X) is calculated as follows. We first modify α and β by changing every lower vertex x' of α and every upper vertex x of β to x''. We then stack these modified diagrams together with α above β so that the vertices x'' are identified in the middle row (there may now be parallel edges in this stacked graph). Finally, for each $a, b \in X \cup X'$ we include the edge $a \to b$ in $\alpha\beta$ if and only if there is a path

from a to b in the stacked graph (as defined above) for which the edges used in the path alternate between the edges of α and the edges of β. An example is given in Figure 3 (where, for convenience, the edges of β are white so that the kinds of paths referred to above are alternating in colour). This operation is associative, so the set of all diagrams on the set X forms a semigroup. When $X = \{1, 2, \ldots, n\}$, we denote this semigroup by $\mathcal{P}\mathrm{B}_n$. The identity element of $\mathcal{P}\mathrm{B}_n$ is the diagram containing the edges $x \to x'$ and $x' \to x$ for each x.

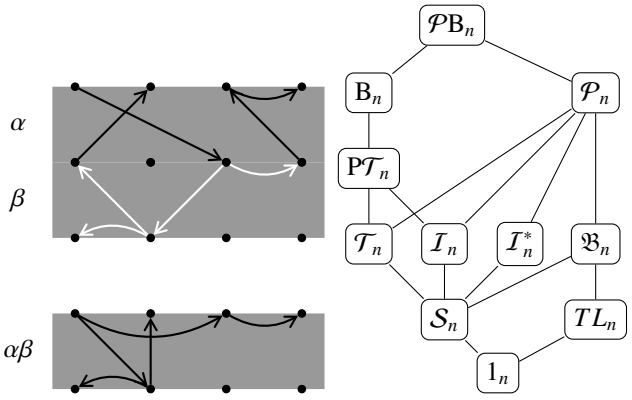

Figure 3: (left) Composing partitioned binary relations α and β. The arrows of the product are induced by paths of the stacked diagram with the property that the consecutive arrows have alternating colors. (right) The partial order of the different types of diagram semigroups of order n. The edges represent embeddings, not necessarily subsets. For instance, we represent $\mathcal{T}_n$ by total functions and not as a special subset of partitioned binary relations in $\mathcal{P}\mathrm{B}_n$.

Other diagram types can be defined by a set of constraints on the arrows of partitioned binary relation. The constraints can be about the direction of the arrows or the partition, limiting the number of arrows from a point, or requiring planarity (no crossing arrows) of the diagram. Some notable types are the following: B_n binary relations [39], $\mathrm{P}\mathcal{T}_n$ partial transformation semigroup [20], $\mathcal{P}_n$ (bi)partition monoid [24, 27, 33], $\mathfrak{B}_n$ Brauer monoid [4], $\mathcal{S}_n$ symmetric group [5, 9], $\mathcal{T}_n$ full transformation semigroup [20], $\mathcal{I}_n$ symmetric inverse monoid [31], $\mathcal{I}_n^*$ dual symmetric inverse monoid [19], and TL_n Temperley-Lieb monoid [23]. For the partial order of diagram semigroups of the same degree see Figure 3.

2 Partitioned Backtrack

Classical backtrack search provides a simple algorithm for constructing semigroup embeddings and isomorphisms, but ignoring algebraic information about the structures involved makes it inefficient. Therefore, we partition the elements of the target semigroup by their suitability for being homomorphic images for the elements of the source semigroup.

2.1 Algorithm for Embedding Multiplication Tables

Given semigroups S and T such that $|S| \leq |T|$, we would like to know if we can construct an embedding $S \hookrightarrow T$, and if so, actually construct one. Let $m = |S|$ and $n = |T|$, so we need a map $\varphi : \{1, \ldots, m\} \to \{1, \ldots, n\}$ such that φ is injective: $\varphi(i) = \varphi(j)$ implies $i = j$, and it is a homomorphism: $\varphi(i)\varphi(j) = \varphi(ij)$, where multiplication on the left hand side is in T and on the right hand side is in S.

Example 2.1. For permutation group $\mathbb{Z}_2 = \{(), (1, 2)\}$ and transformation semigroup $\mathcal{T}_2 = \{[1, 1], [1, 2], [2, 1], [2, 2]\}$, using lexicographic ordering of the elements, the maps $1 \mapsto 2$, $2 \mapsto 3$ give an embedding.

Figure 4: Example of a semigroup element with index-period (3,2). The third power s^3 is the least power that gets repeated, as $s^5 = s^3$. All higher powers alternate between s^3 and s^4 in a 2-cycle.

$\mathbb{Z}_2$	1	2
1	1	2
2	2	1

$\hookrightarrow$

$\mathcal{T}_2$	1	**2**	**3**	4
1	1	1	4	4
2	1	**2**	**3**	4
3	1	**3**	**2**	4
4	1	4	1	4

For finding embeddings, the *brute-force* algorithm goes through all possible maps by checking all possible assignments of the m elements of S to the n elements of T. There are $\frac{n!}{(n-m)!}$ such maps. By gradually exploiting more information about φ, S and T we can construct increasingly more efficient algorithms.

2.1.1 Classical Backtrack

We can observe that any partial map from S to T violating the compatibility condition of homomorphisms cannot be fixed by adding more mappings. A partial non-solution cannot be extended to a solution, therefore the classical *backtrack* method [30] can be applied.

Let p be a partial solution represented by a sequence of integers, such that the ith element is $\varphi(i)$. So, $p = (\varphi(1), \varphi(2), \ldots, \varphi(l))$ for some $l \leq |S|$. Being a homomorphism requires that if $ij = k$ (in S), then $\varphi(i)\varphi(j) = \varphi(k)$, (in T). However, k and $\varphi(k)$ may not be in the partial solution yet. If a product $ij = k$ is not in the domain of the partial map yet, i.e. $k > l$, then the product $\varphi(i)\varphi(j)$, should also be *undefined*, i.e. not being in the sequence p. In an implementation, using a more flexible data structure (e.g., hash table instead of list) allows us to immediately set $\varphi(i)\varphi(j)$ to be $\varphi(k)$.

Now assume that p could be a homomorphism. Then we extend its sequence by choosing a new $\varphi(l + 1)$ from the remaining elements of T and check whether the homomorphism property is true for products containing $l + 1$. We also have to check for previous undefined entries, since they may evaluate to $l + 1$ and thus become defined. If the above conditions are true, then we can continue extending p by $l + 2$. If not, then according to backtrack, we choose another value for $\varphi(l + 1)$, or if there is no such candidate, then going back to l and looking for alternative $\varphi(l)$, and so on.

2.1.2 Partitioned Backtrack

We can improve the search algorithm by reducing the number of choices available at each step. We precompute equivalence relations on S and T such that elements of a class in S may only be mapped by an embedding to an element of a single class of T. In other words, we classify elements of S and T by properties that are invariant under embeddings. Therefore, when trying to extend a partial solution, we need to look for candidates only in the corresponding class. These classes partition the search space, thus we call the method *partitioned backtrack search*.

For abstract semigroups, one such invariant is the semigroup generalization of the order of the element (isomorphism of monogenic semigroups). In a finite semigroup, taking the powers of an element will eventually have a repeated value.

Definition 2.2 (Index-period)**.** For an element a in a finite semigroup S, there exist integers $m \geq 1$ and $r \geq 1$ such that $a^{m+r} = a^m$. The *index* of a is the smallest such integer $m \geq 1$ (the exponent of the first repeated power), and the *period* of a is the smallest such integer $r \geq 1$ (the length of the cycle).

The index-period pair (m, r) is an invariant under isomorphisms. For instance, $(1, 1)$ means $a^2 = a$, i.e., a is an *idempotent*.

Figure 4 visualizes the definition of index-period: first there is a linear succession of powers of s that appear once only, then we hit a cycle. This orbit is the semigroup generated by a single element $S = \langle s \rangle$,

called *monogenic*, or *cyclic* semigroup. Permutations always have index 1, as bijections are reversible, the first power is bound to be repeated. However, not every transformation with index 1 is a permutation. As a trivial example, the identity transformation $[1, 2, 3]$ and the constant map $[1, 1, 1]$ both have the index-period pair $(1, 1)$.

The index-period pairs are invariant under isomorphisms, thus they give a way to classify candidate targets when constructing an isomorphism map.

Definition 2.3 ($\cong_{-\circ}$)**.** In a finite semigroup when two elements s and t have the same index-period pairs, we write $s \cong_{-\circ} t$, and $\cong_{-\circ}$ is an equivalence relation.

Partitioning by $\cong_{-\circ}$ can detect whether an embedding is possible or not: for each $A \in {}^{S}\!/_{\cong_{-\circ}}$ the corresponding class $B \in {}^{T}\!/_{\cong_{-\circ}}$ (with same index-period as A) should satisfy $|A| \leq |B|$. It also potentially reduces the search space. Denoting the class in ${}^{T}\!/_{\cong_{-\circ}}$ corresponding to the class $A \in {}^{S}\!/_{\cong_{-\circ}}$ by $\varphi(A)$, the size of the search space is given by

$$\prod_{A \in {}^{S}\!/_{\cong_{-\circ}}} \frac{|\varphi(A)|!}{(|\varphi(A)| - |A|)!},$$

showing that the efficiency depends on the number of the classes and their cardinalities.

Example 2.4. If $|S| = 5$ and $|T| = 10$ the search space size is $\frac{10!}{(10-5)!} = 30240$. Assuming that we can partition S into $\cong_{-\circ}$-classes of size $2, 3$ and T into corresponding classes of size $3, 5$ (and a class of size 2 with index-period not appearing in S), then the search space size is reduced to $\frac{3!}{(3-2)!} \cdot \frac{5!}{(5-3)!} = 360$.

After this theoretical example, the natural question is what is the size distribution of the index-period equivalence classes for semigroups. As an indicative answer, we check a full transformation semigroup. Table 1 shows the size distribution of the equivalence classes of the $7^7 = 823543$ transformations in $\mathcal{T}_7$. The smallest class has 420, the largest class has 126000 elements. Therefore, by itself the index-period is not a strong enough invariant: there are only a few classes, and their sizes are far from uniformly distributed.

2.2 Deciding Isomorphism of Multiplication Tables

For isomorphism we can define more invariants and a finer partitioning of elements by using a stronger equivalence relation. The invariant properties can be any isomorphism invariant of a multiplication table that do not take into account any information of the actual ordering of its elements.

A *frequency distribution* takes a multiset and enumerates its distinct elements paired with the number of occurrences of the elements. For instance, the frequency distribution of a row vector $[7, 2, 1, 2, 5, 5, 2, 7]$ is $[[1, 1], [2, 3], [5, 2], [7, 2]]$, meaning that 1 appears once, 2 appears three times and 5 and 7 twice. However, we cannot retain the element information as it depends on the sorting of the semigroup elements. We keep only the sorted frequency values $[1, 2, 2, 3]$. Sorting is crucial here to decide whether two distributions are the same or not. For example, the vector $[2, 4, 2, 4]$ has the same frequency distribution as $[1, 3, 3, 1]$ and $[2, 2, 4, 4]$, which is $[2, 2]$, but $[2, 4, 4, 4]$ has a different one, namely $[1, 3]$. We can also take the frequency distribution of frequency values, since it is derived from data containing no information on the ordering of the elements. We can define invariant properties both on the element and on the table level. In addition to the index-period values, the *element level invariants* are the number of occurrences of the element i

1. in the table, *frequency*,
2. in the diagonal of the table, *diagonal frequency*,
3. in its row $S_{i,*}$, *row frequency*,
4. in its column $S_{*,i}$, *column frequency*.

These invariants contain information about the structure of the semigroups (Green's classes). We put them together in a single aggregated data structure called the *element profile*, in order to get a finer equivalence relation.

$\mathcal{T}_7$

$(1,12)$	420	0.05 %
$(1,10)$	504	0.06 %
$(1,7)$	720	0.09 %
$(2,6)$	4200	0.51 %
$(4,3)$	5040	0.61 %
$(2,5)$	5040	0.61 %
$(3,4)$	5040	0.61 %
$(5,2)$	5040	0.61 %
$(6,1)$	5040	0.61 %
$(1,1)$	6322	0.77 %
$(1,5)$	19152	2.33 %
$(1,6)$	22050	2.68 %
$(5,1)$	27720	3.37 %
$(2,4)$	28980	3.52 %
$(3,3)$	29400	3.57 %
$(4,2)$	30240	3.67 %
$(1,4)$	37590	4.56 %
$(1,3)$	37800	4.59 %
$(1,2)$	45843	5.57 %
$(2,1)$	59472	7.22 %
$(4,1)$	68880	8.36 %
$(2,3)$	73080	8.87 %
$(3,2)$	85260	10.35 %
$(3,1)$	94710	11.50 %
$(2,2)$	126000	15.30 %

$\mathcal{S}_{10}$

$(1,1)$	1	0.00 %
$(1,2)$	9495	0.26 %
$(1,3)$	31040	0.86 %
$(1,5)$	78624	2.17 %
$(1,7)$	86400	2.38 %
$(1,30)$	120960	3.33 %
$(1,15)$	120960	3.33 %
$(1,21)$	172800	4.76 %
$(1,20)$	181440	5.00 %
$(1,4)$	209160	5.76 %
$(1,14)$	259200	7.14 %
$(1,12)$	403200	11.11 %
$(1,9)$	403200	11.11 %
$(1,8)$	453600	12.50 %
$(1,10)$	514080	14.17 %
$(1,6)$	584640	16.11 %

$\mathfrak{B}_7$

$(1,12)$	420	0.31 %
$(1,10)$	504	0.37 %
$(1,7)$	720	0.53 %
$(2,3)$	5880	4.35 %
$(1,5)$	6048	4.48 %
$(1,6)$	6090	4.51 %
$(3,1)$	7560	5.59 %
$(1,4)$	7770	5.75 %
$(2,2)$	8820	6.53 %
$(1,1)$	17872	13.23 %
$(1,3)$	18200	13.47 %
$(1,2)$	25851	19.13 %
$(2,1)$	29400	21.76 %

Table 1: Size distributions for index-period equivalence classes of different 'full' diagram semigroups. The tables are for the full transformation semigroup of degree 7, $\mathcal{T}_7$; the symmetric group of degree 10, $\mathcal{S}_{10}$; and for the Brauer monoid of degree 7, $\mathfrak{B}_7$. The rows contain the index-period pair, the number of elements with that index-period, and that size also expressed as a percentage of the total number of elements. In groups the only idempotent is the identity, thus in $\mathcal{S}_{10}$ we have the ratio $\frac{1}{3628800}$ appearing as 0.00%. The point of these tables is to show that sizes are not evenly distributed.

Definition 2.5 ($\cong_{\boxplus}$)**.** For a finite semigroup S, the equivalence relation of having the same element profile is written as $s \cong_{\boxplus} t$, $s, t \in S$.

The *table level invariants* can be used to decide the possibility of an isomorphism. They are frequency distributions of the

1. elements,
2. diagonal frequencies,
3. column and row frequencies,
4. idempotents (very strong semigroup invariant),
5. element profiles.

These invariants are sensitive enough to tell some small groups apart, despite their very special nature (Latin square multiplication table, single principal ideal and idempotent). Diagonal frequencies can tell some groups apart: $\mathbb{Z}_4 \mapsto (2, 2)$, $\mathbb{Z}_2 \times \mathbb{Z}_2 \mapsto (4)$, but it assigns $(2, 6)$ to both D_8 (dihedral group) and Q_8 (quaternion group). In the group case, element profiles reduces to the order of elements, but it can distinguish between D_8 and Q_8. However, this invariant fails to detect the difference between some direct and semidirect products. For instance, $\mathbb{Z}_8 \times \mathbb{Z}_2$ and $\mathbb{Z}_8 \rtimes \mathbb{Z}_2$ both have 1 element of order 1, 3 of order 2, 4 of order 4, and 8 of order 8.

Example 2.6. Let S be the semigroup generated by transformations $[2, 1, 1], [2, 3, 2]$, and $[3, 1, 3]$. Using only index-period values to classify its 15 elements, the size of the search space to find isomorphisms to another representation of S is 2903040. Classifying by element profiles defined above reduces the search space size to 768.

3 Algorithmic Improvements

To scale up practical computations we need to apply several optimization techniques. These all exploit some particular mathematical properties of the semigroups. The incremental nature of homomorphisms allow parallel execution, symmetries of the semigroups lead to more efficient enumeration, generator sets allow computing homomorphisms 'on the fly', and we can speed up computations by implementing custom representations for different diagram semigroups.

3.1 Parallel Execution

A search can be executed easily in parallel if parts of the search space can be separated in a way that instances of the algorithm never cross the boundaries, often called as 'embarrassingly parallel'. For backtrack, this can be achieved by starting the search from different partial solutions. Since homomorphisms take subsemigroups to subsemigroups we can attempt to extend an embedding $\varphi_U : U \hookrightarrow T$ (a partial solution) to $\varphi_S : S \hookrightarrow T$, if U is a subsemigroup of S. Therefore, we can utilize several processes to calculate all embeddings. First we find all embeddings φ_U sequentially, assuming that U is small enough to be calculated quickly. Then we can run searches starting from each different φ_U in parallel in order to find all embeddings φ_T.

Load balancing is a potential issue. In the extreme case, a single or few partial solutions can take most of the required work, while the other threads finish quickly. At the scale of the current experiments, this issue was not observed, since checking a single partial solution is quick relative to the large number of them. Therefore, grouping several of them into a work package mitigates the problem.

The implementation uses Clojure's library for reducers based on Java's Fork-Join framework [32]. The parallel search algorithms are in a separate package called Orbit [17]. The package exposes a single parameter `*task-size*` to counter potential load balancing issues.

3.2 Finding Distinct Embeddings up to Conjugation

It is possible to find an embedding in different ways (different bijections from the source semigroup to one particular image inside the source semigroup), and to find different embeddings (several isomorphic copies of the image in the target semigroup). The first case is when the source semigroup has symmetries, i.e. its automorphism group $\mathrm{Aut}(S) = \{\sigma : S \to S \mid \sigma(S) \simeq S\}$ is non-trivial. Similarly, some pairs of embeddings can be transformed into each other by those symmetry operations. Both correspond to relabelings of the elements. We say that those solutions are *conjugate* to each other and being conjugate is an equivalence relation. Therefore, it is enough to produce a single representative element from each conjugacy class.

The representative of a conjugacy class of embeddings can be chosen to be the minimal one in lexicographic ordering. For a single transformation (endofunction) we can compute the minimal representative in $O(n^2)$ time, where n is the degree of the transformation [38]. However, we also have to calculate conjugacy class representatives for sequences and sets of transformations (and other diagram elements). Those involve enumerating permutations. The worst case is the complete enumeration of the symmetric group $\mathcal{S}_n$, which happens for instance for the set of n constant maps of degree n.

For sequences, since a partial solution is a sequence, we can easily home in on the minimal one (as opposed to finding minimal elements for set-wise conjugation). As a sequence grows, the set of symmetries which can take the sequence to its representative is shrinking, since there are more and more constraints on how the sequence can be transformed into the corresponding minimal one. Computational experiments show that the symmetry classes of sequences become singletons long before the embedding is fully defined. Once we have only one possible such symmetry, we can be sure that any extension to a full solution will be a conjugacy class representative.

3.3 Generator Tables

When computing with algebraic structures, as much it is possible, we want to work with the generators only, not with the complete structure. The generators allow to construct semigroup elements on demand, where the complete semigroup is too big to fit into memory. An extreme example is $\mathcal{T}_n$, the full transformation semigroup of degree n, which has n^n elements, but the standard generating set has only 3 elements: a cycle, a transposition and a collapser (see Figure 9 for this generator set). The fully computed multiplication table would have n^{2n} entries.

The key observation is that we only to be able to multiply by the generators, not by an arbitrary element. Thus, for constructing morphisms, it is enough to have the multiplication table columns corresponding to multiplication by the generators. We call this a *generator table*. In the source semigroup, we can use a generator table. This trick does not work in the target semigroup, as we keep choosing new candidate generators. However, it is enough to enumerate the suitable elements only (e.g., the index-period equivalence classes matching the source generators). This works well for 'full' structures of certain types where we can combinatorially enumerate semigroup elements with the required properties instead of generating all elements and filtering the candidates.

In a sense, we are building an isomorphism of the Cayley-graphs of the source and of the image subsemigroup in the target semigroup. Once we choose a candidate target generator, we can build both semigroups at the same time to see whether the multiplications are compatible. A partial solution for this method is not a sequence (instead of an array we have a hash table). Thus, to enumerate the morphisms up to relabelings, we need to find minimal class representatives by set-wise conjugation.

The generator table algorithm assumes that the given generating set A for the source semigroup S is *independent*: $\forall a \in A,\ a \notin \langle A \setminus \{a\}\rangle$, i.e., no generator is generated by the other generators. The independence property ensures that we do not have to check for mistaken violations of the compatibility condition due to re-assigning a generator.

3.4 Choosing Generating Sets

Using the generator table method we can observe an important fact: different generator sets give different execution times for finding embeddings. One of the factors is the number of possible candidate targets for each generator. Table 1 shows the uneven distribution of index-period class sizes. Thus, it seems to be a good

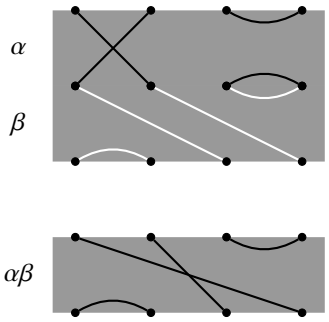

Figure 5: Multiplication in the Brauer monoid $\mathfrak{B}_4$. In our custom representation $\alpha = [6, 5, 4, 3, 2, 1, 8, 7]$, $\beta = [7, 8, 4, 3, 6, 5, 1, 2]$ and the product is $\alpha\beta = [8, 7, 4, 3, 6, 5, 2, 1]$. 'Loops' formed in the middle are ignored. In general, we may need to trace long paths of alternating black and white edges.

idea to choose generators that have index-period with a low frequency in the target semigroup. However, the situation is more subtle. The ordering of the generators could influence the length of the search. For instance, embedding $\mathcal{S}_4$ with generators $[2, 3, 4, 1], [2, 1, 3, 4]$ is faster than with the same generators in reverse order $[2, 1, 3, 4], [2, 3, 4, 1]$.

3.5 Custom Representation for the Brauer Monoid

When computing with semigroups, multiplication is often the most frequently repeated operation. Consequently, we can improve the speed of the algorithms in two ways:

1. reducing the required multiplications, or
2. making individual multiplications faster.

The fastest possible multiplication is just the table lookup in the abstract representation, but it assumes the generation of the complete semigroup (implying lot of concrete representation multiplication). The generator table method above does not require the target semigroup to be fully computed, thus we need to use a concrete combinatorial representation.

For diagram semigroups, we can implement a single multiplication algorithm for the most general partitioned binary relation representation [34] (see Section 1.2). In those, for each point we have a *set* of images. Obviously, this would be a wasteful representation for transformations and permutations, where each point has only a single image, and computing with sets is an unnecessary overhead. Therefore, computer algebra software packages have custom representations for transformations, usually just a list of images.

Another interesting case is when we have some less common diagram representation, closer to the most general. In this project, we implemented a specific data structure and multiplication for Brauer monoid elements [4]. The elements are diagrams with $2n$ points, n at the 'top', n at the 'bottom'. A point can be connected to exactly one other point. We represent such a diagram b with a list of $2n$ entries, $[b(1), \ldots, b(2n)]$, with the constraint that $b(i) = j \implies b(j) = i$. There is no need to enforce these constraints when working with a generating set, since multiplication of two such diagrams produce a third with the same properties. See Figure 5 for an example, and Figure 6 for the details of the multiplication algorithm. Since we represent the Brauer diagrams as transformations, we can use the same mechanism for conjugation. We just need to have to duplicate a permutation on the points $1, \ldots, n$ to points $n + 1, \ldots, 2n$. This made the computation of Table 4 possible.

A performance comparison was made between the general partitioned binary relation implementation and the custom Brauer diagram data structure, see Figure 7. As the actual calculations are the same in both cases, the performance gain is due to the usage of vectors instead of hash-maps and hash-sets. Although the partitioned binary relation implementation is slow, in practice, we used it to test the correctness of more specific representations.

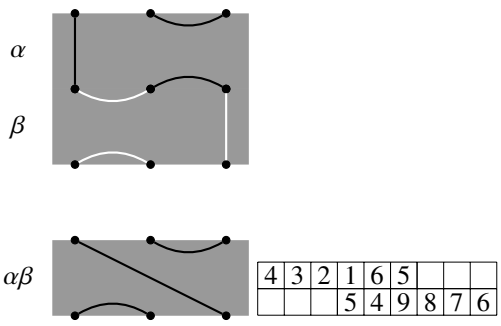

Figure 6: The implementation of the multiplication in the Brauer monoid. Given the diagrams $\alpha = [4, 3, 2, 1, 6, 5]$ and $\beta = [2, 1, 6, 5, 4, 3]$, degree 3 with 6 points, we shift the second by 3 points. This identifies the bottom points of α with the top points of β. We trace paths starting from α's top points $\{1, 2, 3\}$ and from β's shifted bottom points $\{7, 8, 9\}$. Then, we use these lists as transformations. Tracing ends when we have an undefined image (the empty cells). Loops in the middle never occur, since we do not trace from the middle points. At the end, we need to shift back the higher points to get $\alpha\beta$. For instance, tracing from point 1 produces the alternating path $1 \overset{\alpha}{\mapsto} 4 \overset{\beta}{\mapsto} 5 \overset{\alpha}{\mapsto} 6 \overset{\beta}{\mapsto} 9$ and then the pair $(1, 9)$. After shifting back, the result is the pair $(1, 6)$.

	$\mathfrak{B}_4$	$\mathfrak{B}_5$	$\mathfrak{B}_6$
size	105	945	10395

time (ms)	$\mathfrak{B}_4$	$\mathfrak{B}_5$	$\mathfrak{B}_6$
general	38	440	5678
custom	6	62	744

memory	$\mathfrak{B}_4$	$\mathfrak{B}_5$	$\mathfrak{B}_6$
general	183.5 KB	2.1 MB	26.9 MB
custom	31 KB	304.3 KB	3.1 MB

Figure 7: The comparison of the general partitioned binary relation representation and a custom Brauer monoid representation. The computing task is to compute the Brauer monoid from generators, which involves many multiplications. The size of the monoid grows rapidly with the degree (see A001147 at [40]). The computing time was measured by the `criterium` tool (`http://hugoduncan.org/criterium/`), which takes care of statistical sampling, JIT compilation and garbage collection cycles. The memory footprint was measured with `clj-memory-meter` (`https://github.com/clojure-goes-fast/clj-memory-meter`). The tests were carried out on a Raspberry Pi 5.

4 Software Implementations

In computational semigroup theory, the Semigroups package [37] for the Gap computer algebra system [21] emerged as the leading solution. It also serves a base for more specialized packages.

The SubSemi package [10] contains the implementations of the above algorithms. Despite being inefficient for large semigroups due to the multiplication table representation, they proved to be immensely useful in practical computations. The original goal was to extend the results of transformation semigroup enumeration [3, 43]. SubSemi enumerated all 4-state finite computations up to isomorphism [11]. Moreover, the enumeration was extended for more general diagram semigroups [12]. Deciding isomorphism was also needed for enumerating independent generating sets of symmetric groups [14]. These applications required to develop the isomorphism and embedding construction algorithms described in this paper.

For the isomorphism calculation, we can compare our method to two other methods in the Semigroups package [37]. The older method uses the `SmallestMultiplicationTable` function that depends on the function `SmallestImageSet` in Gap [21] and the GRAPE package [41]. This function calculates the smallest multiplication table in lexicographic ordering for the semigroup (by calculating a group action orbit of the symmetric group), which can be used for isomorphism testing. Its complexity depends on the size of the semigroup since we have to act on the table by the corresponding symmetric group, while our method does not require group actions and it uses more information about the semigroup structure, thus it is more scalable. In practice, roughly speaking, with SubSemi we can calculate embeddings and isomorphisms as long as the fully enumerated semigroup fits into the memory of a single computer (e.g. embeddings into $\mathcal{T}_8$ with $8^8 = 16777216$ elements).

The newer method uses graph isomorphism solvers directly for computing isomorphism: the multiplication tables are turned into vertex coloured (non-directed) graphs, and then checked whether they are isomorphic using bliss [29] or nauty [35] (depending on which is enabled in the Digraphs package [2]). The graph constructed has many more nodes than the semigroup has elements (they are used to indicate the direction of the edges, and the element that we are multiplying by).

For computational experiments, where the results are not given by mathematical proofs, *verification* by *recomputation* is important. We routinely reproduced results by other implementations, and developed a 'redundant' package for the algorithms presented here.

We recomputed the results of [1], using our method on small semigroups [8]. We also used this technique for finding isomorphism classes and determining the automorphism groups of all transformation semigroups of degree 4 [11].

For further verification a 'shadow' implementation was also developed, Kigen [16]. It is built on a different architecture: Clojure [25], a purely functional programming language on top of Java, while Gap is a functional object-oriented language built around a C/C++ kernel. The Kigen package benefits from the seamless parallelization features of Clojure, a crucial ingredient for the results below.

5 Computational Experiments and Results

5.1 Minimal Degree Realization of Computations

A basic result of semigroup theory is that any semigroup with n elements can be realized as a transformation semigroup of at most degree $n + 1$ (the semigroup analogue of Cayley's Theorem for groups, e.g. [26], Chapter 1). In practice, we would like to have a smaller degree transformation representation. This is a more general problem than finite automata minimization (where to goal is to recognize a language efficiently). This problem naturally generalizes to all types of diagram semigroups. Given an abstract semigroup, we would like to find minimal diagram semigroup realizations of it (in terms of the points in the diagrams) for a given type of diagram. The general strategy is trying to find embeddings into the degree n full diagram semigroups (containing all degree n diagrams of that type).

Definition 5.1. For a semigroup S the minimal D-diagram representation degree is $\mu_D(S) = \min\{n \mid S \hookrightarrow D_n\}$ where $D \in \{\mathcal{P}\mathrm{B}, \mathrm{B}, \mathrm{P}\mathcal{T}, \mathcal{T}, \mathcal{S}, \mathcal{I}, \mathcal{I}^*, \mathcal{P}, \mathfrak{B}, \mathrm{TL}\}$.

$$S = \left\{ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \right\},$$

S	1	2	3	4	5
1	1	1	1	1	1
2	1	2	3	2	3
3	1	1	1	2	3
4	1	4	5	4	5
5	1	1	1	4	5

Figure 8: An example matrix semigroup with listed elements and the corresponding abstract multiplication table. The numbers in the table correspond to the order of the listed matrices.

Example 5.2. Finding minimal degree diagram representations of a matrix semigroup over $\mathbb{Z}$ by using its multiplication table (Fig. 8). The transformation and partition representation both require 3 points $\mu_{\mathcal{T}}(S) = \mu_{\mathcal{P}}(S) = 3$. We need two more points to represent S as Brauer and Temperley-Lieb monoid: $\mu_{\mathfrak{B}}(S) = \mu_{\mathrm{TL}}(S) = 5$.

5.2 Comparing 'Computational Power'

Various diagram representations can be considered as different models of computation. Traditionally, we compute by composing functions, but we can use binary relations or equivalence relations. An interesting question is how much 'bigger' in size a less capable computational device should be in order to realize a more powerful one. To what extent can size make up for structure?

For diagram semigroups we can use the minimal degree diagram representation search to assess relative computational power. For instance, to emulate computations of equivalence classes with transformations, for $n = 2$ we need at least 7 states: $\mathcal{P}_2 \hookrightarrow \mathcal{T}_7$. In this case we know precisely the minimum degrees [6].

The degree 1 partition binary monoid has only 16 elements, but realized as transformations it requires 6 points: $\mathcal{P}\mathrm{B}_1 \hookrightarrow \mathcal{T}_6$. This value is substantially lower than 17 required by the right regular representation.

Thinking backwards, transformations realized by some stronger computational model, we find that the distinction between reversible and irreversible computation gives a hard limit. Despite its huge size $|\mathcal{P}\mathrm{B}_2| = 65536$, partitioned binary relation monoids of lower degree cannot emulate full transformations semigroups, due to the lack of permutations.

Here is a list of similar results: $\mathfrak{B}_1 \cong \mathcal{T}_1$, $\mathfrak{B}_2 \hookrightarrow \mathcal{T}_3$, $\mathrm{TL}_1 \cong \mathcal{T}_1$, $\mathrm{TL}_2 \hookrightarrow \mathcal{T}_2$, $\mathrm{TL}_3 \hookrightarrow \mathcal{T}_4$, $\mathcal{P}_1 \hookrightarrow \mathfrak{B}_2$, $\mathfrak{B}_3 \hookrightarrow \mathrm{B}_3$, $\mathcal{I}_2^* \hookrightarrow \mathrm{B}_3$. Many of these involve quite heavy computations.

An interesting case is the embedding of the full transformation semigroup into the Brauer monoid $\mathcal{T}_2 \hookrightarrow \mathfrak{B}_3$ but $\mathcal{T}_3$ does not embed into $\mathfrak{B}_7$. This raised the question whether it would be possible to embed it into $\mathfrak{B}_8$. At this point we needed parallelization to check that it does not. The embedding already fails for the semigroup generated by the collapser and the cycle of the standard generating sets (see Figure 9 for the generator types). Missing the transposition generator yields a subsemigroup of $\mathcal{T}_3$ with 24 elements, allowing a bit faster embedding checks.

5.3 Counting Distinct Embeddings

Beyond the question whether an embedding is possible or not, we are also interested in the number of different embeddings, up to conjugation. We use the notation $\#(S \hookrightarrow T)$ to indicate the number of distinct copies of S inside T.

We systematically explored the embeddings of three types of full diagram semigroups into their higher degree counterparts. This question is about emulating something by a more capable computing structure of the same type. There is always a trivial solution: just fix the added points. However, the other solutions are not trivial. The higher degree diagram semigroup has to manage the extra states or points in a way that they do not interfere with the computation in the smaller structure. The following tables are expected to be helpful for mathematical reasoning to eventually produce closed formulas for the numbers of embeddings.

Symmetric groups are useful test cases for verification purposes since we have efficient group theoretical algorithms for computing these embeddings (Table 2). Similarly, Table 3 summarizes distinct embeddings for low degree transformation semigroups. Figure 9 indicates how these embeddings work.

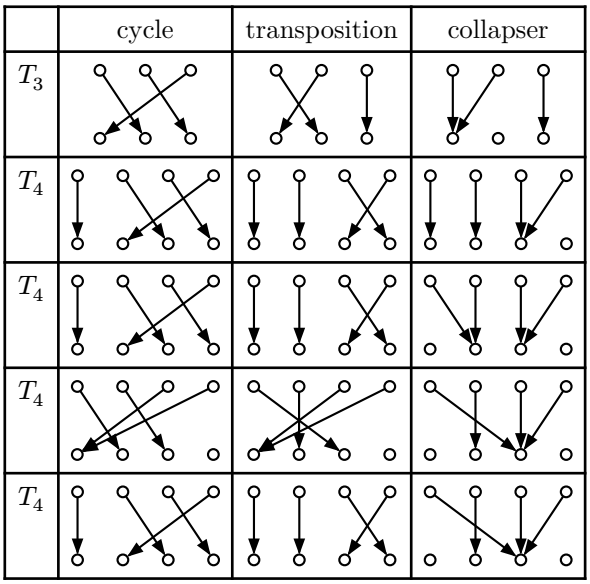

Figure 9: All 4 (up to conjugation) embeddings of $\mathcal{T}_3$ into $\mathcal{T}_4$ given by the mappings of the standard generators. The embeddings simply fix the additional point, or collapse it into another point. For higher degree embeddings, avoiding the interference with the multiplication in $\mathcal{T}_3$ yields combinatorial structures of similar fixing and collapsing.

The full transformation embeddings give interesting information for the enumeration of transformation semigroup. There are 282 non-empty transformation semigroups on 3 points up to conjugation. For degree 4, this number is 132069775 [11]. How many isomorphic copies of the degree 3 transformation semigroups can we find inside $\mathcal{T}_4$? Counting the embeddings up to conjugation, we find only 2347 subsemigroups of $\mathcal{T}_4$ that are isomorphic to some subsemigroup of $\mathcal{T}_3$; so most degree 4 transformation semigroups are 'new'. Calculating the same number for $\mathcal{T}_5$ yields 18236; a modest increase compared to the still unknown, but expected-to-be gigantic number of degree 5 transformation semigroups.

As a third type, we computed Brauer monoid embeddings, see Table 4. Although enumeration results for Brauer semigroups are only preliminary [12], similar arguments seem to apply, most of the next degree semigroups are not isomorphic to the smaller degree semigroups.

5.4 Embeddings into 2-generated Subsemigroups

Here, we will answer a couple of open questions stated in [13] using the embedding algorithm. There, the primary interest is in the embeddability into a 2-generated semigroup, denoted by $S \overset{\text{2-gen}}{\hookrightarrow} T$. Being n-generated means that the semigroup can be generated by n elements but no less. It is easy to prove that $\mathfrak{B}_n \overset{\text{2-gen}}{\hookrightarrow} \mathfrak{B}_{n+2}$, but there is no 2-generated subsemigroup of $\mathfrak{B}_{n+1}$ where we can embed $\mathfrak{B}_n$. The same was conjectured for the partition monoid, but contrary to the expectation we found that $\mathcal{P}_2 \overset{\text{2-gen}}{\hookrightarrow} \mathcal{P}_3$, in 3 different ways.

To obtain this result, we enumerated conjugacy class representatives of subsemigroups of the target semigroup that are 2-generated, then filtered them for the property of being at least as big as the source semigroup. Finally, we used our embedding search to check each of the candidate 2-generated subsemigroups.

6 Conclusion and Future Work

We carried out computational experiments to extend our knowledge about diagram semigroups, which are different combinatorial representations of semigroups. For constructing embeddings and deciding isomor-

$\#(\mathcal{S}_m \hookrightarrow \mathcal{S}_n)$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$	$n=9$	$n=10$
$m=1$	1	1	1	1	1	1	1	1	1	1
$m=2$		1	1	2	2	3	3	4	4	5
$m=3$			1	1	2	4	5	7	10	12
$m=4$				1	1	4	5	10	13	22
$m=5$					1	2	3	4	5	9
$m=6$						1	1	2	2	4
$m=7$							1	1	2	2
$m=8$								1	1	
$m=9$									1	

Table 2: Number of distinct embeddings of symmetric groups calculated with the generator table method. These are used for verification purposes. Further values can be computed by the GAP code `Length(IsomorphicSubgroups(SymmetricGroup(n), SymmetricGroup(m)));`, see sequences A378266, A378273, A378274 on OEIS [40].

$\#(\mathcal{T}_m \hookrightarrow \mathcal{T}_n)$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$
$m=1$	1	2	3	5	7	11	15	22
$m=2$		1	3	12	35	110	309	879
$m=3$			1	4	17	64	221	736
$m=4$				1	2	6	16	48
$m=5$					1	2	6	16
$m=6$						1	2	6
$m=7$							1	2
$m=8$								1

Table 3: Number of distinct embeddings of full transformation semigroups. Embedding the trivial monoid is equivalent to finding idempotent elements ($e^2 = e$) up to conjugation in the target semigroup (see A000041 on OEIS [40]). An interesting pattern seems to emerge for $\#(\mathcal{T}_i \hookrightarrow \mathcal{T}_n)$, when $i \in \{n-3, n-2, n-1\}$.

phisms the partitioned backtrack algorithm reduces the search space by exploiting information about the source and the target semigroups. Further algorithmic optimizations improved scalability, thus we could solve open problems by computational means.

We can extend the optimizations by considering the order of the semigroup elements, borrowing ideas from graph colouring. The so-called "Welsh-Powell" ordering of the nodes (from highest to lowest degree) [42] uses the idea that the definitions made at the start of the sequence imply more restrictions on the later values if they are more connected. The opposite example shows why this could be useful. If there is an isolated vertex, and we define its φ value first, then this implies no restrictions at all on any subsequent definitions.

Another future task is to do a comprehensive comparison of the now available different methods, especially the performance differences between the specialized and the general search methods.

Software Tools

The scripts required to reproduce these results are bundled with the software packages [10, 16]. The subfolder `2025_Computing_Embeddings_and_Isomorphisms_of_Finite_Semigroups` containing the computations for this paper can be found in the `experiments` folder of the package in both projects. These software project did not use any generative AI tools at any stage of the development process.

$\#(\mathfrak{B}_m \hookrightarrow \mathfrak{B}_n)$	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 5$	$n = 6$	$n = 7$	$n = 8$
$m = 1$	1	2	3	5	7	11	15	22
$m = 2$		1	1	9	11	42	63	
$m = 3$			1	2	13	36	145	
$m = 4$				1	1	5	8	
$m = 5$					1	1		
$m = 6$						1		
$m = 7$							1	

Table 4: Brauer monoid embeddings. The number of idempotents up to conjugation is the same as in full transformation semigroups as they are both parametrized by the integer partitions (A000041 on OEIS [40]).

Acknowledgement

This project was funded in part by the Kakenhi grant 22K00015 by the Japan Society for the Promotion of Science (JSPS), titled 'On progressing human understanding in the shadow of superhuman deep learning artificial intelligence entities' (Grant-in-Aid for Scientific Research type C, https://kaken.nii.ac.jp/grant/KAKENHI-PROJECT-22K00015/).

References

[1] J. Araújo, P.V. Bünau, J.D. Mitchell, and M. Neunhöffer. Computing automorphisms of semigroups. *Journal of Symbolic Computation*, 45(3):373 – 392, 2010.

[2] Jan De Beule, Julius Jonušas, James D. Mitchell, Michael Torpey, Maria Tsalakou, and Wilf A. Wilson. Digraphs - GAP package, version 1.13.1, Sep 2025.

[3] G. Bijev and K. Todorov. On the representation of abstract semigroups by transformation semigroups: Computer investigations. *Semigroup Forum*, 43(1):253–256, 1991.

[4] Richard Brauer. On algebras which are connected with the semisimple continuous groups. *Annals of Mathematics*, 38(4):857–872, 1937.

[5] Peter J. Cameron. *Permutation Groups*. London Mathematical Society, 1999.

[6] Reinis Cirpons, James East, and James D. Mitchell. Transformation representations of diagram monoids, 2025.

[7] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. Association for Computing Machinery.

[8] A. Distler and J. Mitchell. Smallsemi, A library of small semigroups, Version 0.7.2. https://gap-packages.github.io/smallsemi, Feb 2025. GAP package.

[9] John D. Dixon and Brian Mortimer. *Permutation Groups*. Graduate Texts in Mathematics 163. Springer, 1996.

[10] J. East, A. Egri-Nagy, and J. D. Mitchell. SubSemi – *GAP package for enumerating subsemigroups, v0.86*, 2025. https://gap-packages.github.io/subsemi/.

[11] J. East, A. Egri-Nagy, and J. D. Mitchell. Enumerating transformation semigroups. *Semigroup Forum*, 95(1):109–125, 2017.

[12] J. East, A. Egri-Nagy, A. R. Francis, and J. D. Mitchell. Finite diagram semigroups: Extending the computational horizon. arXiv:1502.07150 [math.GR], 2015.

[13] James East. Partition monoids and embeddings in 2-generator regular *-semigroups. *Periodica Mathematica Hungarica*, 69(2):211–221, 2014.

[14] A. Egri-Nagy and V. Gebhardt. Computational enumeration of independent generating sets of finite symmetric groups. arXiv:1602.03957 [math.GR], 2016.

[15] Attila Egri-Nagy. Finite computational structures and implementations: Semigroups and morphic relations. *International Journal of Networking and Computing*, 7(2):318–335, 2017.

[16] Attila Egri-Nagy. KIGEN Computational Semigroup Theory Software System written in Clojure. codeberg.org/egri-nagy/kigen, 2025.

[17] Attila Egri-Nagy. ORBIT Generic sequential and parallel search algorithms. codeberg.org/egri-nagy/orbit, 2025.

[18] Attila Egri-Nagy and Chrystopher L. Nehaniv. The attractor-cycle notation for finite transformations. In D. Marc Kilgour, Herb Kunze, Roman N. Makarov, Roderick Melnik, and Xu Wang, editors, *Addressing Modern Challenges in the Mathematical, Statistical, and Computational Sciences*, pages 35–45, Cham, 2025. Springer Nature Switzerland.

[19] D. G. Fitzgerald and Jonathan Leech. Dual symmetric inverse monoids and representation theory. *Journal of the Australian Mathematical Society (Series A)*, 64:345–367, 6 1998.

[20] Olexandr Ganyushkin and Volodymyr Mazorchuk. *Classical Transformation Semigroups*. Algebra and Applications. Springer, 2009.

[21] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.15.1*, 2025. gap-system.org.

[22] Martin Grohe and Pascal Schweitzer. The graph isomorphism problem. *Commun. ACM*, 63(11):128–134, 2020.

[23] E. H. Lieb H. N. V. Temperley. Relations between the 'percolation' and 'colouring' problem and other graph-theoretical problems associated with regular planar lattices: Some exact results for the 'percolation' problem. *Proc. Roy. Soc. A, Mathematical and Physical Sciences*, 322(1549):251–280, 1971.

[24] T. Halverson and A. Ram. Partition Algebras. *European J. Combin.*, 26(6):869–921, 2005.

[25] Rich Hickey. A history of Clojure. *Proc. ACM Program. Lang.*, 4(HOPL), June 2020.

[26] John M. Howie. *Fundamentals of Semigroup Theory*, volume 12 of *London Mathematical Society Monographs New Series*. Oxford University Press, 1995.

[27] V. F. R. Jones. The Potts model and the symmetric group. In *Subfactors (Kyuzeso, 1993)*, pages 259–267. World Sci. Publ., River Edge, NJ, 1994.

[28] V. F. R. Jones. A quotient of the affine Hecke algebra in the Brauer algebra. *Enseign. Math. (2)*, 40(3-4):313–344, 1994.

[29] Tommi Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the ninth Workshop on Algorithm Engineering and Experiments and the fourth Workshop on Analytic Algorithmics and Combinatorics*, pages 135–149, 2007. Society for Industrial and Applied Mathematics cop.; 978-0-898716-28-3; Workshop on Algorithm Engineering and Experiments.

[30] D.E. Knuth. *The Art of Computer Programming*, volume 1-3. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[31] M.V. Lawson. *Inverse Semigroups: The Theory of Partial Symmetries*. World Scientific, 1998.

[32] Doug Lea. A Java fork/join framework. In *Proceedings of the ACM 2000 Conference on Java Grande*, JAVA '00, pages 36–43, New York, NY, USA, 2000. ACM.

[33] Paul Martin. Temperley-Lieb algebras for nonplanar statistical mechanics—the partition algebra construction. *J. Knot Theory Ramifications*, 3(1):51–82, 1994.

[34] Paul Martin and Volodymyr Mazorchuk. Partitioned binary relations. *Mathematica Scandinavica*, 113(1):30–52, 2013.

[35] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, 2014.

[36] Gary L. Miller. Graph isomorphism, general remarks. *Journal of Computer and System Sciences*, 18(2):128–142, April 1979.

[37] J. D. Mitchell et al. *Semigroups - GAP package, Version 5.5.1*, Jun 2025.

[38] James D. Mitchell, Sujoy Mukherjee, and Petr Vojtěchovský. Minimal representatives of endofunctions. *Semigroup Forum*, 109(3):626–638, 2024.

[39] R. J. Plemmons and M. T. West. On the semigroup of binary relations. *Pacific Journal of Mathematics*, 35(3):743–753, 1970.

[40] Neil J. A. Sloane and The OEIS Foundation Inc. The on-line encyclopedia of integer sequences, 2026.

[41] L.H. Soicher. Computing with graphs and groups. In *Topics in Algebraic Graph Theory*. Cambridge University Press, 2004.

[42] D. J. A. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 01 1967.

[43] Carroll Wilde and Sharon Raney. Computation of the transformation semigroups on three letters. *Journal of the Australian Mathematical Society*, 14:335–335, 11 1972.