Int J Data Sci Anal (2016) 1:89–98
DOI 10.1007/s41060-016-0007-0

REGULAR PAPER

A mining-based approach for efficient enumeration of algebraic structures

Majid Ali Khan[1] · **Nazeeruddin Mohammad**[1] · **Shahabuddin Muhammad**[1] · **Asif Ali**[2]

Received: 7 March 2016 / Accepted: 22 March 2016 / Published online: 13 April 2016

Abstract Algebraic structures are well-studied mathematical structures in abstract algebra with applications in many fields of computer security such as cryptography and authentication. Generating such structures is computationally very expensive because of the huge number of permutations. Also, many of these permutations are redundant as they are symmetrically equivalent. The symmetry breaking (finding symmetrically equivalent structures) is also a computationally challenging task. In this paper, we present a mining-based approach for symmetry breaking in algebraic structures. The approach reduces the number of redundant structures by identifying rules based on recurring patterns in the previously known structures. These rules are then used as constraints in a constraint solver. The proposed approach is applied to IP loop, a special class of algebraic structures, and the deduced rules have eliminated a large number of redundant solutions resulting in significant time improvement.

Keywords Algebraic structures · Mining rules · Symmetry breaking · Inverse property loop

This paper is an extension of our earlier paper published in IEEE International Conference on Data Science and Advanced Analytics (DSAA'15) [1].

✉ Majid Ali Khan
makhan@pmu.edu.sa; majidkk@gmail.com

1 College of Computer Engineering and Science, Prince Mohammad Bin Fahd University, Khobar, Kingdom of Saudi Arabia

2 Department of Mathematics, Quaid-i-Azam University, Islamabad, Pakistan

1 Introduction

Algebraic structures are of interest to mathematicians because of their special properties and also have applications in different areas such as cryptography [3,4]. An algebraic structure refers to a set with one or more finite operations defined on it. Quasigroups (Latin squares), loops, and IP loops are some of the widely studied algebraic structures [2]. A *quasigroup* is similar to group, but without the requirement of associativity. In other words, a quasigroup $(S, *)$ is a groupoid S with a binary operation $*$ such that for each x, $y \in S$, $x * a = y$ and $b * x = y$ have unique solutions. The multiplication table of a finite quasigroup is called a *Latin square*. A *loop* is a quasigroup with an identity element e such that for each $x \in S$, $x * e = x = e * x$. A loop L is called an *inverse property (IP) loop* if it has a two-sided inverse x^{-1} such that $x^{-1} * (x * y) = y = (y * x) * x^{-1}$ for each x, $y \in L$.

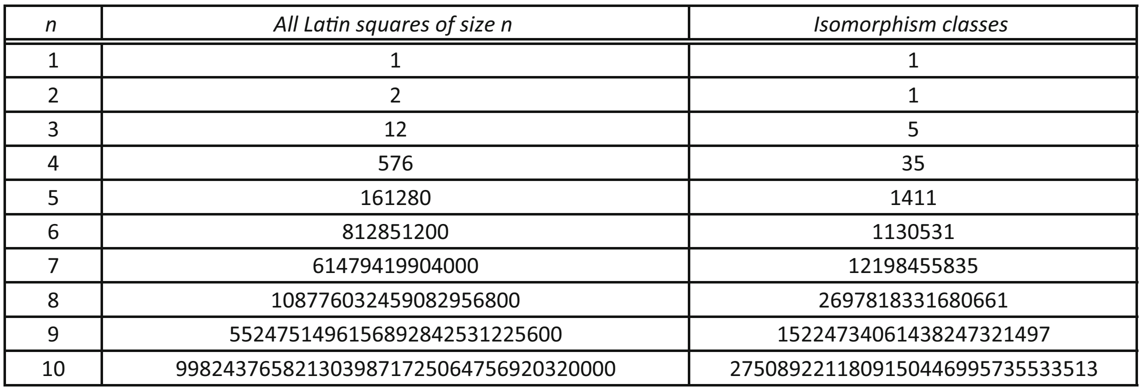
A simple way to count and enumerate algebraic structures of any order is to model them as a finite domain constraint satisfaction problem (CSP), where the range of the binary operation $*$ is a CSP variable whose domain consists of elements of the algebra. Then depending on the required algebraic structure, the corresponding constraint is applied on CSP variables. CSP constraints for Latin Square, loop, and IP loop properties are shown in Table 1. Constraint solver explores the state space in order to find all possible solutions that satisfy the specified constraints.

It is well known that constraint satisfaction problems have symmetries, that is, for every solution there are many equivalent solutions [5,6]. For example, there are 161280 Latin squares of order 5, of which only 1411 isomorphism classes[1]

[1] Please refer to Sect. 2 for background information about constraint programming and isomorphism classes.

Table 1 Constraints for generating algebraic structures

Name	Constraint
Latin square	$\forall row : \forall i, j \in row, x_i = x_j \Rightarrow i = j$
	$\forall col : \forall i, j \in col, y_i = y_j \Rightarrow i = j$
Loop	$x * e = x = e * x$
IP loop	$\forall x, y \in L : x^{-1} * (x * y) = (y * x) * x^{-1} = y$
Basic symmetry breaking in IP loop	$\|x - x^{-1}\| \leq 1$
Isomorphism	$*_1$ Isom.$*_2 \Leftrightarrow \forall u, v \in *_1, f(u *_1 v) = f(u) *_2 f(v)$



n	*All Latin squares of size n*	*Isomorphism classes*
1	1	1
2	2	1
3	12	5
4	576	35
5	161280	1411
6	812851200	1130531
7	61479419904000	12198455835
8	108776032459082956800	2697818331680661
9	5524751496156892842531225600	15224734061438247321497
10	9982437658213039871725064756920320000	2750892211809150446995735533513

Fig. 1 Number of Latin squares and isomorphism classes

exist. To show the enormity of redundant copies, the number of Latin squares and unique isomorphism classes up to order 10 are shown in Fig. 1 [2]. When enumerating algebraic structures, it is sufficient to find only one solution from each class of equivalent solutions. To reduce the search time of constraint solvers, it is better to break symmetries during the search so that redundant search efforts can be avoided. Therefore, additional constraints for symmetry breaking such as those proposed by [7] are added. Generating these symmetric breaking constraints is a time-consuming process and requires a good insight into the problem domain.

Even after applying symmetric breaking constraints, the solutions generated by the constraint solver have enormous number of isomorphic copies. These redundant isomorphic copies need to be eliminated in order to get the count of isomorphism classes. These isomorphic copies are eliminated in a separate post-processing step (henceforth referred to as isomorphism detection) using tools such as Nauty [8]. In this paper, we use a mining-based approach to discover more symmetric breaking constraints. To the best of our knowledge, mining-based approaches have never been used to generate symmetric breaking rules. We demonstrate that new rules can be generated without expertise in specific algebraic structures. To prove the effectiveness of our approach, we apply the proposed approach on IP loops of order *13* and show performance improvements in enumerating IP loops. Inclusion of constraints discovered by our mining process provides two benefits: (1) it cuts down the search space for a constraint solver, which reduces the search time; (2) it minimizes the number of redundant copies, which reduces isomorphism detection time.

The rest of the paper is organized as follows. Section 2 describes the related background information for constraint programming and the history of counting algebraic structures. Section 3 explains the methodology used to extract the symmetric breaking constraints. Section 4 discusses the results obtained by applying our approach to one of the well known algebraic structures. Section 5 provides conclusion and future directions.

2 Background

This section describes the background information about algebraic structure enumeration using constraint solvers, explains isomorphism classes, and describes the history of algebraic structure enumeration.

2.1 Constraint solvers

Constraint programming (CP) is a paradigm where a problem can be modeled in terms of constraints in order to identify feasible solutions from a huge set of possible solutions. CP focuses on finding feasible solutions instead of optimal solutions. CP has been applied in several domains including computer graphics, natural language processing, scheduling, and planning. There are several free and commercial constraint programming solvers available which allow users to model problems in terms of constraints.

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

(a) Latin Square

*	0	1	2	3	4
e=0	0	1	2	3	4
1	1	0	3	4	2
2	2	3	4	1	0
3	3	4	0	2	1
4	4	2	1	0	3

(b) Loop

*	0	1	2	3	4
e=0	0	1	2	3	4
1	1	3	0	4	2
2	2	0	4	1	3
3	3	4	1	2	0
4	4	2	3	0	1

x	x^{-1}
0	0
1	2
2	1
3	4
4	3

(c) IP Loop

Fig. 2 Examples of algebraic structures

In this paper, we use JaCoP and Google's or-tools to enumerate algebraic structures. Constraint solver models the problem as finite domain constraint satisfaction problem (CSP), where the range of the binary operation $*$ is a CSP variable whose domain consists of elements of the algebra. Then the relevant constraints for the algebraic structures (Latin square, loop, and IP loop), as shown in Table 1, are applied on CSP variables.

Latin square constraint in Table 1 implies that each symbol (element) occurs only once in any row and in any column. An example of Latin square of order 5 is shown in Fig. 2a. The loop constraint enforces existence of identity element (e) such that a binary operation ($*$) between e and any other element (x) results in the same element (x). For example, in Fig. 2b $e = 0$ and $(2 * 0) = (0 * 2) = 2$, whereas $(2 * 0) \neq (0 * 2) \neq 2$ in Latin square shown in Fig. 2a. The IP loop constraint implies existence of left and right inverses (x^{-1}) such that $x^{-1} * x = e$ and $x * x^{-1} = e$ and holds left inverse property ($x^{-1} * (x * y) = y$) and right inverse property ($(y * x) * x^{-1} = y$) for each element of the loop. For example in Fig. 2c, the inverse of element 1 is 2, and $2 * (1 * 3) = (3 * 1) * 2 = 3$. On the contrary in Fig. 2b, the inverse of element 1 is 1, but $1*(1*3) \neq 3$ and $(3*1)*1 \neq 3$.

Once the constraints are applied, constraint solvers explore the state space in order to find all possible solutions that satisfy the specified constraints.

2.2 Isomorphism Classes

Given two algebraic structures (e.g., Latin squares, loops or IP loops) $(L_1, *_1)$ and $(L_2, *_2)$, these structures are considered *isomorphic* to each other if there exists a bijective function $f : A \rightarrow B$, where $A = \{0, \ldots, n-1\}$ and B is any permutation of A, such that for all indices u and v in L_1:$f(u *_1 v) = f(u) *_2 f(v)$. In our case, L_1 $(n \times n)$ is isomorphic to L_2 $(n \times n)$ if $\forall i, j < n$, $f(L_1[i][j]) = L_2[f(i)][f(j)]$. All those structures that are isomorphic to each other belong to one *isomorphism class*.

For example, Fig. 3 shows two IP loops L_1 and L_2, which look quite different from each other (as highlighted). But they are isomorphic to each other because there exists a bijective function, $f : \{0, 1, 2, 3, 4, 5, 6\} \rightarrow \{0, 1, 2, 4, 3, 5, 6\}$ that satisfies the isomorphism property for each element of L_1 and

0	1	2	3	4	5	6
1	2	0	5	6	4	3
2	0	1	6	5	3	4
3	6	5	4	0	1	2
4	5	6	0	3	2	1
5	3	4	2	1	6	0
6	4	3	1	2	0	5

$f : (3\ \ 4)$

$\approx$

0	1	2	3	4	5	6
1	2	0	6	5	3	4
2	0	1	5	6	4	3
3	5	6	4	0	2	1
4	6	5	0	3	1	2
5	4	3	1	2	6	0
6	3	4	2	1	0	5

Fig. 3 Isomorphism example: IP loop of order 7 (L_1 on the *left* and L_2 on the *right*)

L_2. Please note that $f(0) = 0$, $f(1) = 1$, $f(2) = 2$, $f(3) = 4$, $f(4) = 3$, $f(5) = 5$, and $f(6) = 6$. For example, it can be seen that at indices $(i, j) = (1, 3)$, isomorphism property is satisfied because $f(L_1[1][3]) = L_2[f(1)][f(3)] = 5$.

We can also describe the above bijective function f as $f : (3\ \ 4)$, which means that symbols 3 and 4 are swapped. Another way to check isomorphism between two algebraic structures L_1 and L_2 is to generate L_2 from L_1 by swapping particular rows, columns, and the values according to the function f. For example, in Fig. 3, L_2 can be generated from L_1 by swapping rows 3 and 4, column 3 and 4, and values 3 and 4.

Finding isomorphism in this way, by applying the above formula for all permutations of f is extremely time-consuming and involves huge number of possibilities for even slightly large n.

2.3 History of algebraic structures enumeration

It is known that researchers had interest in counting and enumerating algebraic structures for over three centuries. Latin squares, loops, and IP loops are some of the well-studied algebraic structures. Earliest history of counting Latin squares (LS) goes back to at least 1782 as the number of reduced LS of order 5 was known to Euler [14] and Cayley [13]. Since that time, the researchers have been trying to get the next order algebraic structures. However, there has been considerable delay in achieving the consecutive milestones. This was because of computational complexity of the problem. The history of counting reduced Latin squares and loops is summarized in Table 2. This table shows the main achievements and the related studies. Additionally, there are numerous other studies [12,17,18,21] on counting algebraic structures, which produced incorrect counts.

Table 2 History of counting Latin squares and loops

Key milestones in Latin square (LS) and loops counting	Historical study details
Reduced LS up to $N=5$	Euler (1782) [14]
Reduced LS up to $N=6$	Frolov (1890) [16]
Isotopy classes up to $N=6$	Fisher and Yates (1934) [15]
Loops up to $N=6$	Schonhardt (1930) [25], Albert (1944) [9] and Sade (1970) [23]
Main & isotopy classes for $N=7$	Sade (1951) [22], Saxena (1951) [24]
Loops up to $N=7$	Brant and Mullen (1985) [11]
Reduced LS for $N=8$	Wells (1967) [26]
Reduced LS for $N=9$	Bammel and Rathstein (1975) [10]
Reduced LS for $N=10$	McKay and Rogoyski (1995) [19]
Reduced LS for $N=11$	McKay and Wanless (2005) [20]
Loops and LS up to $N=10$	McKay, Meynert and Myrvold (2007) [2]
Inverse property loops up to $N=13$	Slaney and Ali (2008) [7]

In this paper, we demonstrate the application of mining techniques in order to reduce the time in enumerating algebraic structures.

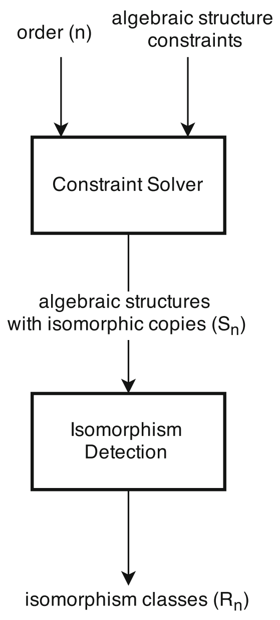
3 Proposed methodology

We propose an approach to find symmetry breaking constraints by mining rules from the previously known solutions of lower-order algebraic structures. These constraints can then be used for efficient enumeration of algebraic structures of higher order. For example, we can find rules by applying a mining technique on known set of matrices for algebraic structures of order 1 to n. The discovered rules can then be used for enumerating the solutions for algebraic structures of order $(n + 1)$.

Thus, the first step in our proposed methodology is to enumerate the required algebraic structures and their respective isomorphism classes. These algebraic structures and the isomorphism classes are then used in the second step to identify rules in the form of symmetry breaking constraints. These rules can then be used to enumerate algebraic structure of higher order. The following subsections describe the details of these steps.

3.1 Enumerating algebraic structures

The steps used for generating the algebraic structures of any order n is shown in Fig. 4. We consider a matrix (our algebraic structure) Mat_n of order n (that is, n rows and n columns). Each element of the matrix is a domain variable that contains a value in the range $\{0 \ldots n-1\}$. We apply algebraic structure constraints on Mat_n to generate a set $S_n = \{mat_1, mat_2, \ldots mat_p\}$ such that $\forall mat_i \in S_n$, where


order (n)
algebraic structure constraints
Constraint Solver
algebraic structures with isomorphic copies (S_n)
Isomorphism Detection
isomorphism classes (R_n)
Fig. 4 Algebraic structure generation

mat_i represents a valid algebraic structure of order n. This set obviously contains many matrices which are isomorphic copies of each other. Next, we apply isomorphic detection that results in another set R_n such that $R_n \subseteq S_n$ and $\forall x \in R_n$, x represents an isomorphism class.

3.2 Mining symmetry breaking rules

In order to identify rules to restrict the number of symmetries, we first need to generate all the algebraic structures of orders p to n (i.e., $S_p, \ldots, S_n$). The value of p is chosen such that S_p has considerably large number of structures. We also determine the corresponding set of isomorphism classes (i.e., $R_p, \ldots, R_n$). Our first attempt for mining rules was to generate *association rules* from the known algebraic structure

Table 3 Some of the association rules extracted from known isomorphism classes of IP loops

Association rule ($\forall mat_i \in R_n, \ldots, R_p$)	Support (%)	Confidence (%)	Lift
$mat_i[3][1] = 4 \implies mat_i[4][4] = 1$	69.24	100	1.44
$mat_i[4][4] = 1 \implies mat_i[3][1] = 4$	69.24	100	1.44
$mat_i[3][1] = 4 \implies mat_i[1][3] = 4$	69.24	100	1.44
$mat_i[1][3] = 4 \implies mat_i[3][1] = 4$	69.24	100	1.44
$mat_i[3][1] = 4 \implies mat_i[4][2] = 3$	69.24	100	1.44
$mat_i[4][2] = 3 \implies mat_i[3][1] = 4$	69.24	100	1.44
$mat_i[3][1] = 4 \wedge mat_i[4][4] = 1 \implies mat_i[1][3] = 4$	69.24	100	1.44

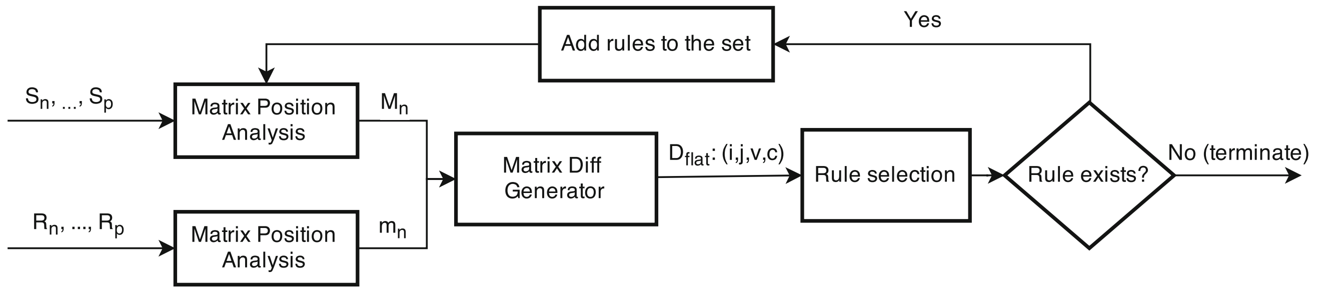


Fig. 5 Rule mining framework

in $R_p, \ldots, R_n$. Each matrix was considered an *itemset*. Each position in the matrix along with its value was considered a unique *item*, $I_{i,j,v}$. So, an item $I_{3,4,4}$ means that the matrix had value 4 at index (3,4). Thus, an $n \times n$ matrix resulted in n^2 items in each itemset.

We applied this approach to the known isomorphism classes of inverse property loop (IP Loop) algebraic structures of order 11 and 13 which consisted of 10391 matrices. The *apriori* algorithm [27] was then applied on these itemsets using R programming language [29]. Please note that we did not feel the need to use any advanced association rule mining algorithm like Eclat [28] as the standard *apriori* algorithm took only few seconds (about 10 seconds) to get all the association rules from 10391 matrices.

We considered rules with support larger than 60 % and the confidence equal to 100 %. This provided us with 186 different rules which had 100 % confidence. Some of these association rules are shown in Table 3. For example, the first rule specified that whenever any matrix in $R_p, \ldots, R_n$ had value 4 at index (3, 1), the matrix also had value 1 at index (4, 4). This was always true (confidence = 100 %) and was observed in about 70 % of the matrices. The rule lift indicated a positive correlation between antecedent and consequent of the rule.

All of the rules were also symmetric with exactly same support and confidence (e.g., $mat_i[3][1] = 4 \Leftrightarrow mat_i[4][4] = 1$). The presence of such rules in these structures was significant. This could potentially be used to add symmetry breaking constraints in the constraint solver. These constraints would discard any matrix as being redundant which did not satisfy these rules. For example, a constraint based on the first rule would discard all matrices which had value 4 at index (3, 1), but did not have value 1 at index (4, 4).

In order to evaluate these association rules, we applied them as additional constraints in our Google's or-tools-based constraint solver for identifying inverse property loop (IP Loops) of order 13. Unfortunately, these rules did not reduce the number of solutions or the time. As it turned out, these rules just discovered certain properties which were holding true because of the constraints (shown in Table 1) placed to generate algebraic structures.

This forced us to rethink our strategy. Our next strategy targeted a rule mining approach that could reduce the number of matrices generated for higher order (S_{n+1}). We noticed that a large number of matrices were eventually discarded later on by the isomorphism detection method. We tried to investigate whether there was any recurring pattern in the matrices which were getting discarded. For this purpose, a new rule mining strategy was devised. The proposed rule mining framework is shown in Fig. 5, and the corresponding algorithm is outlined in Algorithm 1. It considers $S_p, \ldots, S_n$ to create a matrix X such that an element at row i and column j (represented as x_{ij}) of X can be defined as follows. $x_{ij} \subseteq V \times C$ where $V = \{0 \ldots n-1\}$ is a set of values and $C = \{0 \ldots |S_n|\}$ is a set of counts. For example, $x_{00} = \{(3, 5), (4, 1), (1, 6)\}$ means that at (row, column) = (0, 0), 3 appeared in 5 different matrices, 4 appeared in 1 matrix, and 1 appeared in 6 matrices of all the matrices in $S_p, \ldots, S_n$. We apply the same procedure to the set of matrices in $R_p, \ldots, R_n$ to compute a matrix Y such that each element of Y is defined similarly as $y_{ij} \subseteq V \times C$. Finally, we define a difference matrix $D \subseteq V \times C$ and its elements are computed as follows:

$$\forall i, j \leq n, \forall (v, c) \in x_{ij} \wedge (v, c') \notin y_{ij} \implies d_{ij} = d_{ij} \cup (v, c)$$

Algorithm 1 Rule Mining Algorithm

```
1: procedure MineRules($\{S_p, \dots, S_n\}, \{R_p, \dots, R_n\}$)
2:
3:   $R_f \leftarrow empty$                                   ▷ $R_f$ is a set of final rules
4:   while true do
5:     $R_d \leftarrow empty$                                 ▷ $R_d$ is a set of newly discovered rules
6:     $NewRuleDiscovered \leftarrow false$
7:     for $i \leftarrow 0, n-1$ do                           ▷ Initialize matrices
8:       for $j \leftarrow 0, n-1$ do                         ▷ $X[i][j][k] = c$ represents that at row $i$, column $j$, $k$ appears $c$ times
9:         for $k \leftarrow 0, n-1$ do
10:          $X[i][j][k] \leftarrow 0$                        ▷ $X[i][j][k] = c \implies (k, c) \in x_{ij}$
11:          $Y[i][j][k] \leftarrow 0$                        ▷ $Y[i][j][k] = c \implies (k, c) \in y_{ij}$
12:          $D[i][j][k] \leftarrow 0$                        ▷ $D[i][j][k] = c \implies (k, c) \in d_{ij}$
13:    for $q \leftarrow p, n$ do                             ▷ Go over matrices of order $p$ to $n$
14:      for all matrices $mat \in S_q$ do                    ▷ Compute $x_{ij} \in X$
15:        for $i \leftarrow 0, q-1$ do
16:          for $j \leftarrow 0, q-1$ do
17:            for all $rule < row, col, value, count > \in R_f$ do
18:              if $i = row \wedge j = col \wedge mat[i][j] = value$ then
19:                continue;                                  ▷ Ignore the matrices which are already covered by an existing rule
20:              $X[i][j][mat[i][j]] \leftarrow X[i][j][mat[i][j]] + 1$
21:      for all matrices $mat \in R_q$ do                    ▷ Compute $y_{ij} \in Y$
22:        for $i \leftarrow 0, q-1$ do
23:          for $j \leftarrow 0, q-1$ do
24:            $Y[i][j][mat[i][j]] \leftarrow Y[i][j][mat[i][j]] + 1$
25:    for $i \leftarrow 0, n-1$ do                           ▷ Compute $d_{ij} \in D$ containing the difference in $X$ and $Y$
26:      for $j \leftarrow 0, n-1$ do
27:        for $k \leftarrow 0, n-1$ do
28:          if $X[i][j][k] > 0 \wedge Y[i][j][k] = 0$ then
29:            $D[i][j][k] \leftarrow X[i][j][k]$
30:            Create a rule $r < row = i, col = j, value = k, count = D[i][j][k] >$
31:            $R_d \leftarrow R_d \cup r$
32:            $NewRuleDiscovered \leftarrow true$
33:    if $NewRuleDiscovered = true$ then
34:      Find the rule $HighestSignificantRule < row, col, value, count > \in R_d$ with highest significance using Eq. 1
35:      $R_f \leftarrow R_f \cup HighestSignificantRule$
36:    else
37:      return $R_f$
```

Every element $d_{ij} \in D$ contains a set of $(value, count)$ pair. Each such pair indicates that the specified $value$ occurred $count$ times at index (i, j) in the matrices of $S_p, \dots, S_n$ but these values were never observed in any matrix in $R_p, \dots, R_n$ at index (i, j). This means that these matrices are redundant and can be safely discarded. Each such pair thus forms the basis for identifying a rule for symmetry breaking.

For the sake of simplicity in performing significance analysis, we flatten the $d_{ij} \in D$. We define D_{flat} as a 4-tuples (i, j, v, c) set where $v \in V$ and $c \in C$. It can be computed as follows:

$$\forall i, j \leq n, \forall (v, c) \in d_{ij}, D_{flat} = D_{flat} \cup (i, j, v, c)$$

Each tuple (i, j, v, c) identifies a potential rule specifying that any matrix with value v at position i, j can be discarded as no isomorphism class existed with these values at the specified position. However, not every rule has the same significance. A rule which results in highest number of discarded matrices should be preferred over a rule which discards few matrices. Similarly, a rule observed in matrices of all orders (i.e., $n, \dots, p$) should be preferred over the rule that is based on the matrices of any particular order.

We used the above insights to devise our rule significance method. For each element (rule) in D_{flat}, the corresponding significance is computed using the following equation:

$$Sig = (1 - i/n) + (1 - j/n) + (1 - v/n) + c/|S_n| \qquad (1)$$

This equation normalizes each element of the tuple to [0, 1]. In general, the rules with higher counts, lower indices, and lower values are given higher significance. For example, the significance of three elements in D_{flat} which are computed from 200 algebraic structures of order 7, 9, and 11 would be as follows:

– (1, 2, 1, 150) has significance 3.35
– (1, 1, 2, 100) has significance 3.1
– (10, 10, 1, 200) has significance 1.9

The first rule has the highest significance as it has lower values for the indices and higher values for count. The second rule has lower significance than the first rule because of lower value of count. The third rule has the least significance (despite having the highest count) since it has highest values of indices.

4 Results

To show the effectiveness of our proposed methodology, we applied it to the problem of counting and enumerating isomorphism classes of IP loops. We modeled the problem as finite domain constraint satisfaction problem (CSP) and began our study using a leading constraint solver Google's or-tools. We enumerated the algebraic structures and the isomorphism classes of IP loops of order 7, 9, 11, and 13 using known constraints mentioned in Table 1. The reason we chose odd number for the orders was that the set of known constraints are different for odd and even orders of the algebraic structures. Also, as of now, the number of isomorphism classes for IP loop are known up to order 13 only. We evaluated the performance improvements of our proposed approach by enumerating the highest order of known IP loop structures.

Table 4 shows the total number of solutions, the number of isomorphism classes, and the time taken to compute these structures on a general purpose desktop system. It should be noted that the time includes the time taken to determine the solutions as well as the time spent to detect isomorphism classes. Isomorphism was detected based on checking all the permutations of possible mappings. It is quite evident from this table that the number of solutions and the time to identify isomorphism classes increase exponentially with increasing orders of the IP loop structures. For example, the time required for enumerating IP loops of order 13 is increased by 17000 times as compared to time taken for enumerating order 11 IP loops. Similarly, the total number of solutions also increased by 1200 times.

Table 5 shows rules extracted by applying the proposed mining approach to known structures of IP loops of order 7, 9, and 11. This consisted of 6532 matrices. This table also shows the respective support count and the significance of these rules. The first rule specifies the constraint that all matrices which have the value 1 at index (3, 3) should be discarded. This rule was based on the observation that 420 matrices out of 6532 had the value 1 at index (3, 3), but all of them eventually got discarded by the isomorphism detection method (that is, they were redundant copies). The second rule was then discovered from the remaining 6112 matrices. A similar process was repeated to discover the other rules.

Table 4 Time taken and number of solutions for IP loops

Order (n)	Total solutions (with known constraints)	Isomorphism classes	Time (s)
5	1	1	0
7	4	2	0.023
9	64	7	0.024
11	6464	49	5.86
13	7853368	10342	103636

Table 5 Rules extracted from IP loops of order 7, 9, and 11

Rules ($\forall mat_i \in S_n, \ldots, S_p$)	Support count	Significance
$mat_i[3][3] \neq 1$	420	2.58
$mat_i[1][5] \neq 3$	109	2.18
$mat_i[5][5] \neq 1$	259	2.09
$mat_i[3][5] \neq 6$	640	2.08

Table 6 Performance gains for IP loops of order 13

	Without using mined constraints	After using mined constraints	Perf. Gain
Total solutions	7853368	6392816	**18.6 %**
Time (s)	103636	81124	**21 %**
Isomorphism classes	10342	10342	-

These discovered rules were then used as additional constraints for enumerating IP loops of order 13. Table 6 shows the performance improvements. The total number of solutions decreased to 6392816, which is 18.6 % improvement. The time taken to determine isomorphism classes was reduced by 21 % to 81124 s. This shows a considerably large performance gain in terms of time as well as the number of solutions using the mining approach.

4.1 Rules evaluation

It should be noted that inclusion of these additional constraints (based on rules discovered using our proposed rule mining approach) did not cause any loss of information as all the representative isomorphism classes (i.e., 10342 isomorphism classes) were identified successfully.

We conducted further evaluation of the rules to get a break down of the number of solutions discarded by each rule and their corresponding isomorphism classes.

Table 7 shows the total number of solutions discarded by each rule, the number of isomorphism classes which represent the discarded solutions and the corresponding number of unique mappings. For example, the rule $mat_i[5][5] \neq 1$ has discarded 150488 solutions which were represented by 3263 unique isomorphism classes using 4116 different mappings.

Table 7 Rules evaluation: the number of solutions discarded by each rule, the corresponding number of isomorphism classes, and the number of different mappings

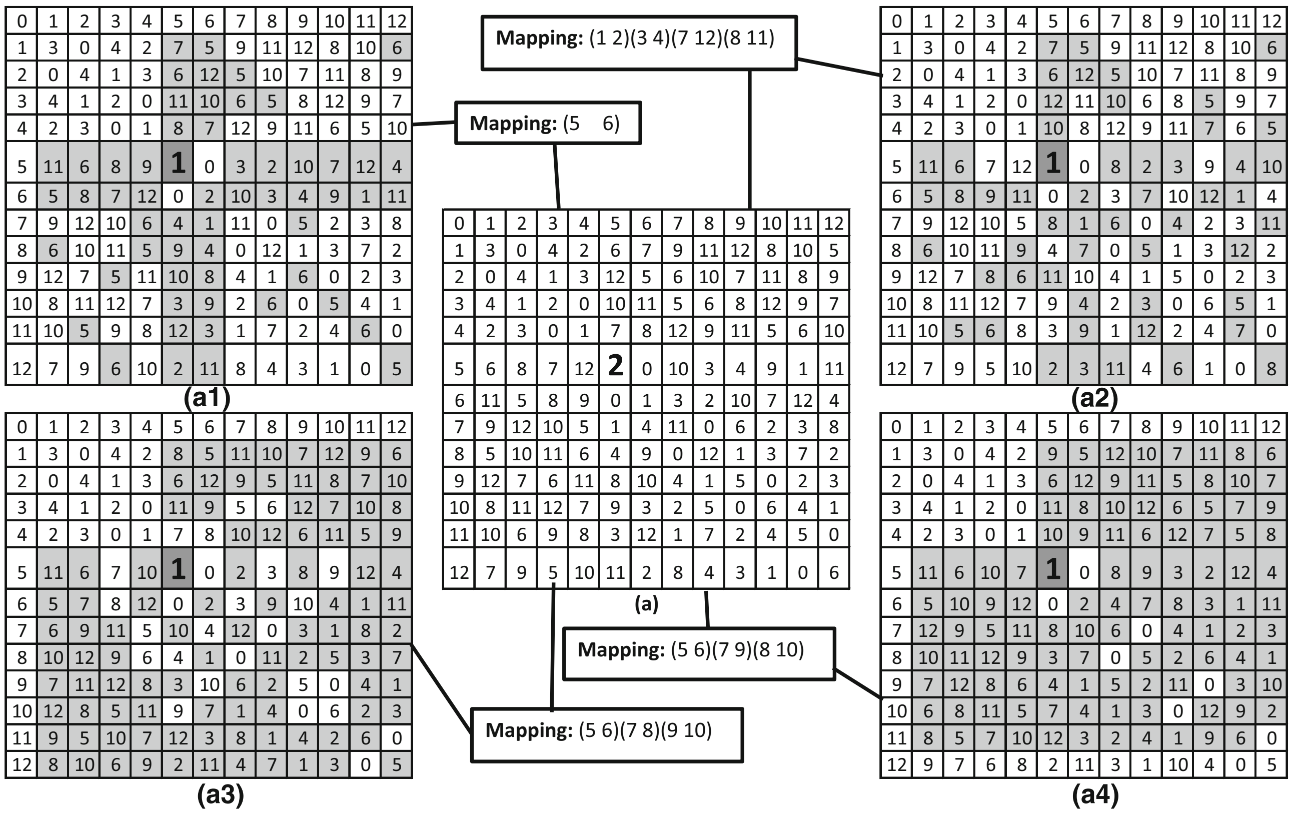
Rules	Number of isomorphic solutions discarded	Number of isomorphism classes	Number of mappings
$mat_i[3][3] \neq 1$	609408	6322	8853
$mat_i[1][5] \neq 3$	304160	6322	6146
$mat_i[5][5] \neq 1$	150448	3263	4116
$mat_i[3][5] \neq 6$	396536	8583	14156
Total solutions (unique)	1460552	8853	21529

Figure 6 shows one of the 3263 isomorphism classes (labeled as **(a)** in the center). It also shows 4 of the 150448 discarded solutions for the rule $mat_i[5][5] \neq 1$ (labeled as **(a1)**, **(a2)**, **(a3)** and **(a4)**). Note that all the discarded solutions shown in Fig. 6 have $mat_i[5][5] = 1$, while the corresponding isomorphism class (in the middle) has $mat_i[5][5] \neq 1$. For clarity, the differences between discarded solutions and their corresponding isomorphism classes are highlighted. The mapping which makes the discarded solution isomorphic to its isomorphism class is also shown in the rectangle box. For example, the matrix labeled **(a1)** is isomorphic to its representative isomorphism class matrix **(a)** due to mapping (5, 6). This means that swapping values 5 with 6, swapping column 5 with column 6 and swapping row 5 with row 6 in matrix **(a1)** will get the matrix **(a)**.

It was observed that the set of isomorphism classes which represented the discarded solutions based on different rules is not mutually exclusive. For example, in Fig. 7, the matrix **(b1)** which was discarded due to $mat_i[5][5] \neq 1$ rule and the matrix **(b2)** which was discarded due to $mat_i[3][5] \neq 6$ rule are represented by the same isomorphism class (i.e., matrix **(b)**). It was also observed that the same set of 6322 isomorphism classes represented all the solutions (matrices) discarded by $mat_i[3][3] \neq 1$ and $mat_i[1][5] \neq 3$ rules. In general, all of the 1460552 discarded solutions were represented by 8853 isomorphism classes using 21529 different mappings.

5 Conclusion

Studying algebraic structures is an important area of research in mathematics with applications in many areas of computer science. However, generating these structures is computa-


Mapping: (1 2)(3 4)(7 12)(8 11)
Mapping: (5 6)
Mapping: (5 6)(7 9)(8 10)
Mapping: (5 6)(7 8)(9 10)
(a1)
(a2)
(a)
(a3)
(a4)

(a1)

0	1	2	3	4	5	6	7	8	9	10	11	12
1	3	0	4	2	7	5	9	11	12	8	10	6
2	0	4	1	3	6	12	5	10	7	11	8	9
3	4	1	2	0	11	10	6	5	8	12	9	7
4	2	3	0	1	8	7	12	9	11	6	5	10
5	11	6	8	9	1	0	3	2	10	7	12	4
6	5	8	7	12	0	2	10	3	4	9	1	11
7	9	12	10	6	4	1	11	0	5	2	3	8
8	6	10	11	5	9	4	0	12	1	3	7	2
9	12	7	5	11	10	8	4	1	6	0	2	3
10	8	11	12	7	3	9	2	6	0	5	4	1
11	10	5	9	8	12	3	1	7	2	4	6	0
12	7	9	6	10	2	11	8	4	3	1	0	5

(a2)

0	1	2	3	4	5	6	7	8	9	10	11	12
1	3	0	4	2	7	5	9	11	12	8	10	6
2	0	4	1	3	6	12	5	10	7	11	8	9
3	4	1	2	0	12	11	10	6	8	5	9	7
4	2	3	0	1	10	8	12	9	11	7	6	5
5	11	6	7	12	1	0	8	2	3	9	4	10
6	5	8	9	11	0	2	3	7	10	12	1	4
7	9	12	10	5	8	1	6	0	4	2	3	11
8	6	10	11	9	4	7	0	5	1	3	12	2
9	12	7	8	6	11	10	4	1	5	0	2	3
10	8	11	12	7	9	4	2	3	0	6	5	1
11	10	5	6	8	3	9	1	12	2	4	7	0
12	7	9	5	10	2	3	11	4	6	1	0	8

(a)

0	1	2	3	4	5	6	7	8	9	10	11	12
1	3	0	4	2	6	7	9	11	12	8	10	5
2	0	4	1	3	12	5	6	10	7	11	8	9
3	4	1	2	0	10	11	5	6	8	12	9	7
4	2	3	0	1	7	8	12	9	11	5	6	10
5	6	8	7	12	2	0	10	3	4	9	1	11
6	11	5	8	9	0	1	3	2	10	7	12	4
7	9	12	10	5	1	4	11	0	6	2	3	8
8	5	10	11	6	4	9	0	12	1	3	7	2
9	12	7	6	11	8	10	4	1	5	0	2	3
10	8	11	12	7	9	3	2	5	0	6	4	1
11	10	6	9	8	3	12	1	7	2	4	5	0
12	7	9	5	10	11	2	8	4	3	1	0	6

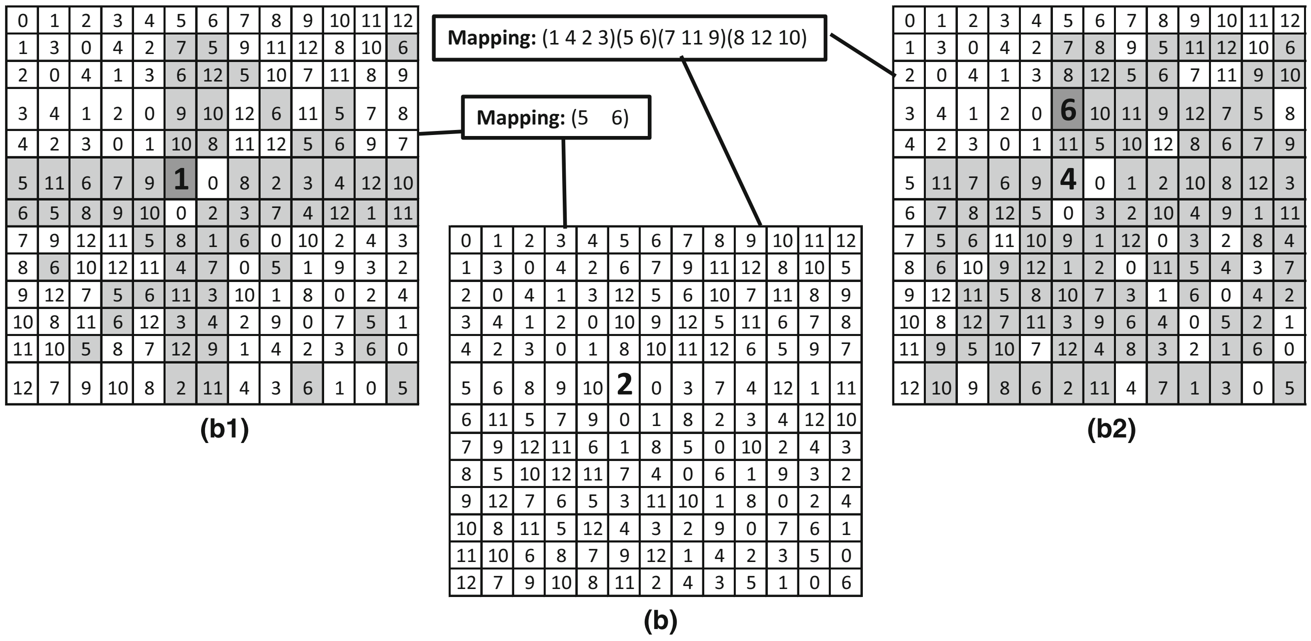
(a3)

0	1	2	3	4	5	6	7	8	9	10	11	12
1	3	0	4	2	8	5	11	10	7	12	9	6
2	0	4	1	3	6	12	9	5	11	8	7	10
3	4	1	2	0	11	9	5	6	12	7	10	8
4	2	3	0	1	7	8	10	12	6	11	5	9
5	11	6	7	10	1	0	2	3	8	9	12	4
6	5	7	8	12	0	2	3	9	10	4	1	11
7	6	9	11	5	10	4	12	0	3	1	8	2
8	10	12	9	6	4	1	0	11	2	5	3	7
9	7	11	12	8	3	10	6	2	5	0	4	1
10	12	8	5	11	9	7	1	4	0	6	2	3
11	9	5	10	7	12	3	8	1	4	2	6	0
12	8	10	6	9	2	11	4	7	1	3	0	5

(a4)

0	1	2	3	4	5	6	7	8	9	10	11	12
1	3	0	4	2	9	5	12	10	7	11	8	6
2	0	4	1	3	6	12	9	11	5	8	10	7
3	4	1	2	0	11	8	10	12	6	5	7	9
4	2	3	0	1	10	9	11	6	12	7	5	8
5	11	6	10	7	1	0	8	9	3	2	12	4
6	5	10	9	12	0	2	4	7	8	3	1	11
7	12	9	5	11	8	10	6	0	4	1	2	3
8	10	11	12	9	3	7	0	5	2	6	4	1
9	7	12	8	6	4	1	5	2	11	0	3	10
10	6	8	11	5	7	4	1	3	0	12	9	2
11	8	5	7	10	12	3	2	4	1	9	6	0
12	9	7	6	8	2	11	3	1	10	4	0	5

Fig. 6 An isomorphism class (in *the middle*) and 4 of the 150448 redundant isomorphic copies discarded by $mat_i[5][5] \neq 1$ rule for IP Loop of order 13


Mapping: (1 4 2 3)(5 6)(7 11 9)(8 12 10)
Mapping: (5 6)

(b1)

0	1	2	3	4	5	6	7	8	9	10	11	12
1	3	0	4	2	7	5	9	11	12	8	10	6
2	0	4	1	3	6	12	5	10	7	11	8	9
3	4	1	2	0	9	10	12	6	11	5	7	8
4	2	3	0	1	10	8	11	12	5	6	9	7
5	11	6	7	9	**1**	0	8	2	3	4	12	10
6	5	8	9	10	0	2	3	7	4	12	1	11
7	9	12	11	5	8	1	6	0	10	2	4	3
8	6	10	12	11	4	7	0	5	1	9	3	2
9	12	7	5	6	11	3	10	1	8	0	2	4
10	8	11	6	12	3	4	2	9	0	7	5	1
11	10	5	8	7	12	9	1	4	2	3	6	0
12	7	9	10	8	2	11	4	3	6	1	0	5

(b)

0	1	2	3	4	5	6	7	8	9	10	11	12
1	3	0	4	2	6	7	9	11	12	8	10	5
2	0	4	1	3	12	5	6	10	7	11	8	9
3	4	1	2	0	10	9	12	5	11	6	7	8
4	2	3	0	1	8	10	11	12	6	5	9	7
5	6	8	9	10	**2**	0	3	7	4	12	1	11
6	11	5	7	9	0	1	8	2	3	4	12	10
7	9	12	11	6	1	8	5	0	10	2	4	3
8	5	10	12	11	7	4	0	6	1	9	3	2
9	12	7	6	5	3	11	10	1	8	0	2	4
10	8	11	5	12	4	3	2	9	0	7	6	1
11	10	6	8	7	9	12	1	4	2	3	5	0
12	7	9	10	8	11	2	4	3	5	1	0	6

(b2)

0	1	2	3	4	5	6	7	8	9	10	11	12
1	3	0	4	2	7	8	9	5	11	12	10	6
2	0	4	1	3	8	12	5	6	7	11	9	10
3	4	1	2	0	**6**	10	11	9	12	7	5	8
4	2	3	0	1	11	5	10	12	8	6	7	9
5	11	7	6	9	**4**	0	1	2	10	8	12	3
6	7	8	12	5	0	3	2	10	4	9	1	11
7	5	6	11	10	9	1	12	0	3	2	8	4
8	6	10	9	12	1	2	0	11	5	4	3	7
9	12	11	5	8	10	7	3	1	6	0	4	2
10	8	12	7	11	3	9	6	4	0	5	2	1
11	9	5	10	7	12	4	8	3	2	1	6	0
12	10	9	8	6	2	11	4	7	1	3	0	5

Fig. 7 An isomorphism class (in *the middle*) and 2 redundant isomorphic copies discarded by $mat_i[5][5] \neq 1$ and $mat_i[3][5] \neq 6$ rules for IP Loop of order 13

tionally expensive because of overwhelmingly large number of symmetries present in these structures. In this paper, we presented a mining-based approach to discover symmetry breaking constraints in algebraic structures. We demonstrated the effectiveness of our approach by applying it to enumerate IP loops. We found new symmetry breaking constraints that resulted in significant reduction in the number of redundant solutions, thereby reducing computational time to generate these structures.

To the best of our knowledge, this is the first time a mining-based approach has been applied to discover symmetry breaking constraints. This work can be enhanced in multiple directions. A similar approach can be applied to other algebraic structures like C loops and flexible loops. Applying other mining approaches such as clustering and classification need further investigation.

Acknowledgments This work was supported by Prince Mohammad Bin Fahd University (PMU) internal research grant. The views and conclusions herein are those of the authors and do not represent the official policies of the university.

References

1. Khan, M.A., Mohammad, N., Muhammad, S., Ali, A.: A mining based approach for efficient enumeration of algebraic structures. In: IEEE International Conference on Data Science and Advanced Analytics (DSAA) (2015)
2. McKay, B.D., Meynert, A., Myrvold, W.: Small latin squares, quasigroups, and loops. J. Comb. Des. **15**, 98–119 (2007)
3. Battey, M., Parakh, A.: An efficient quasigroup block cipher. Wirel. Pers. Commun. **73**(1), 63–76 (2013)
4. Krapez, A.: An application of quasigroups in cryptology. Math. Maced **8**, 47–52 (2010)
5. Gent, I.P., Barbara, S.: Symmetry Breaking During Search in Constraint Programming. University of Leeds, School of Computer Studies, Leeds (1999)
6. Gent, I.P., Harvey, W., Kelsey, T.: Groups and Constraints: Symmetry Breaking During Search. Principles and Practice of Constraint Programming-CP 2002. Springer, Berlin (2002)
7. Ali, A., Slayney, J.: Counting loops with the inverse property. Quasigroups Relat. Syst. **16**, 13 (2008)
8. McKay, B.D.: Practical graph isomorphism. Congr. Numer. **30**, 3587 (1981)
9. Albert, A.A.: Quasigroups. II. Trans. Am. Math. Soc. **55**, 401–409 (1944)
10. Bammel, S.E., Rothstein, J.: The number of 9 × 9 latin squares. Discret. Math. **11**, 83–95 (1975)
11. Brant, L.J., Mullen, G.L.: A note on isomorphism classes of reduced latin squares of order 7. Util. Math. **27**, 261–263 (1985)
12. Brown, J.W.: Enumeration of latin squares with application to order 8. J. Comb. Theory **5**, 177–184 (1972)
13. Cayley, A.: On latin squares. Oxf. Camb. Dublin Messenger Math. **19**, 85–239 (1890)
14. Euler, L.: Recherches sur une nouvelle espéce de quarrés magiques combinatorial aspects of relations. Verhandelingen/uitgegeven door het zeeuwsch Genootschap der Wetenschappen te Vlissingen, **9**, 85–239, (1782)
15. Fisher, R.A., Yates, F.: The 6 × 6 latin squares. Proc. Camb. Philos. Soc. **30**, 492–507 (1934)
16. Frolov, M.: Sur les permutations carrées. J. Math. Spéc **IV**, 8–11 (1890)
17. Jacob, S.M.: The enumeration of the latin rectangle of depth three by means of a formula of reduction, with other theorems relating to non-clashing substitutions and latin squares. Proc. Lond. Math. Soc. **31**, 329–354 (1930)

18. MacMahon, P.A.: Combinatory Analysis, vol. 1. Cambridge University Press, Cambridge (1915)
19. McKay, B.D., Rogoyski, E.: Latin squares of order 10. Electron. J. Combin. **2**, N3 (1995)
20. McKay, B.D., Wanless, I.M.: On the number of latin squares. Ann. Combin. **9**, 335–344 (2005)
21. Norton, H.W.: The 7×7 squares. Ann. Eugenics **9**, 269–307 (1939)
22. Sade, A.: An omission in norton's list of 7×7 squares. Ann. Math. Stat. **22**, 306–307 (1951)
23. Sade, A.: Morphismes de quasigroupes: Tables. *Revista da Faculdade de Ciências de Lisboa, 2: A – Ciências Matemáticas*, **13** 149–172, (1970/71)
24. Saxena, P.N.: A simplified method of enumerating latin squares by macmahon's differential operators; II. The 7×7 latin squares. J. Indian Soc. Agric. Stat. **3**, 24–79 (1951)
25. Schönhardt, E.: Über lateinische quadrate und unionen. J. Reine Angew. Math. **163**, 183–230 (1930)
26. Wells, M.B.: The number of latin squares of order eight. J. Comb. Theory **3**, 98–99 (1967)
27. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of 20th International Conference on very Large Data Bases, VLDB **1215**, 487–499 (1994)
28. Zaki, M.J.: Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng. **12**(3), 372–390 (2000)
29. R: A Language and Environment for Statistical Computing. http://www.R-project.org