NEW ZEALAND JOURNAL OF MATHEMATICS
Volume 56 (2025), 53–68
https://doi.org/10.53733/329

CLASSIFICATION OF CONNECTED SHELVES

Mohamed Elhamdadi, Neranga Fernando, and Mathew Goonewardena

(Received 25 February 2023)

Abstract. We investigate finite right-distributive binary algebraic structures called shelves. We first use symbolic computations with Python to classify (up to isomorphism) all connected shelves with order less than six. We explore the group structure generated by the rows of *latin* shelves. We also define two-variable shelf polynomial by analogy with the quandle polynomial and then state a conjecture about connected idempotent shelves.

1. Introduction

Shelves are sets with binary operations satisfying self-distributivity

$$(x * y) * z = (x * z) * (y * z).$$

These algebraic structures are derived from the axiomatization of Reidemeister move III in classical knot theory. By adding the condition corresponding to Reidemeister move II, one gets the notion of a rack. Racks have been used to obtain invariants of framed knots [**6**]. Framed knots can be visualized as closed loops of knotted flat ribbons. Framed knots generated lot of interests mainly because of the critical role they play in low-dimensional topology [**6**]. By adding the condition corresponding to Reidemeister move I to the definition of a rack, one obtains the notion of a quandle. Quandles have been studied extensively and are used to obtain invariants of knots and links [**5**]. In [**2**], associative shelves have been investigated and it was shown that unital shelves are associative. The authors also investigated one-term and two-term homology groups of some associative self-distributive algebraic structures in [**2**]. In [**1**], self-distributivity was investigated in a unified manner via a categorical technique called internalization, a cohomology theory was developed, and explicit relations to rack and Lie algebra cohomology theories were given. Self distributivity was also investigated from a group theory point of view in [**4**], where the automorphism groups of many quandles, including dihedral quandles, were determined.

A *monounary* algebra is an algebra with one unary operation, usually denoted as a pair (A, f), where A is a nonempty set with a map $f : A \to A$. In [**8**], the author considered finite monounary algebras and proved that every monounary algebra has at least one left-distributive extension E such that $x * y = z$ in A means $x * y = z$ in E. A left-distributive extension of A is simply a left-distributive groupoid. The author also gave an enumeration algorithm which computes the numbers of all left-distributive groupoids and their isomorphism types on a given set of cardinality less or equal to six. It is worth noting that the left-groupoids discussed in [**8**]

Mohamed Elhamdadi was partially supported by Simons Foundation collaboration grant 712462. The authors would like to thank Sujoy Mukhejree for fruitful comments which improved the paper.

are exactly the shelves that are under consideration in our paper, but with right-distributive property. However, to the best of our knowledge, a classification of connected shelves up to isomorphism has not appeared in the literature.

A Laver table [**9**] is a free shelf generated by one element. Precisely, Laver proved that for every $N \geq 1$, there exists a unique binary operation $*$ on the set $X = \{1, 2, \ldots, N\}$ such that, for all x, y, satisfies

$$x * 1 = x + 1 \pmod{N},\ x * (y * 1) = (x * y) * (x * 1)$$

Then the binary operation $*$ is left-distributive if and only if N is a power of 2. Laver tables were introduced in 1995 by Richard Laver while investigating self-embedding in set theory (see for example [**3**] for more details).

In this article, we use symbolic computations with Python to obtain the list of shelves of order less than six. We also define two-variable shelf polynomial by analogy with the quandle polynomial and then state a conjecture about connected idempotent shelves. The organization of this paper is as follows: Section 2 reviews the basics of shelves in general and connected shelves in particular. In Section 3, we describe the algorithm which allows the classification of connected shelves of order less than six. In Section 4, we study connected shelves and give the number of connected shelves of order up to 6. We also present a conjecture regarding connected shelves. A *latin* shelf is a shelf whose rows are permutations. We also study latin shelves in Section 4, especially the group structure generated by rows of latin shelves. Section 5 introduces shelf polynomial, and we end the section with a conjecture strongly supported by our computational results. In Appendix A, we give the list of all connected shelves of order less than six.

2. Review of Shelves

In this section, we review the basics of shelves, give some examples and introduce the notion of connected shelves.

Definition 2.1. A *shelf* $(S, *)$ is a set S with a binary operation $*$ that is right-distributive. Precisely, for all $x, y, z \in S$, we have:

$$(x * y) * z = (x * z) * (y * z).$$

A shelf homomorphism between two shelves $(S, *_1)$ and $(S', *_2)$ is a map $f : S \to S'$ such that for all $x, y \in S$ we have $f(x *_1 y) = f(x) *_2 f(y)$. A shelf *isomorphism* is a bijective shelf homomorphism. Two shelves are isomorphic if there is a shelf isomorphism between them.

Typical examples of shelves include:

- The set $\mathbb{Z}_n$ of integers modulo n with $x*y = \alpha x+\beta y$, such that $\beta(\alpha+\beta-1) = 0$ in $\mathbb{Z}_n$. For example $\mathbb{Z}_{10}$ with $x * y = 2x + 5y$.
- Any group G with conjugation $x * y = yxy^{-1}$.
- Laver tables: $\mathbb{Z}_{2^n}$ with $1 * x = x + 1$ and $(1 * x) * y = (1 * y) * (x * y)$ (notice that we are using right-distributivity instead of left-distributivity).

Let $(S, *)$ be a shelf. The right multiplication R_x by an element x of S is the map $R_x : S \to S$ such that $R_x(y) = y * x$. The monoid generated by R_x is denoted by M_S. This monoid acts naturally on S. When this action is transitive we say that the shelf S is *connected*. In other words, if for all $x, y \in S$, there exists $\phi \in M_S$ such that $\phi(x) = y$, then we say that the shelf S is connected.

A *unital* shelf is a shelf S with an identity element. That is, there exists an element 1 in S such that $1 * x = x = x * 1$ for all $x \in S$. A *spindle* is a shelf whose elements are idempotent. In [**2**], the authors developed a theory of associative shelves, and studied their one-term and two-term homology groups. They also presented the number of unital shelves of order up to 4 in Table 3. The following table is an update of it.

n	number of unital shelves
1	1
2	1
3	2
4	6
5	23

3. An Algorithm to Generate Shelves

A matrix $M \in M_n(\mathbb{Z}_n)$ is a shelf matrix if it satisfies right distributivity (ref. to definition 2.1). A brute-force algorithm that enumerates all matrices in $M_n(\mathbb{Z}_n)$ and check each candidate for right distributivity does not scale for orders higher than four, e.g., order five requires checking 5^{25} candidate matrices. This section presents a computationally efficient algorithm to generate shelves without enumerating all of $M_n(\mathbb{Z}_n)$. The proposed algorithm is also parallelizable.

The right distributivity property requires to check n^3 conditions (all possible orderings of three elements drawn with replacement from the set). Let us denote this set of conditions by $\mathcal{C}_n$ and one of those conditions by a tuple of length three (x, y, z), where $x, y, z \in \mathbb{Z}_n$. For example, consider order three where $\mathbb{Z}_3 = \{0, 1, 2\}$ and $\mathcal{C}_n \coloneqq \{(x, y, z) \mid x, y, z \in \mathbb{Z}_n\}$ is the set of conditions and $(0, 1, 2)$ denotes the particular right distributivity condition $(0 * 1) * 2 = (0 * 2) * (1 * 2)$.

The algorithm starts with an empty $n \times n$ matrix. Then it takes one of the conditions from $\mathcal{C}_n$ and generates all the possible partially filled matrices that satisfy the condition. We call these candidate matrices. For example consider $(0, 0, 0)$ condition in order three. Let us represent the empty matrix as

$$[[-1, -1, -1], [-1, -1, -1], [-1, -1, -1]],$$

where the rows are unstacked into a single row to preserve space and -1 is used to represents the empty value in the computer program (any value outside of $\mathbb{Z}_3$ will do). From this empty matrix generating all partially filled matrices that satisfy condition $(0, 0, 0)$ gives the following seven candidates:

(i) $[[0, -1, -1], [-1, -1, -1], [-1, -1, -1]]$
(ii) $[[1, -1, -1], [\ 0,\ \ 0, -1], [-1, -1, -1]]$
(iii) $[[1, -1, -1], [\ 1,\ \ 1, -1], [-1, -1, -1]]$
(iv) $[[1, -1, -1], [\ 2,\ \ 2, -1], [-1, -1, -1]]$
(v) $[[2, -1, -1], [-1, -1, -1], [\ 0, -1,\ \ 0]]$
(vi) $[[2, -1, -1], [-1, -1, -1], [\ 1, -1,\ \ 1]]$
(vii) $[[2, -1, -1], [-1, -1, -1], [\ 2, -1,\ \ 2]]$

The generation technique of the candidates is as follows. In Python or C programming array indices start at 0. Therefore, the operation $(x * y)$, where $x, y \in \mathbb{Z}_n$,

is identical to reading the value at (x, y) indices of the 2D array containing the matrix. Assume we are given a matrix M (partially or fully empty) and the condition (x, y, z). First check if the location (x, y) is empty in M and if so it can be filled with n possible values. This is done in a loop. Enter the loop and fill with one value. Now check if (x, z) is empty and if so it too can be filled with n possible values in a similar loop. Do the same for (y, z). Then in the final stage check if either one of $((x, y), z)$ or $((x, z), (y, z))$ is empty. If both are empty (or they point to same location in the matrix and that is empty) there are n possible values to consider, each of which satisfies the condition. If just one is empty then assign the value of other to it. If both are filled with the same value then the condition is satisfied. If they are filled but with different values then the condition is violated. The generation process is computationally intensive with a series of nested loops. Therefore, an efficient algorithm should minimize the total number of false candidates that it generates while searching for shelves. This brings us to the next important step of the algorithm.

For each of these candidates check if at least one of the remaining conditions is violated. For example consider the condition $(0, 0, 1)$ and first candidate

$$[[\ 0, -1, -1], [-1, -1, -1], [-1, -1, -1]]$$

generated above. To check both LHS and RHS of $(0, 0, 1)$ we need the value at $(0, 1)$. However, $(0, 1) = -1$, as the matrix location $(0, 1)$ has not yet been filled by a value from $\mathbb{Z}_3$. Therefore, one cannot make a conclusion as to if this condition is violated. Likewise, check this matrix for all of the remaining conditions for violations. This checking is important to reduce the search space in the latter steps. Those that do not violate any of the remaining conditions become candidates for the next step. Now in the next step the algorithm takes each of these candidates and the next condition and generates all possible candidates that do not violate the remaining conditions. Thus, in each step the candidates satisfy all the conditions that have been applied so far to generate them and they also do not violate the remaining conditions. This process can be visualized as a tree with the empty matrix at the root.

The above described steps are succinctly presented in Algorithm 1 in the form of a recursive depth first search. The function `getNextCondition()`, takes as input the current condition (`None` for the first call) and returns the next condition to apply. The function `generateCandidates()` takes as input a candidate matrix, and a condition and then derives all candidates that satisfy the given condition. The function `notViolateRestOfTheConditions()` takes as input a matrix and the current condition and returns True if the matrix does not violate any of the remaining conditions. This function can be modified to verify additional conditions when looking for specific kinds of shelves, such as connected or unitial. The algorithm is started by calling the function `getShelves()` with the empty initial matrix M_{init} and the first condition. Then `getShelves()` proceeds recursively in a depth first manner collecting the matrices that satisfy all the conditions of right distributivity, which are shelves.

A Python implementation of Algorithm 1, a list of shelves of order less than 6 (up to isomorphism), and a list of connected shelves of order less than seven (up to isomorphism) are available in [**7**]. This Python implementation also parallelizes

the algorithm for multiprocessing. The parallelization is achieved by first applying two of the n^3 conditions in a breadth first search, thus generating all the candidates that satisfy these two conditions and that do not violate remaining conditions and then parallelly applying the depth first search on these candidates. The order in which the conditions are applied does not affect the final result, however, through experimentation we found that the order very much affects the time to completion. Optimal ordering is left for future research.

Algorithm 1 Algorithm to construct shelves

```
n ← order                                                    ▷ n ≥ 2
M_init ← empty matrix of size n × n
c_1 ← getNextCondition(None)
L ← []                                                ▷ list to store shelves
procedure getShelves(M,c_in)                  ▷ recursively applies conditions
    c ← getNextCondition(c_in)
    if all conditions applied  & M not in L then
        append M to L
    else
        for m in generateCandidates(M, c_in) do
            if notViolateRestOfTheConditions(m, c_in) then
                getShelves(m, c)
            end if
        end for
    end if
end procedure
getShelves(M_init, c_1)                                  ▷ call the procedure
                ▷ when the procedure exits L contains all the shelves of order n
```

4. Connected Shelves

In this section, we classify connected shelves of order up to 5.

Definition 4.1. A connected shelf $(X, *)$ is a shelf such that for all $x, y \in X$, there exists a finite number of elements $x_1, x_2, \ldots, x_m$ such that

$$y = ((((x * x_1) * x_2) * x_3) \ldots) * x_m$$

The definition of a connected shelf simply means that we can go from one element to any other element by finite number of steps. In other words, the orbit of each element must equal the shelf itself.

n	# of connected shelves	# of connected racks	# of connected quandles
1	1	1	1
2	2	1	0
3	5	2	1
4	18	2	1
5	165	4	3
6	3987	4	2

We refer the reader to Appendix A for the complete list of connected shelves of order less than or equal to 5, up to isomorphism.

Our computer search results support the following conjecture for order up to 5.

Conjecture 4.2. *Let S be a shelf. Then there exists a shortest cycle that covers all the elements in S exactly once if and only if S is connected.*

A cycle from 0 to 0 that covers all elements in the shelf may contain an element more than once. For example, consider the following connected shelf of order 4.

*	0	1	2	3
0	0	1	1	3
1	0	1	2	0
2	0	1	2	0
3	0	1	1	3

A cycle from 0 to 0 in this shelf is

$$0 \to 1 \to 2 \to 1 \to 3 \to 0.$$

However the shortest cycle from 0 to 0 is

$$0 \to 3 \to 1 \to 2 \to 0.$$

4.1. Latin shelves. In this subsection, we explore the groups generated by rows in Latin shelves. Recall the left multiplication map. For each $x \in S$, the left multiplication by x is the map denoted by

$$L_x \, : \, S \to S$$

and given by

$$L_x(y) \, := \, x * y.$$

Definition 4.3. A shelf is *Latin* or *strongly connected* if the shelf operation is left-invertible. This means the rows of a Latin shelf are permutations of $\{1, 2, \ldots, n\}$.

Clearly, Latin shelves are connected, and a Latin quandle is always a Latin shelf. Let $z \in S$, where S is a Latin shelf.

$$L_{x*y}(z) = (x * y) * z = (x * z) * (y * z) = L_x(z) * L_y(z)$$

In a Latin shelf, each L_x, where $x \in S$, is a bijection. Let X be the set of all L_x, i.e.

$$X = \{L_x \, : \, x \in S\}.$$

Clearly, we have $X \subset S_n$.

Let $t \in S$. Define an operation $\triangleright$ on the set X as follows:

$$(L_x \triangleright L_y)(t) \, := \, L_x(t) * L_y(t).$$

Then

$$(L_x \triangleright L_y)(t) = L_{x*y}(t).$$

Now consider

$$\begin{aligned}((L_x \rhd L_y) \rhd L_z)(t) &= (L_x \rhd L_y)(t) * L_z(t)\\ &= (L_x(t) * L_y(t)) * L_z(t)\\ &= (L_x(t) * L_z(t)) * (L_y(t) * L_z(t))\\ &= (L_x \rhd L_z)(t) * (L_y \rhd L_z)(t)\\ &= L_{x*z}(t) * L_{y*z}(t)\\ &= (L_{x*z} \rhd L_{y*z})(t)\\ &= ((L_x \rhd L_z) \rhd (L_y \rhd L_z))(t)\end{aligned}$$

Since t is arbitrary, we have

$$(L_x \rhd L_y) \rhd L_z = (L_x \rhd L_z) \rhd (L_y \rhd L_z),$$

i.e.

$$L_{x*y} \rhd L_z = L_{x*z} \rhd L_{y*z}.$$

Thus, $(X, \rhd)$ is a shelf.

Note that $L_x \rhd L_x = L_{x*x}$. So, if the elements in the shelf S are idempotent, so are the elements in X.

Now we show that the elements in X do not satisfy the second axiom of a quandle. Let $t \in S$.

$$\begin{aligned}((L_x \rhd L_y) \rhd L_y)(t) &= (L_x \rhd L_y)(t) * L_y(t)\\ &= (L_x(t) * L_y(t)) * L_y(t)\\ &= (L_x(t) * L_y(t)) * (L_y(t) * L_y(t))\\ &= (L_x \rhd L_y)(t) * (L_y \rhd L_y)(t)\\ &= L_{x*y}(t) * L_{y*y}(t)\\ &= L_{(x*y)*(y*y)}(t) \neq L_x(t)\end{aligned}$$

For each $x \in S$, define a map

$$\phi : S \to X$$

given by

$$x \mapsto L_x$$

The map ϕ is a homomorphism because

$$\phi(x * y) = L_{x*y} = L_x \rhd L_y = \phi(x) \rhd \phi(y)$$

In fact, it is an epimorphism.

Now, let G be the group generated by L_x, where $x \in S$, i.e. $G = \langle L_x : x \in S \rangle$. Since we are only considering finite shelves, G is a finitely generated group.

Define an operation on G as

$$L_x \diamond L_y := L_y^{-1} L_x L_y$$

Then we have

$$\begin{aligned}(L_x \diamond L_y) \diamond L_z &= L_z^{-1}(L_x \diamond L_y) L_z\\ &= L_z^{-1}(L_y^{-1} L_x L_y) L_z\\ &= (L_y L_z)^{-1} L_x (L_y L_z)\end{aligned}$$

and

$$\begin{aligned}(L_x \diamond L_z) \diamond (L_y \diamond L_z) &= (L_z^{-1} L_x L_z) \diamond (L_z^{-1} L_y L_z) \\ &= (L_z^{-1} L_y L_z)^{-1} (L_z^{-1} L_x L_z)(L_z^{-1} L_y L_z) \\ &= (L_z^{-1} L_y^{-1} L_z)(L_z^{-1} L_x L_z)(L_z^{-1} L_y L_z) \\ &= (L_z^{-1} L_y^{-1}) L_x (L_y L_z) \\ &= (L_y L_z)^{-1} L_x (L_y L_z),\end{aligned}$$

which imply

$$(L_x \diamond L_y) \diamond L_z = (L_x \diamond L_z) \diamond (L_y \diamond L_z)$$

Hence the conjugation in G turns it into a shelf.

We are now interested in seeing whether the conjugation in G satisfies Axiom 1 and Axiom 2 of a quandle. Consider $L_x \diamond L_x$.

$$L_x \diamond L_x = L_x^{-1} L_x L_x = L_x$$

Thus the elements in G are idempotent.

Let's consider Axiom 2. We show that under certain conditions on left multiplication, elements in G satisfy Axiom 2. In other words, under certain conditions on left multiplication, G is a quandle.

$$\begin{aligned}(L_x \diamond L_y) \diamond L_y &= L_y^{-1} (L_x \diamond L_y) L_y \\ &= L_y^{-1} (L_y^{-1} L_x L_y) L_y \\ &= (L_y^{-1})^2 L_x (L_y)^2 \\ &= (L_y^2)^{-1} L_x (L_y^2)\end{aligned}$$

If rows are either 2-cycles or the identity permutation, then $(L_x \diamond L_y) \diamond L_y = L_x$, and thus $(G, \diamond)$ is a quandle. If cycles are disjoint, $(L_x \diamond L_y) \diamond L_y = L_x$, and thus $(G, \diamond)$ is still a quandle.

Since the rows of Latin shelves are permutations, we are curious about the groups generated by them. We are interested in knowing whether the group structure can be used to distinguish shelves in different isomorphism classes. In the following table, we present the groups generated by rows of Latin shelves of order up to 5. In the table, $S_{n,k}$ denotes the kth shelf of order n listed in Appendix A. The shelf $S_{3,3}$ denotes the Latin shelf # 3 listed under order 3 in Appendix A. We first list disjoint cycle notation for the rows of a Latin shelf, and then we list the group generated by them. The cycle (id) denotes the identity permutation.

Consider the following example:

$$S_{3,3} = [[0, 2, 1], [2, 1, 0], [1, 0, 2]]$$

The first, second and third rows are two cycles $(12), (02), (01)$, respectively. Thus the group generated by rows (left multiplications) is

$$G = \langle (12), (02), (01) \rangle = S_3 \cong D_3$$

In fact, this is the only connected Latin quandle of order 3. The abbreviation LQ stands for Latin Quandle.

Latin Shelf	Disjoint Cycle Notation for the Rows of the Latin Shelf	G
$S_{2,1}$	(id),(id)	{id}
$S_{3,1}$	(id),(id),(id)	{id}
$S_{3,2}$	(id),(id),(01)	$\mathbb{Z}_2$
$S_{3,3(LQ)}$	(12),(02),(01)	D_3
$S_{4,1}$	(id),(id),(id),(id)	{id}
$S_{4,2(LQ)}$	(132),(023),(031),(012)	A_4
$S_{4,3}$	(23),(23),(01),(01)	$\mathbb{Z}_2 \times \mathbb{Z}_2$
$S_{4,4}$	(id),(id),(01),(01)	$\mathbb{Z}_2$
$S_{4,7}$	(id),(id),(id),(021)	$\mathbb{Z}_3$
$S_{4,13}$	(id),(id),(id),(12)	$\mathbb{Z}_2$
$S_{5,126}$	(id),(id),(id),(id),(id)	{id}
$S_{5,127}$	(id),(id),(id),(id),(23)	$\mathbb{Z}_2$
$S_{5,129}$	(id),(id),(id),(id),(132)	$\mathbb{Z}_3$
$S_{5,133}$	(id),(id),(id),(id),(01)(23)	$\mathbb{Z}_2$
$S_{5,135}$	(id),(id),(id),(id),(0321)	$\mathbb{Z}_4$
$S_{5,136}$	(id),(id),(id),(12),(12)	$\mathbb{Z}_2$
$S_{5,139}$	(id),(id),(id),(12),(01)	D_3
$S_{5,140}$	(id),(id),(id),(12),(021)	D_3
$S_{5,150}$	(id),(id),(id),(021),(021)	$\mathbb{Z}_3$
$S_{5,151}$	(id),(id),(id),(021),(012)	$\mathbb{Z}_3$
$S_{5,152}$	(id),(id),(34),(01),(01)	$\mathbb{Z}_2 \times \mathbb{Z}_2$
$S_{5,153}$	(id),(id),(01),(01),(01)	$\mathbb{Z}_2$
$S_{5,154}$	(id),(id),(01),(01),(01)(23)	$\mathbb{Z}_2 \times \mathbb{Z}_2$
$S_{5,155}$	(id),(34),(34),(12),(12)	$\mathbb{Z}_2 \times \mathbb{Z}_2$
$S_{5,156}$	(34),(34),(34),(12),(12)	$\mathbb{Z}_2 \times \mathbb{Z}_2$
$S_{5,158}$	(34),(34),(34),(021),(021)	$\mathbb{Z}_2 \times \mathbb{Z}_3$
$S_{5,159}$	(34),(34),(01)(34),(01),(01)	$\mathbb{Z}_2 \times \mathbb{Z}_2$
$S_{5,162(LQ)}$	(12)(34),(03)(24),(13)(04),(02)(14),(01)(23)	D_5
$S_{5,163(LQ)}$	(1432),(0342),(0413),(0124),(0231)	$GA(1,5)$
$S_{5,164(LQ)}$	(1432),(0423),(0134),(0241),(0312)	$GA(1,5)$

We are able to distinguish many Latin shelves that belong to different isomorphism classes, but there are quite a few cases starting from order 4 in which shelves that belong to different isomorphism classes have the same group structure. For example, in order 4, the rows of Latin shelves $S_{4,4}$ and $S_{4,13}$ have the group structure $\mathbb{Z}_2$, and in order 5, the rows of Latin shelves $S_{5,152}$, $S_{5,154}$, $S_{5,155}$, $S_{5,156}$ and $S_{5,159}$ generate the group $\mathbb{Z}_2 \times \mathbb{Z}_2$.

It also seems like the number of such cases increases as the order gets higher. However, we believe that the group structure of the rows of Latin shelves would be an interesting topic for further research, especially as the order gets higher.

5. Shelf Polynomials

In this section, we study shelf polynomials.

Definition 5.1. Let S be a finite shelf. For each element $x \in S$, let $r(x)$ be the number of elements of S which act trivially on x, i.e. the set

$$r(x) = |\{y \in S \,|\, x * y = x\}|$$

and let $c(x)$ be the number of elements of S on which x acts trivially, i.e. the set

$$c(x) = |\{y \in S \,|\, y * x = y\}|$$

In terms of the shelf's operation table, $r(x)$ counts the number of xs in row x and $c(x)$ counts the number of entries in the column of x equal their row number.

For every element $x \in S$, we have a pair $(r(x), c(x))$ of integers. We can express this data as a polynomial in two variables which we call the *shelf polynomial* of S:

$$P(S) = \sum_{x \in S} t^{r(x)} s^{c(x)}.$$

Example 5.2. *The shelf*

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

has the shelf polynomial $P(S) = 4st$

Definition 5.3. A shelf is *Latin* or *strongly connected* if the shelf operation is left-invertible. This means the rows of a Latin shelf are permutations of $\{1, 2, \ldots, n\}$.

We now classify shelves as latin quandles, non-quandle racks, non-rack latin shelves or non-rack shelves. The numbers in parentheses are the numbers assigned to shelves listed in Appendix A. We also present the shelf polynomial of shelves in Appendix A.

Order 2
- Non-rack shelves: (1)
- Non-quandle Racks: (2)

There are neither non-rack Latin shelves nor Latin quandles of order 2. The shelf polynomials of shelves (1) and (2) are $P(S) = 2st$ and $P(S) = 2$, respectively.

Order 3
- Non-rack shelves: (5)
- Non-rack Latin shelves: (1), (2)
- Non-quandle Racks: (4)
- Latin Quandles: (3)

The shelf polynomial of the shelf (non-quandle rack) (4) is the constant polynomial $P(S) = 3$, and the shelf polynomial of all other shelves is $P(S) = 3st$.

Order 4
- Non-rack shelves: (6), (8) – (12), (14) – (18)
- Non-rack Latin shelves: (1), (3), (4), (7), (13)
- Non-quandle Racks: (5)
- Latin Quandles: (2)

The shelf polynomial of the shelves (5) and (6) is the constant polynomial $P(S) = 4$, and the shelf polynomial of all other shelves is $P(S) = 4st$.

We note to the reader that if $P(S) = |S|st$, then S is clearly idempotent. Then the natural question to ask is "Is the converse true?". This leads to the following conjecture strongly supported by our computational results.

Conjecture 5.4. *If a connected shelf* S *is idempotent, then* $P(S) = |S|st$*, where* $|S|$ *is the cardinality of* S*.*

We note to the reader that an idempotent shelf does not always have the shelf polynomial $P(S) = |S|st$.

For example, the order 3 idempotent shelf, which is non-connected,

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

has the shelf polynomial $P(S) = 3st$, whereas the order 3 idempotent shelf, which is non-connected,

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

has the shelf polynomial $P(S) = 3s^3t^3$. The conjecture says that the shelf polynomial of a connected spindle S is $P(S) = |S|st$, where $|S|$ is the cardinality of S.

Appendix A

In Appendix A, we list all connected shelves of order less than or equal to 5 up to isomorphism. For a full list of shelves of order less than or equal to 5 up to isomorphism and a full list of connected shelves of order less than or equal to six up to isomorphism, we refer the reader to [**7**].

Order 2: There are two connected shelves.

(1) [[0, 1], [0, 1]]

(2) [[1, 1], [0, 0]]

Order 3: There are five connected shelves.

(1) [[0, 1, 2], [0, 1, 2], [0, 1, 2]]

(2) [[0, 1, 2], [0, 1, 2], [1, 0, 2]]

(3) [[0, 2, 1], [2, 1, 0], [1, 0, 2]]

(4) [[1, 1, 1], [2, 2, 2], [0, 0, 0]]

(5) [[0, 1, 1], [0, 1, 2], [0, 1, 2]]

Order 4: There are 18 connected shelves.

(1) [[0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3]]

(2) [[0, 2, 3, 1], [3, 1, 0, 2], [1, 3, 2, 0], [2, 0, 1, 3]]

(3) [[0, 1, 3, 2], [0, 1, 3, 2], [1, 0, 2, 3], [1, 0, 2, 3]]

(4) [[0, 1, 2, 3], [0, 1, 2, 3], [1, 0, 2, 3], [1, 0, 2, 3]]

(5) [[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [0, 0, 0, 0]]

(6) [[1, 1, 2, 2], [0, 0, 3, 3], [0, 0, 3, 3], [1, 1, 2, 2]]

(7) [[0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [1, 2, 0, 3]]

(8) [[0, 1, 1, 1], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3]]

(9) [[0, 1, 1, 3], [0, 1, 2, 0], [0, 1, 2, 0], [0, 1, 1, 3]]

(10) [[0, 1, 1, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 1, 3]]

(11) [[0, 1, 1, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 2, 2, 3]]

(12) [[0, 1, 1, 3], [3, 1, 2, 0], [3, 1, 2, 0], [0, 1, 1, 3]]

(13) [[0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 2, 1, 3]]

(14) [[0, 1, 1, 1], [0, 1, 2, 2], [0, 1, 2, 3], [0, 1, 2, 3]]

(15) [[0, 1, 1, 2], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3]]

(16) [[0, 1, 1, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3]]

(17) [[0, 1, 1, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 2, 1, 3]]

(18) [[0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [1, 0, 0, 3]]

Order 5: There are 165 connected shelves.

(1) [[0, 1, 1, 1, 1], [0, 1, 2, 2, 2], [0, 1, 2, 3, 3], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

(2) [[0, 1, 1, 1, 1], [0, 1, 2, 2, 2], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

(3) [[0, 1, 1, 1, 1], [0, 1, 2, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 2, 4]]

(4) [[0, 1, 1, 1, 1], [0, 1, 2, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

(5) [[0, 1, 1, 1, 1], [0, 1, 2, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 3, 2, 4]]

(6) [[0, 1, 1, 1, 1], [0, 1, 2, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 3, 3, 4]]

(7) [[0, 1, 1, 1, 1], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

(8) [[0, 1, 1, 1, 2], [0, 1, 2, 3, 3], [0, 1, 2, 3, 3], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

(9) [[0, 1, 1, 1, 2], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

(10) [[0, 1, 1, 1, 2], [0, 1, 2, 4, 4], [0, 1, 2, 4, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

(11) [[0, 1, 1, 1, 4], [0, 1, 2, 2, 0], [0, 1, 2, 3, 0], [0, 1, 2, 3, 0], [0, 1, 1, 1, 4]]

(12) [[0, 1, 1, 1, 4], [0, 1, 2, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 1, 4]]

(13) [[0, 1, 1, 1, 4], [0, 1, 2, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 2, 4]]

(14) [[0, 1, 1, 1, 4], [0, 1, 2, 3, 0], [0, 1, 2, 3, 0], [0, 1, 2, 3, 0], [0, 1, 1, 1, 4]]

(15) [[0, 1, 1, 1, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 1, 4]]

(16) [[0, 1, 1, 1, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 2, 4]]

(17) [[0, 1, 1, 1, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 3, 4]]

(18) [[0, 1, 1, 1, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 2, 4]]

(19) [[0, 1, 1, 1, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

(20) [[0, 1, 1, 1, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 3, 2, 4]]

(21) [[0, 1, 1, 1, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 1, 4]]

(22) [[0, 1, 1, 1, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 2, 4]]

(23) [[0, 1, 1, 1, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 3, 4]]

(24) [[0, 1, 1, 1, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 2, 1, 4]]

(25) [[0, 1, 1, 1, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 2, 2, 4]]

(26) [[0, 1, 1, 1, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 2, 3, 4]]

(27) [0, 1, 1, 1, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 3, 1, 4]]

(28) [0, 1, 1, 1, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 3, 2, 4]]

(29) [0, 1, 1, 1, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 3, 3, 4]]

(30) [0, 1, 1, 1, 4], [4, 1, 2, 2, 0], [4, 1, 2, 3, 0], [4, 1, 2, 3, 0], [0, 1, 1, 1, 4]]

(31) [0, 1, 1, 1, 4], [4, 1, 2, 3, 0], [4, 1, 2, 3, 0], [4, 1, 2, 3, 0], [0, 1, 1, 1, 4]]

(32) [0, 1, 1, 2, 2], [0, 1, 2, 3, 3], [0, 1, 2, 3, 3], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

(33) [0, 1, 1, 2, 2], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

(34) [0, 1, 1, 2, 3], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

(35) [0, 1, 1, 2, 4], [0, 1, 2, 3, 0], [0, 1, 2, 3, 0], [0, 1, 2, 3, 0], [0, 1, 1, 2, 4]]

(36) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 2, 4]]

(37) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 3, 4]]

(38) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 1, 4]]

(39) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 2, 4]]

(40) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

(41) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 3, 1, 4]]

(42) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 3, 2, 4]]

(43) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 3, 3, 4]]

(44) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 1, 4], [0, 2, 1, 1, 4]]

(45) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 2, 4], [0, 2, 1, 2, 4]]

(46) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 3, 4], [0, 2, 1, 3, 4]]

(47) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 2, 1, 4], [0, 2, 2, 1, 4]]

(48) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 2, 3, 4], [0, 2, 2, 3, 4]]

(49) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 3, 1, 4], [0, 2, 3, 1, 4]]

(50) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 3, 2, 4], [0, 2, 3, 2, 4]]

(51) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 3, 3, 4], [0, 2, 3, 3, 4]]

(52) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 3, 1, 1, 4]]

(53) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 3, 1, 2, 4]]

(54) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 3, 1, 3, 4]]

(55) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 3, 2, 1, 4]]

(56) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 3, 2, 3, 4]]

(57) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 3, 3, 1, 4]]

(58) [0, 1, 1, 2, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 3, 3, 2, 4]]

(59) [0, 1, 1, 2, 4], [4, 1, 2, 3, 0], [4, 1, 2, 3, 0], [4, 1, 2, 3, 0], [0, 1, 1, 2, 4]]

(60) [0, 1, 1, 3, 3], [0, 1, 2, 3, 2], [0, 1, 2, 3, 4], [0, 1, 2, 3, 2], [0, 1, 2, 3, 4]]

(61) [0, 1, 1, 3, 3], [0, 1, 2, 3, 3], [0, 1, 2, 3, 3], [0, 1, 1, 3, 4], [0, 1, 1, 3, 4]]

(62) [0, 1, 1, 3, 3], [0, 1, 2, 3, 3], [0, 1, 2, 3, 3], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

(63) [0, 1, 1, 3, 3], [0, 1, 2, 3, 3], [0, 1, 2, 3, 3], [0, 2, 1, 3, 4], [0, 2, 1, 3, 4]]

(64) [0, 1, 1, 3, 3], [0, 1, 2, 3, 3], [0, 1, 2, 3, 3], [0, 2, 2, 3, 4], [0, 2, 2, 3, 4]]

(65) [0, 1, 1, 3, 3], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

(66) [0, 1, 1, 3, 3], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 3, 4], [0, 2, 1, 3, 4]]

(67) [0, 1, 1, 3, 3], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 2, 3, 4], [0, 2, 2, 3, 4]]

(68) [0, 1, 1, 3, 3], [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [0, 2, 1, 3, 4], [0, 2, 1, 3, 4]]

(69) [0, 1, 1, 3, 3], [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [0, 2, 2, 3, 4], [0, 2, 2, 3, 4]]

(70) [0, 1, 1, 3, 3], [0, 1, 2, 4, 4], [0, 1, 2, 4, 4], [0, 2, 2, 3, 4], [0, 2, 2, 3, 4]]

(71) [0, 1, 1, 3, 4], [0, 1, 2, 0, 4], [0, 1, 2, 0, 4], [0, 1, 1, 3, 4], [0, 1, 1, 3, 4]]

(72) [0, 1, 1, 3, 4], [0, 1, 2, 0, 4], [0, 1, 2, 0, 4], [0, 1, 1, 3, 4], [0, 1, 2, 3, 4]]

(73) [0, 1, 1, 3, 4], [0, 1, 2, 0, 4], [0, 1, 2, 0, 4], [0, 1, 1, 3, 4], [0, 2, 1, 0, 4]]

(74) [0, 1, 1, 3, 4], [0, 1, 2, 0, 4], [0, 1, 2, 0, 4], [0, 1, 1, 3, 4], [0, 2, 1, 3, 4]]

(75) [0, 1, 1, 3, 4], [0, 1, 2, 0, 4], [0, 1, 2, 0, 4], [0, 1, 1, 3, 4], [0, 2, 2, 3, 4]]

(76) [0, 1, 1, 3, 4], [0, 1, 2, 0, 4], [0, 1, 2, 0, 4], [0, 1, 1, 3, 4], [3, 2, 1, 0, 4]]

(77) [0, 1, 1, 3, 4], [0, 1, 2, 0, 4], [0, 1, 1, 3, 4], [3, 2, 1, 3, 4]

(78) [0, 1, 1, 3, 4], [0, 1, 2, 3, 3], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]

(79) [0, 1, 1, 3, 4], [0, 1, 2, 3, 3], [0, 2, 1, 3, 4], [0, 2, 1, 3, 4]

(80) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 3, 4], [0, 1, 1, 3, 4]

(81) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 3, 4], [0, 1, 2, 3, 4]

(82) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 3, 4], [0, 2, 1, 0, 4]

(83) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 3, 4], [0, 2, 1, 3, 4]

(84) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 3, 4], [0, 2, 2, 3, 4]

(85) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 3, 4], [3, 0, 0, 0, 4]

(86) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 3, 4], [3, 0, 3, 0, 4]

(87) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 3, 4], [3, 1, 1, 0, 4]

(88) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 3, 4], [3, 1, 2, 0, 4]

(89) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 3, 4], [3, 2, 1, 0, 4]

(90) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 1, 3, 4], [3, 2, 2, 0, 4]

(91) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 1], [0, 1, 2, 3, 4]

(92) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 2], [0, 1, 2, 3, 4]

(93) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]

(94) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 3, 2, 4]

(95) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 3, 3, 4]

(96) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 1, 4]

(97) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 2, 4]

(98) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 3, 4]

(99) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 2, 3, 4]

(100) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 3, 1, 4]

(101) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 3, 2, 4]

(102) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]

(103) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]

(104) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 3, 2, 1, 4]

(105) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 3, 3, 1, 4]

(106) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 3, 3, 2, 4]

(107) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 3, 4]

(108) [0, 1, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 2, 3, 4]

(109) [0, 1, 1, 3, 4], [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [0, 1, 1, 3, 4]

(110) [0, 1, 1, 3, 4], [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [0, 1, 2, 3, 4]

(111) [0, 1, 1, 3, 4], [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [0, 2, 1, 3, 4]

(112) [0, 1, 1, 3, 4], [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [0, 2, 2, 3, 4]

(113) [0, 1, 1, 3, 4], [3, 1, 2, 0, 0], [3, 1, 2, 0, 0], [0, 1, 1, 3, 4]

(114) [0, 1, 1, 3, 4], [3, 1, 2, 0, 4], [3, 1, 2, 0, 4], [0, 1, 1, 3, 4]

(115) [0, 1, 1, 3, 4], [3, 1, 2, 0, 4], [3, 1, 2, 0, 4], [0, 2, 1, 0, 4]

(116) [0, 1, 1, 3, 4], [3, 1, 2, 0, 4], [3, 1, 2, 0, 4], [0, 2, 1, 3, 4]

(117) [0, 1, 1, 3, 4], [3, 1, 2, 0, 4], [3, 1, 2, 0, 4], [3, 1, 1, 0, 4]

(118) [0, 1, 1, 3, 4], [3, 1, 2, 0, 4], [3, 1, 2, 0, 4], [3, 1, 2, 0, 4]

(119) [0, 1, 1, 3, 4], [3, 1, 2, 0, 4], [3, 1, 2, 0, 4], [3, 2, 1, 0, 4]

(120) [0, 1, 1, 3, 4], [3, 1, 2, 0, 4], [3, 1, 2, 0, 4], [3, 2, 2, 0, 4]

(121) [0, 1, 1, 3, 4], [3, 1, 2, 4, 0], [3, 1, 2, 4, 0], [0, 1, 1, 3, 4]

(122) [0, 1, 1, 4, 3], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]

(123) [0, 1, 1, 4, 3], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 3, 4]

(124) [0, 1, 1, 4, 3], [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [0, 1, 2, 3, 4]

(125) [0, 1, 1, 4, 3], [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [0, 2, 1, 3, 4]

(126) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]

(127) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 3, 2, 4]]

(128) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 1, 4]]

(129) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 3, 1, 4]]

(130) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 0, 0, 0, 4]]

(131) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 0, 0, 1, 4]]

(132) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 0, 0, 2, 4]]

(133) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 0, 3, 2, 4]]

(134) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 2, 0, 0, 4]]

(135) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 2, 3, 0, 4]]

(136) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 3, 4], [0, 2, 1, 3, 4]]

(137) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 3, 4], [1, 0, 0, 3, 4]]

(138) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 3, 4], [1, 0, 1, 3, 4]]

(139) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 3, 4], [1, 0, 2, 3, 4]]

(140) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 3, 4], [1, 2, 0, 3, 4]]

(141) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 3, 4], [1, 2, 1, 3, 4]]

(142) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 2, 1, 3, 4], [3, 3, 3, 0, 4]]

(143) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 0, 0, 3, 4], [1, 0, 0, 3, 4]]

(144) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 0, 0, 3, 4], [1, 0, 1, 3, 4]]

(145) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 0, 0, 3, 4], [1, 2, 0, 3, 4]]

(146) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 0, 0, 3, 4], [1, 2, 1, 3, 4]]

(147) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 0, 0, 3, 4], [2, 0, 0, 3, 4]]

(148) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 0, 0, 3, 4], [2, 0, 1, 3, 4]]

(149) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 0, 0, 3, 4], [2, 2, 1, 3, 4]]

(150) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 2, 0, 3, 4], [1, 2, 0, 3, 4]]

(151) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 2, 0, 3, 4], [2, 0, 1, 3, 4]]

(152) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 4, 3], [1, 0, 2, 3, 4], [1, 0, 2, 3, 4]]

(153) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 0, 2, 3, 4], [1, 0, 2, 3, 4], [1, 0, 2, 3, 4]]

(154) [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 0, 2, 3, 4], [1, 0, 2, 3, 4], [1, 0, 3, 2, 4]]

(155) [0, 1, 2, 3, 4], [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [0, 2, 1, 3, 4], [0, 2, 1, 3, 4]]

(156) [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [0, 2, 1, 3, 4], [0, 2, 1, 3, 4]]

(157) [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [1, 0, 0, 3, 4], [1, 0, 0, 3, 4]]

(158) [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [1, 2, 0, 3, 4], [1, 2, 0, 3, 4]]

(159) [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [1, 0, 2, 4, 3], [1, 0, 2, 3, 4], [1, 0, 2, 3, 4]]

(160) [0, 1, 2, 4, 3], [0, 1, 2, 4, 3], [1, 0, 2, 4, 3], [2, 2, 0, 3, 4], [2, 2, 1, 3, 4]]

(161) [0, 1, 3, 2, 2], [0, 1, 4, 2, 2], [3, 4, 2, 0, 1], [2, 2, 0, 3, 4], [2, 2, 1, 3, 4]]

(162) [0, 2, 1, 4, 3], [3, 1, 4, 0, 2], [4, 3, 2, 1, 0], [2, 4, 0, 3, 1], [1, 0, 3, 2, 4]]

(163) [0, 2, 3, 4, 1], [2, 1, 4, 0, 3], [3, 4, 2, 1, 0], [4, 0, 1, 3, 2], [1, 3, 0, 2, 4]]

(164) [0, 2, 3, 4, 1], [3, 1, 4, 2, 0], [4, 0, 2, 1, 3], [1, 4, 0, 3, 2], [2, 3, 1, 0, 4]]

(165) [1, 1, 1, 1, 1], [2, 2, 2, 2, 2], [3, 3, 3, 3, 3], [4, 4, 4, 4, 4], [0, 0, 0, 0, 0]]

References

[1] J. S. Carter, A. S. Crans, M. Elhamdadi and M. Saito, *Cohomology of categorical self-distributivity*, J. Homotopy Relat. Struct. **3** (2008), 13–63. Doi: 10.4303/jglta/s070102.

[2] A. S. Crans, S. Mukherjee and J. H. Przytycki, *On homology of associative shelves*, J. Homotopy Relat. Struct. **12** (2017), 741–763. Doi: 10.1007/s40062-016-0164-9.

[3] P. Dehornoy, *Braids and self-distributivity*, Progress in Mathematics 192, Birkhäuser Verlag, Basel, 2000. Doi: 10.1007/978-3-0348-8442-6.

[4] M. Elhamdadi, J. Macquarrie and R. Restrepo, *Automorphism groups of quandles*, J. Algebra Appl. **11** (2012), 1250008, 9. Doi: 10.1142/S0219498812500089.

[5] M. Elhamdadi and S. Nelson, *Quandles—an introduction to the algebra of knots*, Student Mathematical Library 74, American Mathematical Society, Providence, RI, 2015. Doi: 10.1090/stml/074.

[6] R. Fenn and C. Rourke, *Racks and links in codimension two*, J. Knot Theory Ramifications **1** (1992), 343–406. Doi: 10.1142/S0218216592000203.

[7] M. Goonewardena, *shelf*, Github, 2023. Available at `https://github.com/mathrep/shelf`.

[8] J. Ježek, *Enumerating left distributive groupoids*, Czechoslovak Math. J. **47(122)** (1997), 717–727. Doi: 10.1023/A:1022826819995.

[9] R. Laver, *On the algebra of elementary embeddings of a rank into itself*, Adv. Math. **110** (1995), 334–346. Doi: 10.1006/aima.1995.1014.

Mohamed Elhamdadi
Department of Mathematics and Statistics,
University of South Florida,
4202 E. Fowler Avenue
Tampa, FL 33620
USA
emohamed@usf.edu

Neranga Fernando
Department of Mathematics,
Knox College,
2 E South St
Galesburg, IL 61401
USA
nfernando@knox.edu

Mathew Goonewardena
Ericsson,
Montreal, QC,
Canada
mathew.goonewardena@ericsson.com