

# Computing autotopism groups of partial Latin rectangles: a pilot study

Rebecca J. Stones

College of Computer Science, Nankai University, Tianjin, China.  
rebecca.stones82@gmail.com

Raúl M. Falcón

School of Building Engineering, University of Seville, Spain.  
rafalgan@us.es

Daniel Kotlar

Department of Computer Science, Tel-Hai College, Upper Galilee, Israel.  
dannkotlar@gmail.com

Trent G. Marbach

College of Computer Science, Nankai University, Tianjin, China.  
trent.marbach@outlook.com

October 23, 2019

## Abstract

Computing the autotopism group of a partial Latin rectangle can be performed in a variety of ways. This pilot study has two aims: (a) to compare these methods experimentally, and (b) to identify the design goals one should have in mind for developing practical software. To this end, we compare six families of algorithms (two backtracking methods and four graph automorphism methods), with and without the use of entry invariants, on two test suites. We consider two entry invariants: one determined by the frequencies of row, column, and symbol representatives, and one determined by  $2 \times 2$  submatrices. We find: (a) with very few entries, many symmetries often exist, and these should be identified mathematically rather than computationally, (b) with an intermediate number of entries, a quick-to-compute entry invariant was effective at reducing the need for computation, (c) with an almost-full partial Latin rectangle, more sophisticated entry invariants are needed, and (d) the performance for (full) Latin squares is significantly poorer than other partial Latin rectangles of comparable size, obstructed by the existence of Latin squares with large (possibly transitive) autotopism groups.

**Keywords:** Autotopism, Latin square, partial Latin rectangle.

## 1 Introduction

Let  $[n] = \{1, 2, \dots, n\}$ . An  $r \times s$  *partial Latin rectangle* is an  $r \times s$  array containing symbols from  $[n] \cup \{\cdot\}$ , where each symbol in  $[n]$  appears at most once within each row and each column. If  $r = s = n$ , then this is a *partial Latin square of order  $n$* .

Let  $\text{PLR}(r, s, n)$  denote the set of  $r \times s$  partial Latin rectangles. Every  $L \in \text{PLR}(r, s, n)$  is uniquely determined by its *entry set*

$$\text{Ent}(L) := \{(i, j, L[i, j]) : i \in [r], j \in [s], L[i, j] \in [n]\}.$$

Any triple  $(i, j, L[i, j]) \in \text{Ent}(L)$  is an *entry* of  $L$ , while any pair  $(i, j) \in [r] \times [s]$  is a *cell*. The latter is considered *empty* if it contains the symbol  $\cdot$ , in which case,  $L[i, j]$  is *undefined*. If  $L \in \text{PLR}(r, s, n)$  does not have empty cells, then it is a *Latin rectangle* (or a *Latin square*, if  $r = s = n$ ). This constitutes a generalization of the standard definition of a Latin rectangle [1], which requires that  $s = n$ .

Let  $S_m$  denote the symmetric group on  $[m]$ . A triple of permutations  $\theta := (\alpha, \beta, \gamma) \in S_r \times S_s \times S_n$  acts on  $\text{PLR}(r, s, n)$ , with  $\theta$  acting on  $L \in \text{PLR}(r, s, n)$  by permuting its rows, columns, and symbols by  $\alpha$ ,  $\beta$ , and  $\gamma$ , respectively. The triple  $\theta$  is called an *isotopism* and both  $L$  and the resulting partial Latin rectangle  $L^\theta \in \text{PLR}(r, s, n)$  are *isotopic*. Moreover,

$$\text{Ent}(L^\theta) = \{(\alpha(i), \beta(j), \gamma(L[i, j])) : (i, j, L[i, j]) \in \text{Ent}(L)\},$$

where, from here on, we use  $(i, j, L[i, j])^\theta$  to denote  $(\alpha(i), \beta(j), \gamma(L[i, j]))$ . If  $L = L^\theta$ , then  $\theta$  is said to be an *autotopism* of  $L$ . The set  $\text{Atop}(L)$  of autotopisms of  $L$  forms a group, called the *autotopism group* of  $L$ .

Autotopisms of (partial) Latin squares have arisen in a range of topics and are also studied in their own right [2–19]. In particular, they are beginning to find applications in cryptography [20–22], erasure codes [23, 24] (relating to Blackburn partial Latin squares [25]), and graph colouring games [26, 27]. However, autotopisms of (partial) Latin rectangles, have only briefly been studied in recent papers [28–30].

The standard way of computing autotopism groups of Latin squares was given by McKay, Meynert, and Myrvold [31], who describe a graph whose automorphism group is isomorphic to the autotopism group of the corresponding Latin square. A generalization of this method has already been used for studying partial Latin squares with trivial symmetry groups [32]. For autotopisms of Latin squares, Kotlar [33] gave a method which enumerates cycles belonging to entries, which can be thought of as a special case of, what we call in this paper, row invariants.

Almost all Latin squares have trivial autotopism groups [34, 35], so, in almost all cases, computing such a group boils down to eliminating the possible existence of a non-trivial autotopism. This procedure cannot be, however, straightforwardly generalized to (partial) Latin rectangles, because: (a) two empty rows (or two empty columns) in a partial Latin rectangle can be swapped, giving rise to a large family of partial Latin rectangles with a non-trivial autotopism; (b)  $2 \times n$  Latin rectangles always have non-trivial symmetries; and (c) partial Latin squares with very few entries must have non-trivial symmetries [32].

In this paper, six families of algorithms for computing this group are implemented and experimentally compared. Some of them have been used in prior research on partial Latin rectangles, while others are introduced here.

## 2 Autotopisms of partial Latin rectangles

### 2.1 Introduction

We start with a characterization of the existence of an autotopism of a partial Latin rectangle with specified row and column permutations, but with an unspecified symbol permutation. We defer proofs to a subsequent paper.

**Lemma 2.1.** *Let  $L \in \text{PLR}(r, s, n)$ . If  $\alpha \in S_r$  and  $\beta \in S_s$ , then there exists  $\gamma \in S_n$  for which  $\theta = (\alpha, \beta, \gamma) \in \text{Atop}(L)$  if and only if*

1.  $L[\alpha(i), \beta(j)]$  is defined for all  $(i, j, L[i, j]) \in \text{Ent}(L)$ , and
2.  $L[\alpha(i), \beta(j)] = L[\alpha(i'), \beta(j')]$ , whenever  $L[i, j] = L[i', j'] \in [n]$ .

*The permutations  $\gamma$  for which  $\theta \in \text{Atop}(L)$  are those satisfying  $\gamma(L[i, j]) = L[\alpha(i), \beta(j)]$ , for all  $(i, j, L[i, j]) \in \text{Ent}(L)$ .  $\square$*

Lemma 2.1 suggests an algorithm for simultaneously determining whether or not a row permutation  $\alpha$  and a column permutation  $\beta$  participate in an autotopism of a partial Latin rectangle  $L \in \text{PLR}(r, s, n)$ , and identifying which symbol permutations  $\gamma$  are possible. We proceed entry by entry, setting  $\gamma(L[i, j]) = L[\alpha(i), \beta(j)]$  for every entry  $(i, j, L[i, j]) \in \text{Ent}(L)$ . Three clashes can arise: (a)  $L[\alpha(i), \beta(j)]$  is undefined; (b)  $\gamma(L[i, j])$  was previously set to something else; or (c)  $\gamma^{-1}(L[\alpha(i), \beta(j)])$  was previously set to something else. If no clashes arise, then the triple  $(\alpha, \beta, \gamma)$  is an autotopism for all completions of  $\gamma$  to a permutation in  $S_n$ .

Lemma 2.1 also enables one to compute the autotopism group while ignoring empty rows, empty columns, and unused symbols. Specifically, suppose  $L$  has  $r' \leq r$  non-empty rows,  $s' \leq s$  non-empty columns and  $n' \leq n$  used symbols. Let  $\bar{L} \in \text{PLR}(r', s', n')$  denote the partial Latin rectangle that results after eliminating the empty rows and columns of  $L$  and relabeling the used symbols of  $L$  to  $[n']$ . Thus, the following result holds.

**Lemma 2.2.**  $|\text{Atop}(L)| = (r - r')!(s - s')!(n - n')!|\text{Atop}(\bar{L})|$ .  $\square$

## 2.2 Entry invariants

An *entry invariant*  $\mathcal{P}$  is any property of the entries of partial Latin rectangles which is preserved under autotopisms. That is, if  $L$  is a partial Latin rectangle, then  $\mathcal{P}(e) = \mathcal{P}(e^\theta)$ , for all  $e \in \text{Ent}(L)$  and  $\theta \in \text{Atop}(L)$ . We also define a *row invariant*, *column invariant*, and *symbol invariant* of  $L$  to be respectively any property of its rows, columns, or symbols which is preserved under autotopisms. Given an entry invariant  $\mathcal{P}$ , a useful row invariant is the multiset of entry invariants of the entries in that row, denoted  $\mathcal{P}_R$ , and likewise for columns, denoted  $\mathcal{P}_C$ . The symbol invariant  $\mathcal{P}_S$  is the multiset of entry invariants of the entries containing the particular symbol in the third component.

**Lemma 2.3.** *Let  $L \in \text{PLR}(r, s, n)$  have entry invariant  $\mathcal{P}$ . Then,  $L$  has a trivial autotopism group if one of the following conditions holds.*

- a)  $\mathcal{P}(e) \neq \mathcal{P}(e')$ , for every pair of distinct entries  $e, e' \in \text{Ent}(L)$ .
- b) *There exist distinct  $X, Y \in \{R, C, S\}$  such that  $\mathcal{P}_X(i) \neq \mathcal{P}_X(i')$  whenever  $i \neq i'$ , and  $\mathcal{P}_Y(j) \neq \mathcal{P}_Y(j')$  whenever  $j \neq j'$ , where  $i, i', j$  and  $j'$  are taken from the appropriate sets of rows, columns, or symbols.  $\square$*

Every entry invariant  $\mathcal{P}$  of  $L$  determines a partition of its set  $\text{Ent}(L)$  whereby we replace all the entries on  $L$  by the index of their entry invariant (we subsequently give examples in (1) and (2)). Let  $\mathbf{c}_j^{\mathcal{P}}(L)$  denote the  $j$ -th column of the resulting new array, for all  $j \in [s]$ . Let  $\bar{\mathbf{c}}_j(L) \subseteq [r]$  denote the set of row indices  $i$  for which the cell  $(i, j)$  is non-empty.

**Lemma 2.4.** *Let  $L \in \text{PLR}(r, s, n)$  and  $\theta = (\alpha, \beta, \gamma) \in \text{Atop}(L)$ . If  $\mathcal{P}$  is invariant under symbol permutations, then*

a)  $\mathbf{c}_j^{\mathcal{P}}(L^{(\alpha, \text{id}, \text{id})})[i] = \mathbf{c}_{\beta(j)}^{\mathcal{P}}(L)[i]$ , for all  $i \in [r]$  and  $j \in [s]$ .

b)  $\bar{\mathbf{c}}_j(L) = \bar{\mathbf{c}}_{\beta(j)}(L)$ , for all  $j \in [s]$ .

In this paper, all entry invariants are invariant under symbol permutations.

### 2.2.1 Strong entry invariants

The *strong entry invariant* [30] of an entry  $(i, j, L[i, j])$  is the triple  $(a, b, c)$  formed by the number  $a$  of entries in row  $i$ , the number  $b$  of entries in column  $j$ , and the number  $c$  of times the symbol  $L[i, j]$  appears in  $L$ . For example:

1	·	2	·	·	·	3	·	·
2	·	·	4	1	5	6	·	7
·	1	5	3	·	4	·	·	·
·	2	·	5	·	3	·	4	·
4	3	·	·	5	·	1	·	2
·	·	·	·	2	·	·	1	3

strong entry invariants  
relabelled 1, 2, ...

 $\longrightarrow$

1	·	2	·	·	·	1	·	·
3	·	·	4	3	4	5	·	5
·	6	7	6	·	8	·	·	·
·	6	·	8	·	6	·	7	·
9	10	·	·	9	·	10	·	10
·	·	·	·	1	·	·	2	1

(1)

The multisets of strong entry invariants in each row and column imply that in any autotopism  $(\alpha, \beta, \gamma)$  of this partial Latin rectangle,  $\alpha$  and  $\beta$  fix the partitions  $\{\{1, 6\}, \{2\}, \{3, 4\}, \{5\}\}$  and  $\{\{1, 5\}, \{2\}, \{3, 8\}, \{4, 6\}, \{7, 9\}\}$  set-wise, respectively. It turns out there is a single non-trivial autotopism with row permutation (16)(34), column permutation (15)(38)(46)(79) and symbol permutation (12)(45)(67). Thus, no finer partitions of the sets [6] and [9] can be achieved using row and column invariants, respectively.

### 2.2.2 Square invariants

Given a partial Latin rectangle  $L \in \text{PLR}(r, s, n)$ , any entry  $(i, j, k)$  belongs to exactly  $(r-1)(s-1)$  distinct  $2 \times 2$  submatrices of  $L$ , a general one having the following form:

$$\begin{array}{c|cc} & j & j' \\ \hline i & k & x \\ i' & y & z \end{array}$$

which may have some of the following five properties: (a)  $x$  is undefined, (b)  $y$  is undefined, (c)  $z$  is undefined, (d)  $k = z$ , and (e)  $x = y$ . This gives a maximum of  $2^5 = 32$  possibilities, whose enumeration gives a length-32 vector that sums to  $(r-1)(s-1)$ . This vector constitutes an entry invariant, which we call the *square invariant*. Unlike strong entry invariants, the square invariants for Latin squares are not necessarily equal. For example:

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

$2 \times 2$  entry invariants  
relabelled 1, 2, ...

 $\longrightarrow$

1	2	1	1	2
2	1	1	1	2
1	1	1	2	2
1	1	2	1	2
2	2	2	2	3

(2)

### 3 Computing the autotopism group

Here, we introduce the methods we consider for computing the autotopism group.

#### 3.1 Backtracking methods

##### 3.1.1 Alpha-beta backtracking

We start with the outputs of  $\alpha$ ,  $\beta$ , and  $\gamma$  as undefined, and use backtracking to determine the possible row permutations  $\alpha$ . At each level of the search tree we designate

$$i \xrightarrow{\alpha} a$$

for some  $i, a \in [r]$  provided  $\alpha^{-1}(a)$  is not already defined as something other than  $i$ . Once  $\alpha$  has been decided, we again use backtracking to establish the possible column permutations  $\beta$ . At each level, we designate

$$j \xrightarrow{\beta} b$$

for some  $j, b \in [s]$  provided  $\beta^{-1}(b)$  is not already defined as something other than  $j$ . Finally, for each pair  $(\alpha, \beta)$ , Lemma 2.1 describes how to determine whether an autotopism  $(\alpha, \beta, \gamma)$  exists, and how to find the possible symbol permutations  $\gamma$ .

In practice, both alpha- and beta-search trees can benefit from invariants, e.g., if we decide  $\alpha(i) = a$ , then rows  $i$  and  $a$  must have the same value for the invariants described in 2.2.1 and 2.2.2. In this regard, we can use Lemma 2.4 to prune the backtracking method described above. We refer to this improvement of the backtracking method as the *CV method* (Column Vector).

##### 3.1.2 Entrywise backtracking

We start with the outputs of  $\alpha$ ,  $\beta$ , and  $\gamma$  as undefined, and at each level of the search tree, we designate

$$(i, j, L[i, j]) \xrightarrow{\theta} (a, b, L[a, b])$$

or equivalently, we simultaneously designate

$$i \xrightarrow{\alpha} a, \quad j \xrightarrow{\beta} b, \quad \text{and} \quad L[i, j] \xrightarrow{\gamma} L[a, b],$$

provided none of the next six clashes arises: (a)  $\alpha(i)$  is already defined as something other than  $a$ ; (b)  $\beta(j)$  is already defined as something other than  $b$ ; (c)  $\gamma(L[i, j])$  is already defined as something other than  $L[a, b]$ ; (d)  $\alpha^{-1}(a)$  is already defined as something other than  $i$ ; (e)  $\beta^{-1}(b)$  is already defined as something other than  $j$ ; or (f)  $\gamma^{-1}(L[a, b])$  is already defined as something other than  $L[i, j]$ . In addition,  $i$  and  $\alpha(i)$  and  $j$  and  $\beta(j)$  respectively require the same row and column invariants implied by the invariants (as in Sections 2.2.1 and 2.2.2).

#### 3.2 Graph theoretic methods

A *vertex invariant* is a property of the vertices of graphs which is preserved under automorphisms. In this way, a vertex invariant partitions the vertices such that no automorphism of the graph permutes vertices from one part to another. In addition to being able to list the automorphisms of the graph, NAUTY [36] (available from <http://pallini.di.uniroma1.it/>) also outputs a function called `orbits`, which gives the finest possible vertex invariant partition.

In the graphs we study, the vertex set is or contains  $\text{Ent}(L)$  for some partial Latin rectangle  $L \in \text{PLR}(r, s, n)$ . Entry invariants can be used to color these vertices. In general, this coloring does not correspond to a vertex invariant, as it might not be preserved under all automorphisms of the graph, but the automorphisms which have been eliminated by coloring the vertices do not correspond to autotopisms of  $L$ .

### 3.2.1 Adapted McKay, Meynert, and Myrvold method

We define the vertex-colored graph  $G_L$  by

$$\begin{aligned} V(G_L) := & \text{Ent}(L) \cup \{R_i : i \in [r] \text{ and row } i \text{ of } L \text{ is non-empty}\} \\ & \cup \{S_j : j \in [s] \text{ and column } j \text{ of } L \text{ is non-empty}\} \\ & \cup \{N_k : k \in [n] \text{ and symbol } k \text{ occurs in } L\} \end{aligned}$$

where each of the four subsets,  $\text{Ent}(L)$ ,  $\{R_i\}$ ,  $\{S_j\}$ , and  $\{N_k\}$ , are assigned a distinct vertex color, and edge set

$$E(G_L) := \{eR_i, eS_j, eN_{L[i,j]} : e := (i, j, L[i, j]) \in \text{Ent}(L)\}.$$

By restricting  $\text{Aut}(G_L)$  to  $\text{Ent}(L)$ , we determine  $\text{Atop}(L)$  up to permutations of empty rows and columns, and unused symbols.

### 3.2.2 Bipartite graph method

Pairs of permutations  $(\alpha, \beta) \in S_r \times S_s$  act on the set of  $r \times s$   $(0, 1)$ -matrices by permuting the rows by  $\alpha$  and the columns by  $\beta$ ; we define an *autotopism* as a stabilizer under this group action. Corresponding to  $L \in \text{PLR}(r, s, n)$ , we define a  $(0, 1)$ -matrix  $M = M[i, j]_{r \times s}$  with  $M[i, j] = 1$  if and only if  $L[i, j]$  is defined. The group  $\text{Atop}(L)$  is thus isomorphic to a subgroup of the autotopism group of  $M$ . Moreover, the matrix  $M$  can be interpreted as the biadjacency matrix of a bipartite graph  $B_M$ . This gives rise to two ways of computing  $\text{Aut}(L)$ :

1. Use NAUTY to compute the automorphism group of  $B_M$ . This determines the autotopism group of  $M$ , from which we check each  $(\alpha, \beta)$  in the autotopism group of  $M$  to see if it defines an autotopism of  $L$  through Lemma 2.1.
2. Use NAUTY to compute the **orbits** of  $B_M$ . This determines the finest possible partitions of  $[r]$  and  $[s]$  for use as row and column invariants, respectively. We then apply one of the backtracking methods of Section 3.1 with the added condition that **orbits** are preserved.

### 3.2.3 Partial Latin rectangle graph method

We define the edge-colored graph  $\Gamma_L$  with vertex set  $V(\Gamma_L) := \text{Ent}(L)$  and:

- green edges between two distinct entries if they share a row,
- orange edges between two distinct entries if they share a column, and
- purple edges between two distinct entries if they share a symbol.

It is known [30] that any automorphism of  $\Gamma_L$  is equivalent to an autotopism of  $L$ . The vertex set of  $\Gamma_L$  is  $\text{Ent}(L)$ , which implies the automorphisms of  $\Gamma_L$  act on the entry set of  $L$ , so they are determined directly from NAUTY's output.

However, NAUTY does not allow its input graphs to have edge colors; we consider two ways to overcome this. The first one is to simply ignore the edge colors, then check each automorphism of  $\Gamma_L$  returned by NAUTY to see if it corresponds to an autotopism of  $L$ . The second method is to define a new non-edge-colored graph  $\overline{\Gamma}_L$ , whose automorphisms correspond to the automorphisms of the edge-colored  $\Gamma_L$ . It has vertex set

$$V(\overline{\Gamma}_L) := \text{Ent}(L) \cup \{R_e : e \in \text{Ent}(L)\} \cup \{S_e : e \in \text{Ent}(L)\} \cup \{N_e : e \in \text{Ent}(L)\},$$

where the unions above indicate distinct vertex colors, and edge set

$$\begin{aligned} E(\overline{\Gamma}_L) := & \{eR_e, eS_e, eN_e : e \in \text{Ent}(L)\} \\ & \cup \{R_{e_1}R_{e_2} : \text{distinct entries } e_1 \text{ and } e_2 \text{ belong to the same row}\} \\ & \cup \{S_{e_1}S_{e_2} : \text{distinct entries } e_1 \text{ and } e_2 \text{ belong to the same column}\} \\ & \cup \{N_{e_1}N_{e_2} : \text{distinct entries } e_1 \text{ and } e_2 \text{ share the same symbol}\}. \end{aligned}$$

The graph  $\overline{\Gamma}_L$  has more vertices than  $\Gamma_L$ , so NAUTY takes longer, but the automorphism group of  $\overline{\Gamma}_L$ , when restricted to  $\text{Ent}(L)$  gives the autotopism group of  $L$  directly. Entry invariants can be used to color the vertices in  $\text{Ent}(L)$  in both  $\Gamma_L$  and  $\overline{\Gamma}_L$ .

### 3.2.4 Rook's graph method

We define the edge-colored graph  $\Xi_L$  from the partial Latin rectangle graph  $\Gamma_L$  by deleting the purple edges, thereby forming an induced subgraph of the rook's graph  $K_r \times K_s$ . Any autotopism of  $L$  is equivalent to an automorphism of  $\Xi_L$ . The converse does not necessarily hold, so after NAUTY has been called, we check each automorphism of  $\Xi_L$  to see if it corresponds to an autotopism of  $L$ . Differing from partial Latin rectangle graphs, we define  $\overline{\Xi}_L$  with

$$V(\overline{\Xi}_L) := \text{Ent}(L) \cup \{R_e : e \in \text{Ent}(L)\} \cup \{S_e : e \in \text{Ent}(L)\} \cup \{k \in [n] : \text{symbol } k \text{ appears in } L\},$$

where the unions above indicate distinct vertex colors, and edge set

$$\begin{aligned} E(\overline{\Xi}_L) := & \{eR_e, eS_e : e \in \text{Ent}(L)\} \\ & \cup \{R_{e_1}R_{e_2} : \text{distinct entries } e_1 \text{ and } e_2 \text{ belong to the same row}\} \\ & \cup \{S_{e_1}S_{e_2} : \text{distinct entries } e_1 \text{ and } e_2 \text{ belong to the same column}\} \\ & \cup \{R_ek : e \in \text{Ent}(L) \text{ and } e \text{ has symbol } k\} \\ & \cup \{S_ek : e \in \text{Ent}(L) \text{ and } e \text{ has symbol } k\}. \end{aligned}$$

It is also possible to use entry invariants to color the vertices in  $E(L)$  in both  $\Xi_L$  and  $\overline{\Xi}_L$ .

## 4 Experimental results

We compare run times of all the methods for computing autotopism groups introduced in Section 3. The experimental platform has an Intel Core i7-4700MQ, 2.40GHz processor (4 physical cores) with 7895MiB RAM running Ubuntu 16.04 LTS (64-bit). In each experiment, we calculate the average run time over 10000 random PLR( $r, s, n$ )s on the following two randomly generated sets, for some fixed parameters  $r, s$ , and  $n$ :

- In *PLR set A*, we begin with an empty  $\text{PLR}(r, s, n)$  and attempt  $x$  times to add an entry chosen uniformly at random from  $[r] \times [s] \times [n]$ . If a clash arises (that is, if the cell is full, or adding the entry would introduce a repeated symbol in a row or column), we do nothing. A random partial Latin rectangle generated in this way could have anywhere from 1 to  $rs$  entries, with the expected number of entries growing as the number of attempts  $x$  increases.
- In *PLR set B*, we generate a random Latin square of order  $n \geq \max\{r, s\}$  by using the Jacobson and Matthews method [37], then delete the last  $n - r$  rows and last  $n - s$  columns to obtain an  $(rs)$ -entry  $\text{PLR}(r, s, n)$ , that is, a Latin rectangle. We then delete random entries until we obtain a  $\text{PLR}(r, s, n)$  with  $x$  entries.

Figure 1 in the Appendix plots the experimental run times for random partial Latin rectangles in  $\text{PLR}(5, 5, 5)$ . We observe how backtracking methods perform significantly better when invariants are used. Thus, for instance, the run time for entrywise backtracking without the use of entry invariants seems to increase exponentially as the number of entries increases. This fact is not so clear for graph theoretic methods, which show unpredictable performance depending on the input.

Results for higher orders are shown in Figure 2, where, for *PLR Set A*, we see fluctuating run times for the alpha-beta backtracking method and bipartite graph method using strong entry invariants. It indicates a high sensitivity to the number of entries. Therefore, if we compute autotopism groups for partial Latin rectangles where entries are added randomly [32], we should be aware of the problem that partial Latin rectangles with many entries may drastically reduce the software’s performance.

When using entry invariants, sometimes we can deduce that the autotopism group is trivial (up to the autotopisms counted in Lemma 2.2). In these cases, further computation is not required. Figure 3 plots the proportion of the time that further computation is required for three example parameter vectors  $(r, s, n)$ . We find that entry invariants are ineffective when a partial Latin rectangle has very few entries (likely because it has non-trivial autotopisms). Further, while strong entry invariants are more efficient to compute than square invariants, Figure 3 shows that square invariants are more useful at reducing subsequent computation when there are many entries.

Table 1 lists the average run times for the McKay, Meynert, and Myrvold (MMM) method, the bipartite graph method, and the partial Latin rectangle (PLR) graph method when computing autotopism groups for partial Latin rectangles with at most two empty cells. We see that the PLR graph method outperforms the other methods in every case. Also clear from Table 1 is the extreme difference between the case  $r = s = n$  (that is, Latin squares) and the other cases. Without entry invariants, the  $(17, 18, 19)$  data set was processed much faster than the  $(7, 7, 7)$ ; these matrices have 306 cells vs. 49 cells, respectively. In the case of Latin squares, the use of a well-chosen entry invariant is essential to both the MMM and PLR graph methods. Further experimental run times for Latin squares are given in Figure 4, where we see that the PLR graph method consistently outperforms the other methods, and that the bipartite graph method performance is unstable. Figure 5 includes run times for Latin squares after we delete one or two entries, where the instability is no longer present.

Table 1 shows significantly worse performance when  $(r, s, n) = (17, 18, 19)$  using the MMM or the PLR graph methods with strong entry invariants. In this setting there is a “trap”: while there are many entries, there are few distinct strong entry invariants, so coloring the vertices for input into NAUTY involves sorting a long repetitive list.



$(r, s, n)$ no. entries	run time ( $\mu\text{s}$ )					
	(17, 18, 19)			(7, 7, 7)		
	$rs - 0$	$rs - 1$	$rs - 2$	$rs - 0$	$rs - 1$	$rs - 2$
MMM (NAUTY)	446.0	203.1	190.3	11087.8	266.4	34.1
MMM (NAUTY, SEI)	1714.0	262.1	293.5	11092.8	227.5	38.7
MMM (NAUTY, sq.)	586.2	301.3	294.9	55.2	32.1	26.5
MMM (NAUTY, SEI, sq.)	577.6	302.8	296.9	56.6	34.9	27.4
Bipartite graph (NAUTY, sq., RC)	777.6	295.0	292.8	5312.0	160.0	23.0
Bipartite graph (NAUTY, sq., RC) e/w	839.8	305.7	298.4	128.0	38.2	29.0
Bipartite graph (NAUTY, sq., RC) $\alpha$ - $\beta$	657.7	300.3	296.6	2230.7	87.8	25.6
PLR graph (NAUTY)	<b>243.4</b>	<b>170.5</b>	<b>171.3</b>	5476.1	154.3	<b>17.0</b>
PLR graph (NAUTY, SEI)	885.4	192.3	183.7	5482.3	37.6	20.1
PLR graph (NAUTY, sq.)	438.1	299.6	296.4	<b>52.0</b>	<b>28.8</b>	24.4
PLR graph (NAUTY, SEI, sq.)	440.7	302.2	299.0	52.8	29.2	25.9

Table 1: Average run times of the MMM, bipartite graph, and PLR graph methods for partial Latin rectangles with few empty cells (PLR set B). The best run times are in bold.

## 5 Conclusions and further work

From this work, we identify the design goals we should have in mind for software for computing autotopism groups of partial Latin rectangles (PLRs):

1. Several methods showed unpredictable performance depending on its input; we should choose a method which has stable performance.
2. We should mathematically account for PLRs with very few entries, which can have many autotopisms.
3. A cheap-to-compute entry invariant (such as the strong entry invariant) is useful for PLRs with an intermediate number of entries.
4. We should use sophisticated entry invariants for PLRs with a large number of entries.
5. It might be possible to identify PLRs with large (or even transitive) autotopism groups, where we could e.g. offload these problematic cases onto e.g. a graphics processing unit (GPU) for separate computation.

The PLR graph method offers reasonable and stable performance under most conditions. It could be improved by

- a) finding mathematical conditions on when automorphisms of non-edge-colored partial Latin rectangle graphs are autotopisms of the corresponding partial Latin rectangle,
- b) developing software for edge-colored graph automorphism (BLISS [38], available from <http://www.tcs.hut.fi/Software/bliss/>), also does not incorporate edge colors), or
- c) developing purpose-built software which adapts the individualization/refinement approach of NAUTY and BLISS to the problem of computing autotopisms of partial Latin rectangles.

Further work on the development of efficient algorithms for the computational processes that have been proposed throughout the paper, along with a study of the reasons that give rise to better or worse experimental results, is also required.

## Acknowledgements

Marbach and Stones's work is partially supported by NSF of China (61602266, 61872201), the Science and Technology Development Plan of Tianjin (17JCYBJC15300, 16JCYBJC41900), and the Fundamental Research Funds for the Central Universities and SAFEA: Overseas Young Talents in Cultural and Educational Sector. Stones is also supported by the Thousand Youth Talents Plan in Tianjin.

Stones was supported by her NSFC Research Fellowship for International Young Scientists (grant numbers: 11450110409, 11550110491), and the Thousand Youth Talents Plan in Tianjin.

Falcón's work is partially supported by the research project FQM-016 from Junta de Andalucía, and the Departmental Research Budget of the Department of Applied Mathematics I of the University of Seville.

The authors thank Zhuanhao Wu for assistance coding.

## References

- [1] Stones DS. The many formulae for the number of Latin rectangles. *Electron. J. Combin.* 2010; 17: A1.
- [2] Artzy R. A note on the automorphisms of special loops. *Riveon Lematematika* 1954; 8: 81. In Hebrew.
- [3] Bailey RA. Latin squares with highly transitive automorphism groups. *J. Aust. Math. Soc.* 1982; 33: 18-22.
- [4] Browning J, Stones DS, Wanless IM. Bounds on the number of autotopisms and subsquares of a Latin square. *Combinatorica* 2013; 33: 11–22.
- [5] Bryant D, Buchanan M, Wanless IM. The spectrum for quasigroups with cyclic automorphisms and additional symmetries. *Discrete Math.* 2009; 304(4): 821-833.
- [6] Cavenagh N, Stones DS. Near-automorphisms of Latin squares. *J. Combin. Des.* 2011; 19: 355-377.
- [7] Drisko AA. Loops of order  $p^n + 1$  with transitive automorphism groups. *Adv. Math.* 1997; 128: 36-39.
- [8] Falcón RM. Cycle structures of autotopisms of the Latin squares of order up to 11. *Ars Combin.* 2012; 103: 239-256.
- [9] Falcón RM, Martín-Morales J. Gröbner bases and the number of Latin squares related to autotopisms of Order  $\leq 7$ . *J. Symbolic Comput.* 2007; 42(11-12): 1142-1154.
- [10] Falcón RM, Núñez J. Partial Latin squares having a Santilli's autotopism in their autotopism groups. *J. Dyn. Syst. Geom. Theor.* 2007; 5: 19-32.
- [11] Ihrig EC, Ihrig BM. The recognition of symmetric Latin squares. *J. Combin. Des.* 2008; 16(4): 291-300.
- [12] Kerby B, Smith JDH. Quasigroup automorphisms and symmetric group characters. *Comment. Math. Univ. Carol.* 2010; 51(2): 279-286.

- [13] Kerby B, Smith JDH. Quasigroup automorphisms and the Norton-Stein complex. *Proc. Amer. Math. Soc.* 2010; 138: 3079-3088.
- [14] Kotlar D. Parity types, cycle structures and autotopisms of Latin squares. *Electron. J. Combin.* 2012; 19(3). P10.
- [15] McKay BD, Wanless IM, Xiande Z. The order of automorphisms of quasigroups. *J. Combin. Des.* 2015; 23: 275-288.
- [16] Mendis MJL, Wanless IM. Autoparatopisms of quasigroups and Latin squares. *J. Combin. Des.* 2017; 25: 51-74.
- [17] Meng H, Zheng Y, Zheng Y. The classification construction and the non-isomorphism counting of symmetric Latin square. *Adv. Stud. Contemp. Math. (Kyungshang)* 2008; 17(2): 169-179.
- [18] Stones DS, Vojtěchovský P, Wanless IM. Cycle structure of autotopisms of quasigroups and Latin squares. *J. Combin. Des.* 2012; 20: 227-263.
- [19] Wanless IM, Ihrig EC. Symmetries that Latin squares inherit from 1-factorizations. *J. Combin. Des.* 2005; 13: 157-172.
- [20] Falcón RM. Latin squares associated to principal autotopisms of long cycles. Application in cryptography. In: *Proc. Transgressive Computing 2006: a conference in honor of Jean Della Dora*; 2006: 213-230.
- [21] Yan M, Feng J, Marbach T, Stones R, Wang G, Liu X. Gecko: A Resilient Dispersal Scheme for Multi-Cloud Storage. *IEEE Access* 2019; 7: 77387-77397.
- [22] Stones RJ, Su M, Liu X, Wang G, Lin S. A Latin square autotopism secret sharing scheme. *Des. Codes Cryptogr.* 2015; 35: 1–16.
- [23] Yi L, Stones RJ, Wang G. Two-erasure codes from 3-plexes. In: *Proc. Network and Parallel Computing*; 2019.
- [24] Stones RJ. K-plex 2-Erasure Codes and Blackburn Partial Latin Squares. Submitted for publication.
- [25] Wanless IM. A partial Latin squares problem posed by Blackburn. *Bull. Inst. Comb. Appl.* 2004; 42: 76-80.
- [26] Andres S, Falcón R. Colouring games based on autotopisms of Latin hyper-rectangles. *Quaest. Math.* 2019; 42: 953-975.
- [27] Andres S, Falcón R. Autotopism stabilized colouring games on rook's graphs. *Discrete Appl. Math.* 2019; 266: 200-212.
- [28] Falcón RM. The set of autotopisms of partial Latin squares. *Discrete Math.* 2013; 313(11): 1150-1161.
- [29] Falcón RM, Stones RJ. Classifying partial Latin rectangles. *Electron. Notes Discrete Math.* 2015; 49: 765-771.
- [30] Falcón RM, Stones RJ. Partial Latin rectangle graphs and autoparatopism groups of partial Latin rectangles with trivial autotopism groups. *Discrete Math.* 2017; 340(6): 1242-1260.

- [31] McKay BD, Meynert A, Myrvold W. Small Latin squares, quasigroups, and loops. *J. Combin. Des.* 2007; 15: 98-119.
- [32] Stones DS. Symmetries of partial Latin squares. *European J. Combin.* 2013; 34(7): 1092-1107.
- [33] Kotlar D. Computing the autotopy group of a Latin square by cycle structure. *Discrete Math.* 2014; 331: 74–82.
- [34] Cameron PC. Asymmetric Latin squares, Steiner triple systems, and edge-parallelisms. arXiv:1507.02190 [math.CO]; 2015.
- [35] McKay BD, Wanless IM. On the number of Latin squares. *Ann. Comb.* 2005; 9: 335-344.
- [36] McKay BD, Piperno A. Practical graph isomorphism, II. *J. Symb. Comput.* 2014; 60: 94-112.
- [37] Jacobson M, Matthews P. Generating uniformly distributed Latin squares. *J. Combin. Des.* 1996; 4(6): 405-437.
- [38] Junttila T, Kaski P. Engineering an efficient canonical labeling tool for large and sparse graphs. In: Proc. SIAM Workshop on Algorithm Engineering and Experiments; 2007: 135–149.

## A Experimental plots

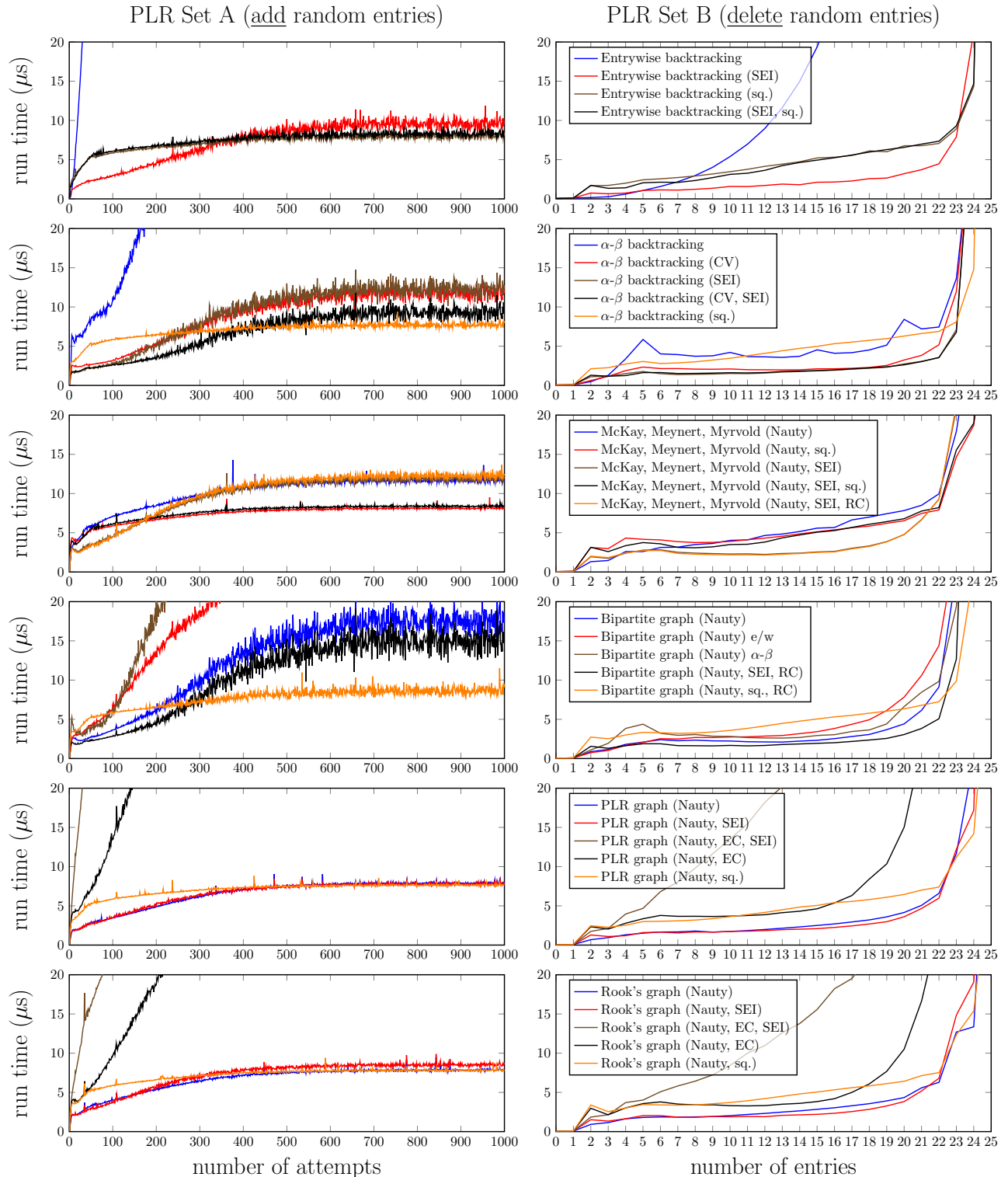


Figure 1: The average run time of various methods used to compute the autotopism group, with or without the use of strong entry invariants (SEI) and/or square invariants (sq.); parameters  $(r, s, n) = (5, 5, 5)$ . Other acronyms: RC = “row and column invariants”; e/w = “entrywise”; EC = “edge colors”; CV = “column vectors”. Some experimental results are omitted for space reasons.

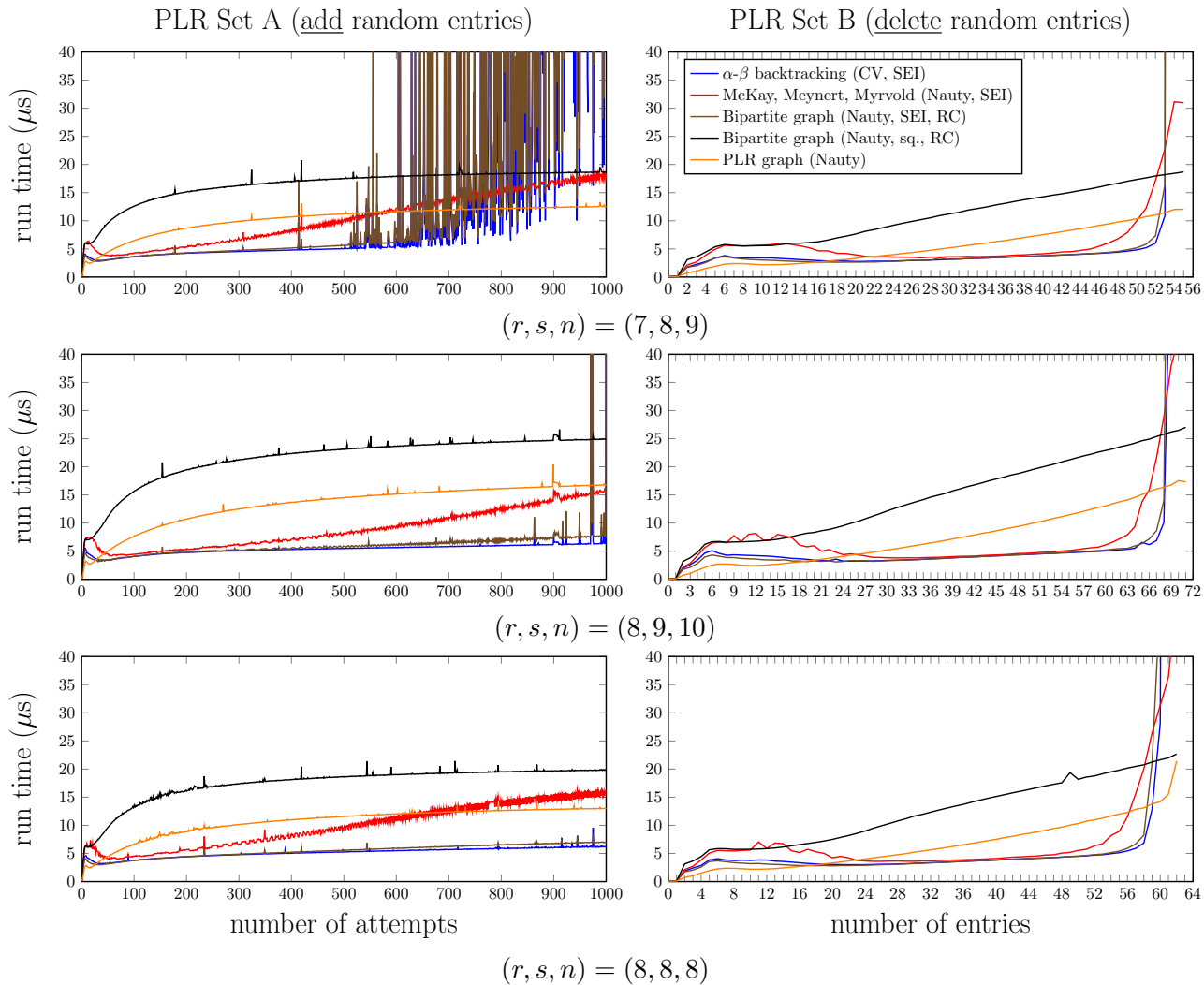


Figure 2: The average run time of the remaining methods. For PLR set B, the 63- and 64-entry data points for  $(r, s, n) = (8, 8, 8)$  and the 72-entry data points for  $(r, s, n) = (8, 9, 10)$  are omitted as they took too long to compute.

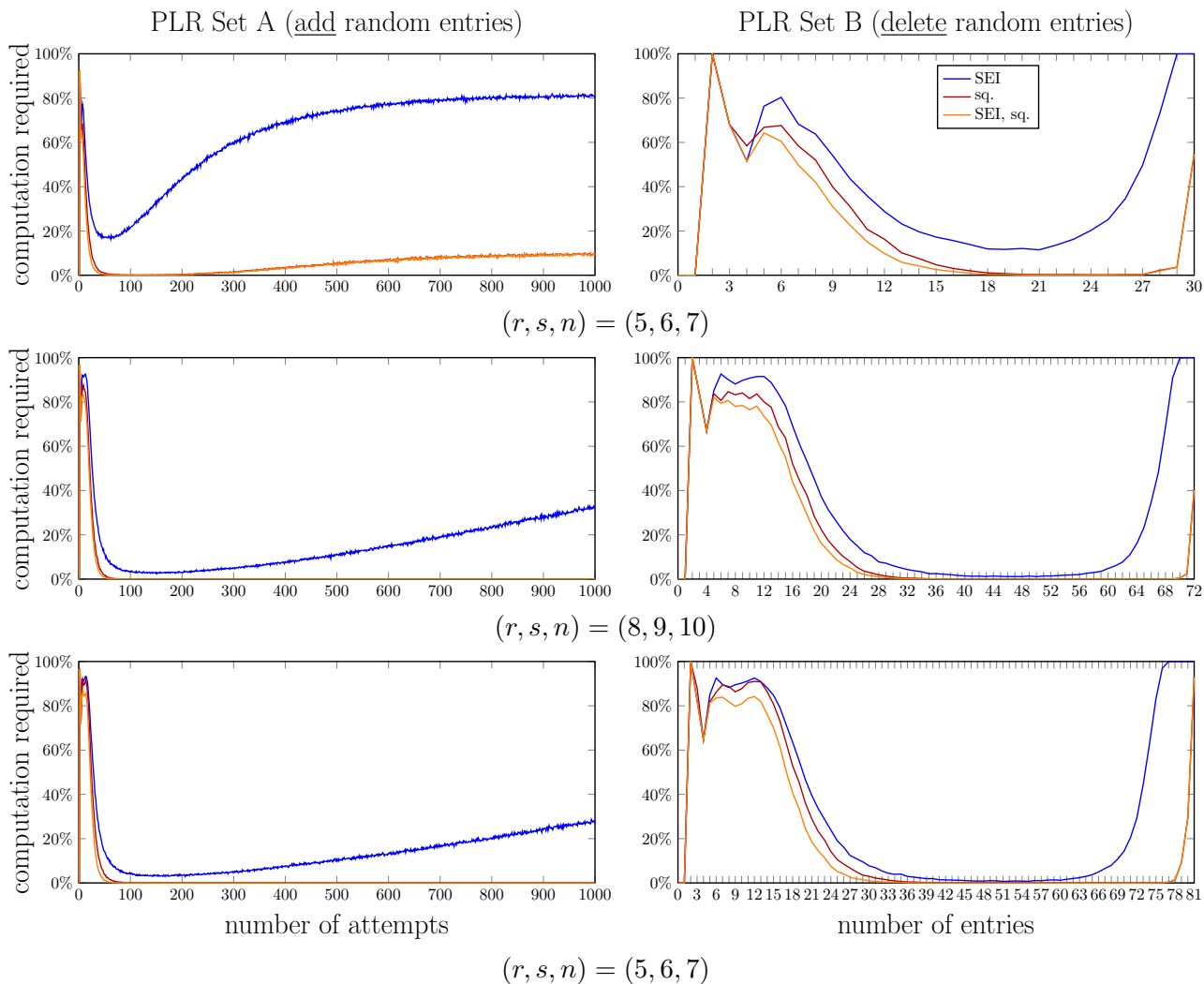


Figure 3: The proportion of time (over 10000 samples) computation is required to compute the autotopism group despite using strong entry invariants (SEI) and/or square invariants (sq.).

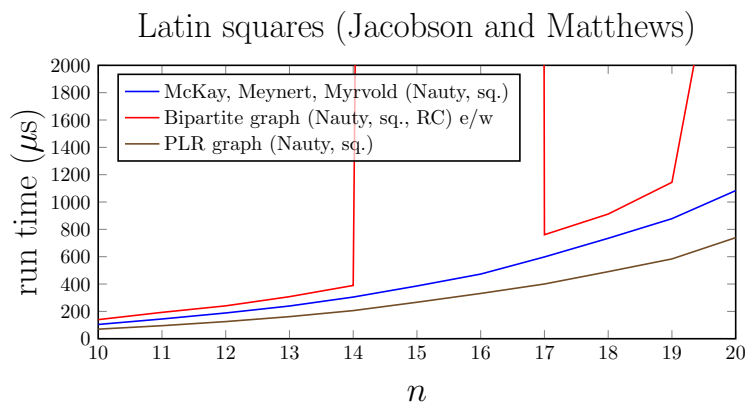


Figure 4: The average run time of the MMM method, the bipartite graph method using entrywise backtracking, and the PLR graph method, each using square entry invariants, for random Latin squares of varying order  $n$  (here,  $r = s = n$ ).

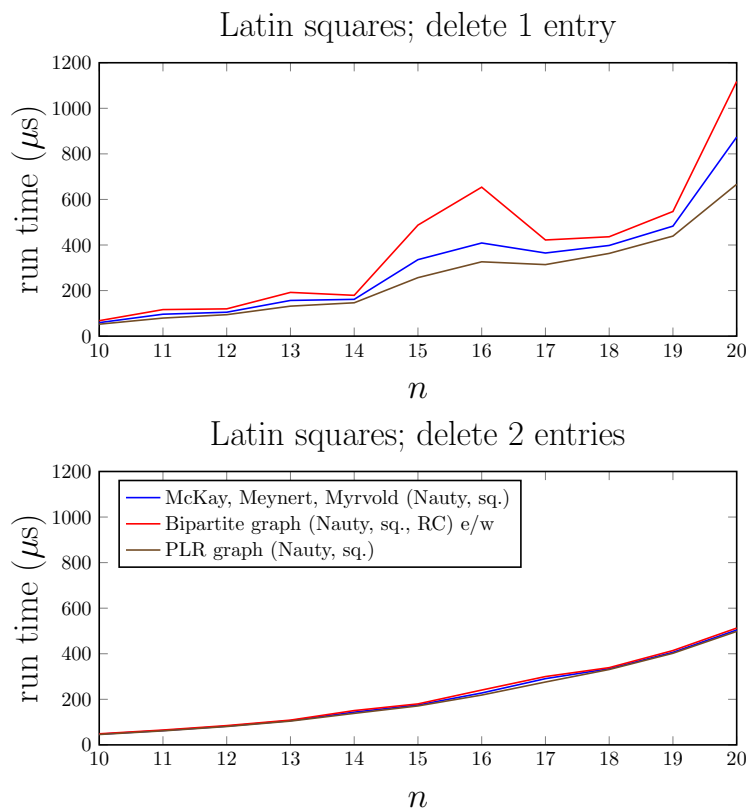


Figure 5: The average run time of the MMM method, the bipartite graph method using entrywise backtracking, and the PLR graph method, each using square entry invariants, for random Latin squares of varying order  $n$  after deleting either one entry (top plot) or two entries (bottom plot).