Computing autotopism groups of partial Latin rectangles: a pilot study

Raúl M. Falcón (U. Seville);
Daniel Kotlar (Tel-Hai College);
Rebecca J. Stones (Nankai U.)

19 December 2016

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

Partial Latin rectangles

An $r \times s$ partial Latin rectangle is an $r \times s$ matrix containing symbols from $[n] \cup \{\cdot\}$ such that each row and each column contains at most one copy of any symbol in $[n]$.

$\cdot$	5	4	3	2
5	$\cdot$	$\cdot$	$\cdot$	1
4	$\cdot$	$\cdot$	1	$\cdot$

Partial Latin rectangles

An $r \times s$ partial Latin rectangle is an $r \times s$ matrix containing symbols from $[n] \cup \{\cdot\}$ such that each row and each column contains at most one copy of any symbol in $[n]$.

$\cdot$	5	4	3	2
5	$\cdot$	$\cdot$	$\cdot$	1
4	$\cdot$	$\cdot$	1	$\cdot$

Every partial Latin rectangle $L \in \mathrm{PLR}(r, s, n)$ is uniquely determined by its entry set:

$$\mathrm{Ent}(L) := \{\overbrace{(i, j, L[i, j])}^{\text{entry}} : i \in [r], j \in [l], \text{ and } L[i, j] \in [n]\}.$$

$$\mathrm{Ent}(\text{above}) = \left\{ \begin{matrix} (2,5,1), \\ (3,4,1), \end{matrix} \quad (1,5,2), \quad (1,4,3), \quad \begin{matrix} (1,3,4), \\ (3,1,4), \end{matrix} \quad \begin{matrix} (1,2,5), \\ (2,1,5) \end{matrix} \right\}$$

Isotopisms and autotopisms

$$\left\{\begin{array}{l} r \times s \text{ partial Latin rectangles} \\ \text{on symbol set } [n] \end{array}\right\}$$

The isotopism $\theta := (\alpha, \beta, \gamma) \in S_r \times S_s \times S_n$ acts on $\mathrm{PLR}(r, s, n)$.

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

swap first two rows $\alpha = (12)$

swap last two columns $\beta = (56)$

do nothing to symbols $\gamma = \mathrm{id}$

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

Isotopisms and autotopisms

$$\left\{\begin{array}{l} r \times s \text{ partial Latin rectangles} \\ \text{on symbol set } [n] \end{array}\right\}$$

The isotopism $\theta := (\alpha, \beta, \gamma) \in S_r \times S_s \times S_n$ acts on $\mathrm{PLR}(r, s, n)$.

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

swap first two rows $\alpha = (12)$

swap last two columns $\beta = (56)$

do nothing to symbols $\gamma = \mathrm{id}$

$\longrightarrow$

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

And, in some cases, we can apply an isotopism θ and end up back where we started $\implies$ θ is an autotopism.

Isotopisms and autotopisms

$$\left\{\begin{array}{l} r \times s \text{ partial Latin rectangles} \\ \text{on symbol set } [n] \end{array}\right\}$$

The isotopism $\theta := (\alpha, \beta, \gamma) \in S_r \times S_s \times S_n$ acts on $\mathrm{PLR}(r, s, n)$.

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

swap first two rows $\alpha = (12)$

swap last two columns $\beta = (56)$

do nothing to symbols $\gamma = \mathrm{id}$

$\longrightarrow$

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

And, in some cases, we can apply an isotopism θ and end up back where we started $\implies$ θ is an autotopism.

The set of autotopisms form a group, named the autotopism group.

1	2
3	4

This member of $\mathrm{PLR}(2, 2, 4)$ has 4 autotopisms.

- $(\mathrm{id}, \mathrm{id}, \mathrm{id})$,
- $((12), \mathrm{id}, (13)(24))$,
- $(\mathrm{id}, (12), (12)(34))$,
- $((12), (12), (14)(23))$,

1	2
3	4

This member of $\mathrm{PLR}(2,2,4)$ has 4 autotopisms.

- $(\mathrm{id}, \mathrm{id}, \mathrm{id})$,
- $((12), \mathrm{id}, (13)(24))$,
- $(\mathrm{id}, (12), (12)(34))$,
- $((12), (12), (14)(23))$,

...forming a group isomorphic to $C_2 \times C_2$.

1	2
3	4

This member of $\mathrm{PLR}(2,2,4)$ has 4 autotopisms.

- $(\mathrm{id}, \mathrm{id}, \mathrm{id})$,
- $((12), \mathrm{id}, (13)(24))$,
- $(\mathrm{id}, (12), (12)(34))$,
- $((12), (12), (14)(23))$,

...forming a group isomorphic to $C_2 \times C_2$.

Note: The row and column permutations determine the autotopism.

How to *efficiently* compute the autotopism group?

Input: partial Latin rectangle.
Output: its autotopism group.

How to *efficiently* compute the autotopism group?

Input: partial Latin rectangle.
Output: its autotopism group.

By the looks of things, the answer is...

How to *efficiently* compute the autotopism group?

Input: partial Latin rectangle.
Output: its autotopism group.

By the looks of things, the answer is...

Basically, the answer depends on the partial Latin rectangle.

How to *efficiently* compute the autotopism group?

Input: partial Latin rectangle.
Output: its autotopism group.

By the looks of things, the answer is...

Basically, the answer depends on the partial Latin rectangle.

This work is a "pilot study" to (a) identify design goals of future software for computing the autotopism group, and (b) eliminate unpromising methods.

How to *efficiently* compute the autotopism group?

Input: partial Latin rectangle.
Output: its autotopism group.

By the looks of things, the answer is...

Basically, the answer depends on the partial Latin rectangle.

This work is a "pilot study" to (a) identify design goals of future software for computing the autotopism group, and (b) eliminate unpromising methods.

We experimentally compare 6 families of methods...

Backtracking methods...

- **Family 1: Alpha-beta backtracking.**

Backtracking methods...

Family 1: Alpha-beta backtracking.

At each level of the α search tree,
we designate

$$\text{row } i \quad \overset{\alpha}{\longmapsto} \quad \text{row } a$$

provided it doesn't clash.

$$\begin{array}{c|c|c|c|}
 & 3 & 1 & 4 \\
\hline
a & 1 & \cdot & \cdot \\
\hline
i & 2 & \cdot & \cdot \\
\hline
\end{array}$$

Backtracking methods...

Family 1: Alpha-beta backtracking.

At each level of the α search tree, we designate

$$\text{row } i \quad \overset{\alpha}{\mapsto} \quad \text{row } a$$

provided it doesn't clash.

$$\begin{array}{c|ccc} & 3 & 1 & 4 \\ a & 1 & \cdot & \cdot \\ i & 2 & \cdot & \cdot \end{array}$$

Once α is determined...

At each level of the β search tree, we designate

$$\text{column } j \quad \overset{\beta}{\mapsto} \quad \text{column } b$$

provided it doesn't clash.

$$\begin{array}{ccc} & j & b \\ \hline 3 & 1 & 4 \\ 1 & \cdot & \cdot \\ 2 & \cdot & \cdot \end{array}$$

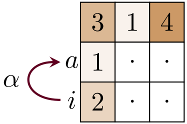
Backtracking methods...

Family 1: Alpha-beta backtracking.

At each level of the α search tree,
we designate

$$\text{row } i \quad \overset{\alpha}{\longmapsto} \quad \text{row } a$$

provided it doesn't clash.

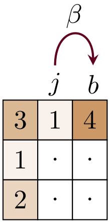


Once α is determined...

At each level of the β search tree,
we designate

$$\text{column } j \quad \overset{\beta}{\longmapsto} \quad \text{column } b$$

provided it doesn't clash.



Then we check if $(\alpha, \beta, ??)$ is an autotopism.

Backtracking methods...

- **Family 2: Entrywise backtracking.**

Backtracking methods...

Family 2: Entrywise backtracking.

At each level of the search tree,
we designate

$$\text{entry } (i, j, L[i,j]) \quad \overset{\theta}{\mapsto} \quad \text{entry } (a, b, L[a,b])$$

provided it doesn't clash.

$$\alpha(i) = a$$
$$\beta(j) = b$$
$$\gamma(L[i,j]) = L[a,b]$$

$$\begin{array}{c|ccc|}
 & j & & b \\
\hline
a & 3 & 1 & 4 \\
i & 1 & \cdot & \cdot \\
 & 2 & \cdot & \cdot \\
\hline
\end{array}$$

Graph methods...

- **Family 3: McKay, Meynert, and Myrvold method.**

Graph methods...

Family 3: McKay, Meynert, and Myrvold method.

Vertex set:

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

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

Edge set:

$$\begin{aligned} &\{R_i L[i,j]\colon\ (i,j,L[i,j]) \in \mathrm{Ent}(L)\} \\ \cup\, &\{S_j L[i,j]\colon\ (i,j,L[i,j]) \in \mathrm{Ent}(L)\} \\ \cup\, &\{N_{L[i,j]} L[i,j]\colon\ (i,j,L[i,j]) \in \mathrm{Ent}(L)\}. \end{aligned}$$

Graph methods...

Family 3: McKay, Meynert, and Myrvold method.

Vertex set:

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

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

Edge set:

$$\begin{aligned}&\{R_i L[i,j]\colon\ (i,j,L[i,j]) \in \mathrm{Ent}(L)\}\\ \cup\ &\{S_j L[i,j]\colon\ (i,j,L[i,j]) \in \mathrm{Ent}(L)\}\\ \cup\ &\{N_{L[i,j]} L[i,j]\colon\ (i,j,L[i,j]) \in \mathrm{Ent}(L)\}.\end{aligned}$$

The <u>automorphism group of this graph</u> is isomorphic to the <u>autotopism group of the partial Latin rectangle</u>.

Graph methods...

Family 3: McKay, Meynert, and Myrvold method.

Vertex set:

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

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

Edge set:

$$\begin{aligned}&\{R_i L[i,j]\colon\ (i,j,L[i,j]) \in \mathrm{Ent}(L)\}\\ \cup\, &\{S_j L[i,j]\colon\ (i,j,L[i,j]) \in \mathrm{Ent}(L)\}\\ \cup\, &\{N_{L[i,j]} L[i,j]\colon\ (i,j,L[i,j]) \in \mathrm{Ent}(L)\}.\end{aligned}$$

The automorphism group of this graph is isomorphic to the autotopism group of the partial Latin rectangle.
We compute this using **Nauty**.

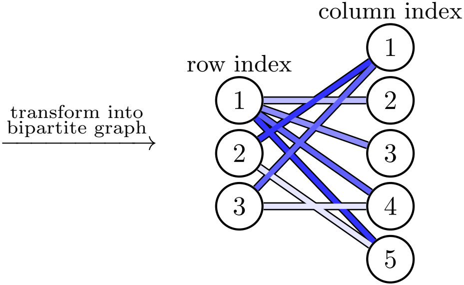
Graph methods...

- **Family 4: Bipartite graph method.**

Graph methods...

Family 4: Bipartite graph method.

·	5	4	3	2
5	·	·	·	1
4	·	·	1	·

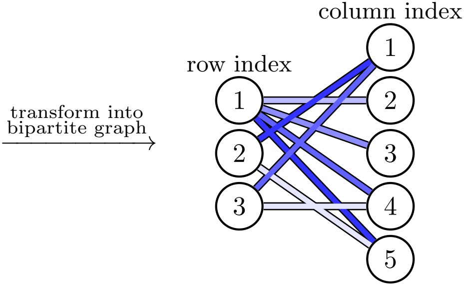


(Edges are colored to illustrate construction.)

Graph methods...

Family 4: Bipartite graph method.

·	5	4	3	2
5	·	·	·	1
4	·	·	1	·

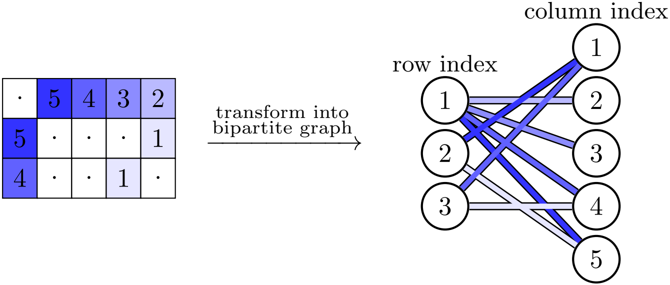


(Edges are colored to illustrate construction.)

Then compute the automorphism group of the bipartite graph using **Nauty**.

Graph methods...

Family 4: Bipartite graph method.

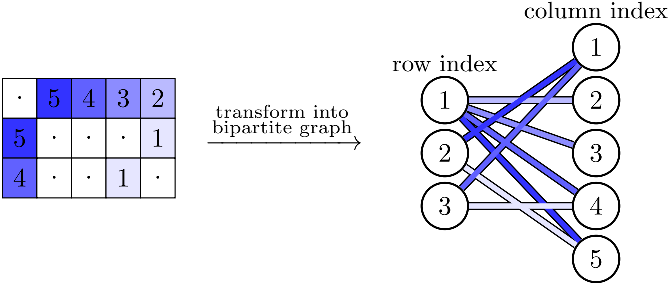


(Edges are colored to illustrate construction.)

Then compute the automorphism group of the bipartite graph using **Nauty**. Filter out non-autotopisms.

Graph methods...

Family 4: Bipartite graph method.



(Edges are colored to illustrate construction.)

Then compute the automorphism group of the bipartite graph using **Nauty**. Filter out non-autotopisms.

Nauty can also return (a) the row/column orbits or (b) entry orbits, under the autotopism group. We can alternatively use alpha-beta or entrywise backtracking on these orbits.

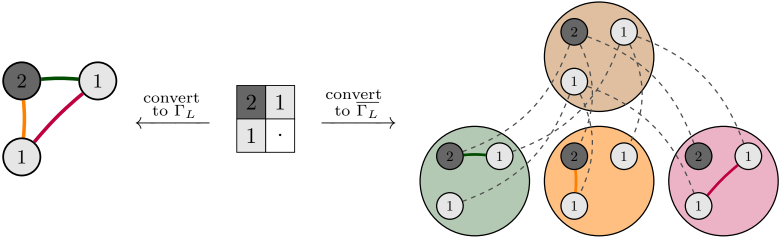
Graph methods...

- **Family 5: Partial Latin rectangle graph method.**

Graph methods...

Family 5: Partial Latin rectangle graph method.

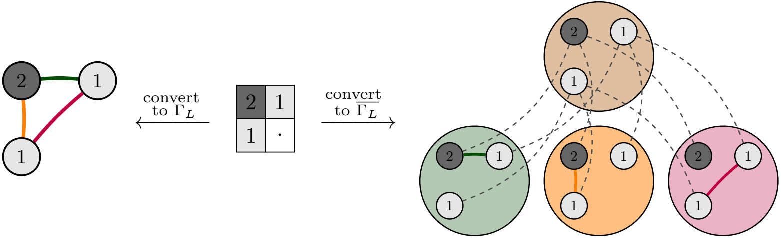
Partial Latin rectangle graph Γ_L (left):



Graph methods...

Family 5: Partial Latin rectangle graph method.

Partial Latin rectangle graph Γ_L (left):

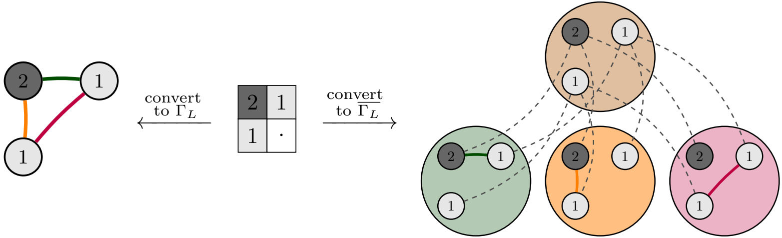


Then compute the automorphism group of Γ_L using **Nauty**. Filter out non-autotopisms [autoparatopisms & graph artifacts].

Graph methods...

Family 5: Partial Latin rectangle graph method.

Partial Latin rectangle graph Γ_L (left):



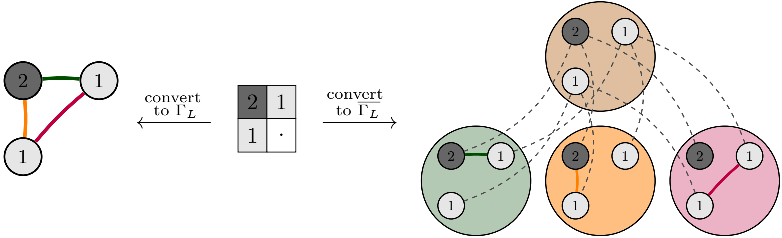
Then compute the automorphism group of Γ_L using **Nauty**. Filter out non-autotopisms [autoparatopisms & graph artifacts].

Also edge-colored version $\overline{\Gamma_L}$ (right), because **Nauty** doesn't allow edge colors.

Graph methods...

Family 5: Partial Latin rectangle graph method.

Partial Latin rectangle graph Γ_L (left):



Then compute the automorphism group of Γ_L using **Nauty**. Filter out non-autotopisms [autoparatopisms & graph artifacts].

Also edge-colored version $\overline{\Gamma_L}$ (right), because **Nauty** doesn't allow edge colors. No filtering required—**Nauty** output is autotopism group.

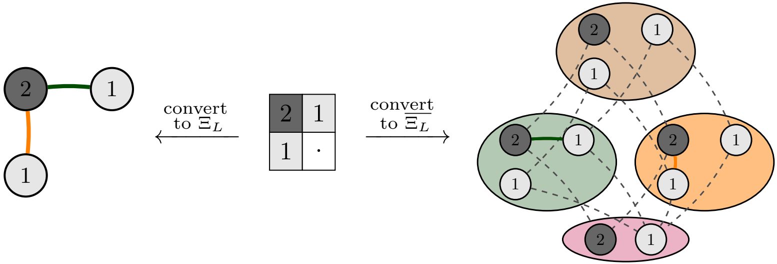
Graph methods...

- **Family 6: Rook's graph method.**

Graph methods...

Family 6: Rook's graph method.

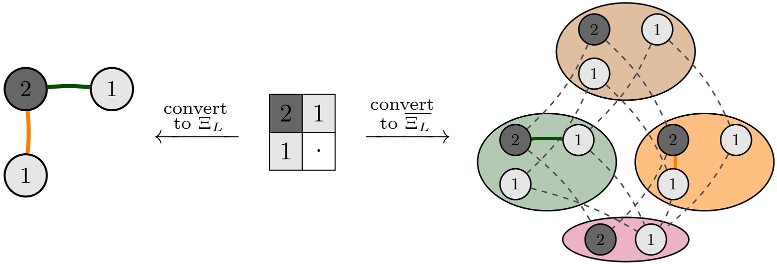
Induced subgraph of the rook's graph Ξ_L (left):



Graph methods...

Family 6: Rook's graph method.

Induced subgraph of the rook's graph Ξ_L (left):

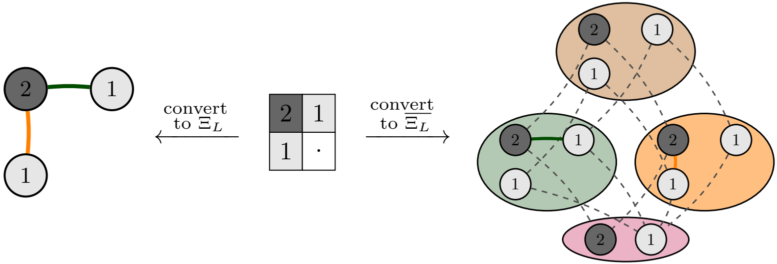


Then compute the automorphism group of Ξ_L using **Nauty**.
Filter out non-autotopisms.

Graph methods...

Family 6: Rook's graph method.

Induced subgraph of the rook's graph Ξ_L (left):



Then compute the automorphism group of Ξ_L using **Nauty**.
Filter out non-autotopisms.

Also edge-colored version $\overline{\Xi_L}$ (right)...

Summary thus far...

We have six families of methods:

- *Backtracking*: alpha-beta and entrywise.
- *Graphical*: bipartite graph, MMM graph, PLR graph, rook's graph.

Summary thus far...

We have six families of methods:

- *Backtracking*: alpha-beta and entrywise.
- *Graphical*: bipartite graph, MMM graph, PLR graph, rook's graph.

Each of these methods can be improved by using invariants, properties of partial Latin rectangles which are invariant under autotopisms.

Summary thus far...

We have six families of methods:

- *Backtracking*: alpha-beta and entrywise.
- *Graphical*: bipartite graph, MMM graph, PLR graph, rook's graph.

Each of these methods can be improved by using invariants, properties of partial Latin rectangles which are invariant under autotopisms.

We consider two invariants.

Summary thus far...

We have six families of methods:

- *Backtracking*: alpha-beta and entrywise.
- *Graphical*: bipartite graph, MMM graph, PLR graph, rook's graph.

Each of these methods can be improved by using invariants, properties of partial Latin rectangles which are invariant under autotopisms.

We consider two invariants. (More sophisticated invariants may improve run-times, but will improve run-times for every method.)

Strong entry invariants...

For entry (i, j, k) we define the *strong entry invariant* as the vector (a, b, c) where

- a is the number of entries in row i,
- b is the number of entries in column j,
- c is the number of copies of symbol k in the partial Latin rectangle.

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

$\xrightarrow[\text{relabeled } 1, 2, \ldots]{\text{strong entry invariants}}$

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

Strong entry invariants...

For entry (i, j, k) we define the *strong entry invariant* as the vector (a, b, c) where

- a is the number of entries in row i,
- b is the number of entries in column j,
- c is the number of copies of symbol k in the partial Latin rectangle.

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

$\xrightarrow[\text{relabeled } 1, 2, \ldots]{\text{strong entry invariants}}$

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

...useless for Latin rectangles (i.e., no empty cells and number columns = number symbols).

Square invariants...

The entry (i, j, k) belongs to exactly $(r-1)(s-1)$ 2×2 sub-matrices, a typical one looking like:

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

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) $x = y$.

Square invariants...

The entry (i, j, k) belongs to exactly $(r-1)(s-1)$ 2×2 sub-matrices, a typical one looking like:

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

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 $(r-1)(s-1)$.

Square invariants...

The entry (i, j, k) belongs to exactly $(r-1)(s-1)$ 2×2 sub-matrices, a typical one looking like:

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

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 $(r-1)(s-1)$.

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

$\xrightarrow{\text{square invariants relabeled } 1, 2, \ldots}$

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

We call this length-32 vector the square invariant.

Row and column invariants

Row and column invariants

Given some kind of entry invariant:

- the multiset of entry invariants in a given row is preserved under autotopisms and

Row and column invariants

Given some kind of entry invariant:

- the multiset of entry invariants in a given row is preserved under autotopisms and
- the multiset of entry invariants in a given column is preserved under autotopisms.

Row and column invariants

Given some kind of entry invariant:

- the multiset of entry invariants in a given row is preserved under autotopisms and
- the multiset of entry invariants in a given column is preserved under autotopisms.

In the alpha-beta backtracking method, once α is determined, then...

...when we decide that $\beta(j) = b$, the filled/unfilled cells in column j map to filled/unfilled cells in column b.

Row and column invariants

Given some kind of entry invariant:

- the multiset of entry invariants in a given row is preserved under autotopisms and
- the multiset of entry invariants in a given column is preserved under autotopisms.

In the alpha-beta backtracking method, once α is determined, then...

...when we decide that $\beta(j) = b$, the filled/unfilled cells in column j map to filled/unfilled cells in column b. This can also be used to improve the computation.

Experiments...

So we have six families of methods, and two invariants, etc.

Experiments...

So we have six families of methods, and two invariants, etc.

Putting these together gives around 48 different ways of computing the autotopism group of a partial Latin rectangle.

Experiments...

So we have six families of methods, and two invariants, etc.

Putting these together gives around 48 different ways of computing the autotopism group of a partial Latin rectangle.

Q: **Which is the best?**

Experiments...

So we have six families of methods, and two invariants, etc.

Putting these together gives around 48 different ways of computing the autotopism group of a partial Latin rectangle.

Q: Which is the best?

Experiment set 1: Start with empty $\mathrm{PLR}(r, s, n)$ and add try to add entry $(i, j, k) \in [r] \times [s] \times [n]$ randomly.

Experiments...

So we have six families of methods, and two invariants, etc.

Putting these together gives around 48 different ways of computing the autotopism group of a partial Latin rectangle.

Q: Which is the best?

Experiment set 1: Start with empty $\mathrm{PLR}(r,s,n)$ and add try to add entry $(i,j,k) \in [r] \times [s] \times [n]$ randomly.

Experiment set 2: Start with random $r \times s$ submatrix of a $\mathrm{LS}(n)$ and delete entries randomly.

Experiments...

So we have six families of methods, and two invariants, etc.

Putting these together gives around 48 different ways of computing the autotopism group of a partial Latin rectangle.

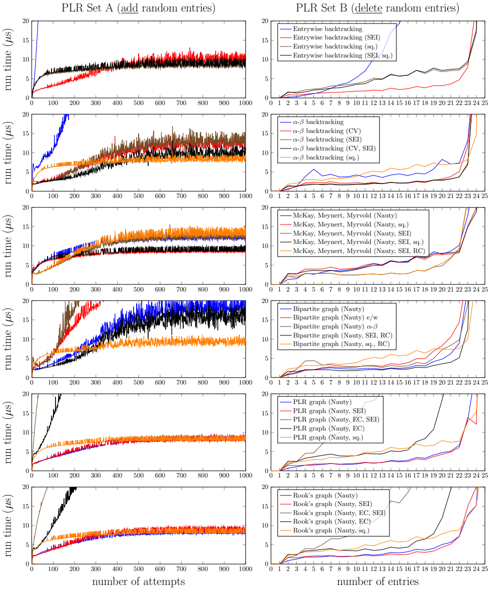
Q: Which is the best?

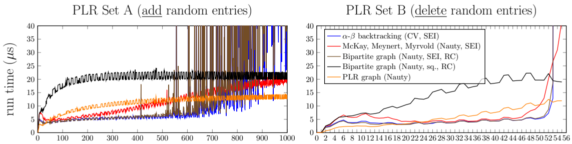
Experiment set 1: Start with empty $\mathrm{PLR}(r, s, n)$ and add try to add entry $(i, j, k) \in [r] \times [s] \times [n]$ randomly.

Experiment set 2: Start with random $r \times s$ submatrix of a $\mathrm{LS}(n)$ and delete entries randomly.

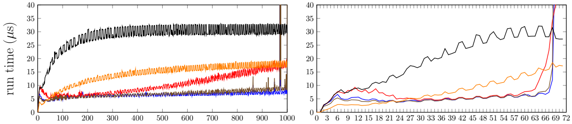
(Each data point is averaged over 10000 samples.)

$(r, s, n) = (5, 5, 5)$; discard bad methods

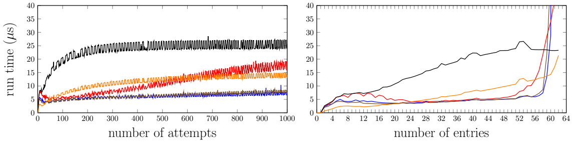




(a) $(r, s, n) = (7, 8, 9)$



(b) $(r, s, n) = (8, 9, 10)$



(c) $(r, s, n) = (8, 8, 8)$

Figure: Average run times of the remaining methods.

Squares vs. rectangles

	run time (μs)					
(r, s, n)	$(17, 18, 19)$			$(7, 7, 7)$		
no. entries	$rs - 0$	$rs - 1$	$rs - 2$	$rs - 0$	$rs - 1$	$rs - 2$
MMM (Nauty)	58.2	30.4	29.0	1203.0	30.8	5.6
MMM (Nauty, SEI)	206.5	42.6	42.7	1200.6	24.2	5.7
MMM (Nauty, sq.)	42.4	33.1	33.0	6.6	2.8	2.2
MMM (Nauty, SEI, sq.)	42.2	33.6	33.6	6.5	3.0	2.2
PLR graph (Nauty)	**29.7**	**18.5**	**18.5**	840.3	22.1	2.2
PLR graph (Nauty, SEI)	110.7	20.7	20.1	837.7	4.3	2.6
PLR graph (Nauty, sq.)	35.5	33.0	33.1	**5.3**	**2.2**	2.1
PLR graph (Nauty, SEI, sq.)	36.1	34.0	33.7	5.4	2.3	**2.1**

Squares vs. rectangles

	run time (μs)					
(r, s, n)	$(17, 18, 19)$			$(7, 7, 7)$		
no. entries	$rs - 0$	$rs - 1$	$rs - 2$	$rs - 0$	$rs - 1$	$rs - 2$
MMM (Nauty)	58.2	30.4	29.0	1203.0	30.8	5.6
MMM (Nauty, SEI)	206.5	42.6	42.7	1200.6	24.2	5.7
MMM (Nauty, sq.)	42.4	33.1	33.0	6.6	2.8	2.2
MMM (Nauty, SEI, sq.)	42.2	33.6	33.6	6.5	3.0	2.2
PLR graph (Nauty)	**29.7**	**18.5**	**18.5**	840.3	22.1	2.2
PLR graph (Nauty, SEI)	110.7	20.7	20.1	837.7	4.3	2.6
PLR graph (Nauty, sq.)	35.5	33.0	33.1	**5.3**	**2.2**	2.1
PLR graph (Nauty, SEI, sq.)	36.1	34.0	33.7	5.4	2.3	**2.1**

- PLR beats MMM method (to my surprise!).

Squares vs. rectangles

(r, s, n)	run time (μs)					
	$(17, 18, 19)$			$(7, 7, 7)$		
no. entries	$rs - 0$	$rs - 1$	$rs - 2$	$rs - 0$	$rs - 1$	$rs - 2$
MMM (Nauty)	58.2	30.4	29.0	1203.0	30.8	5.6
MMM (Nauty, SEI)	206.5	42.6	42.7	1200.6	24.2	5.7
MMM (Nauty, sq.)	42.4	33.1	33.0	6.6	2.8	2.2
MMM (Nauty, SEI, sq.)	42.2	33.6	33.6	6.5	3.0	2.2
PLR graph (Nauty)	**29.7**	**18.5**	**18.5**	840.3	22.1	2.2
PLR graph (Nauty, SEI)	110.7	20.7	20.1	837.7	4.3	2.6
PLR graph (Nauty, sq.)	35.5	33.0	33.1	**5.3**	**2.2**	2.1
PLR graph (Nauty, SEI, sq.)	36.1	34.0	33.7	5.4	2.3	**2.1**

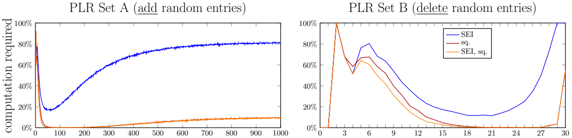
- PLR beats MMM method (to my surprise!).
- Massive difference between Latin squares and everything else.

Squares vs. rectangles

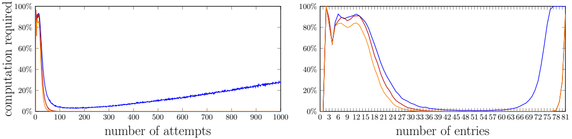
(r,s,n)	run time (μs)					
	$(17,18,19)$			$(7,7,7)$		
no. entries	$rs-0$	$rs-1$	$rs-2$	$rs-0$	$rs-1$	$rs-2$
MMM (Nauty)	58.2	30.4	29.0	1203.0	30.8	5.6
MMM (Nauty, SEI)	206.5	42.6	42.7	1200.6	24.2	5.7
MMM (Nauty, sq.)	42.4	33.1	33.0	6.6	2.8	2.2
MMM (Nauty, SEI, sq.)	42.2	33.6	33.6	6.5	3.0	2.2
PLR graph (Nauty)	**29.7**	**18.5**	**18.5**	840.3	22.1	2.2
PLR graph (Nauty, SEI)	110.7	20.7	20.1	837.7	4.3	2.6
PLR graph (Nauty, sq.)	35.5	33.0	33.1	**5.3**	**2.2**	2.1
PLR graph (Nauty, SEI, sq.)	36.1	34.0	33.7	5.4	2.3	**2.1**

- PLR beats MMM method (to my surprise!).
- Massive difference between Latin squares and everything else.
- Square invariants were crucial for Latin squares.

Usefulness of invariants...



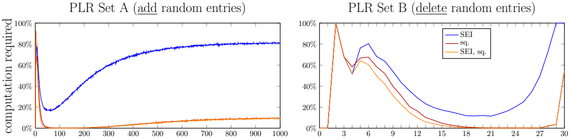
(a) $(r, s, n) = (5, 6, 7)$



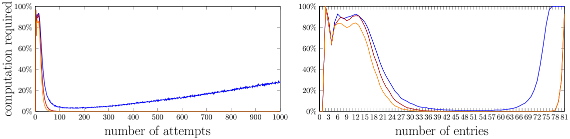
(b) $(r, s, n) = (9, 9, 9)$

Figure: Proportion of time computation is required (10000 samples).

Usefulness of invariants...



(a) $(r, s, n) = (5, 6, 7)$



(b) $(r, s, n) = (9, 9, 9)$

Figure: Proportion of time computation is required (10000 samples).

- Invariants often eliminate the need for computation with an intermediate number of entries.

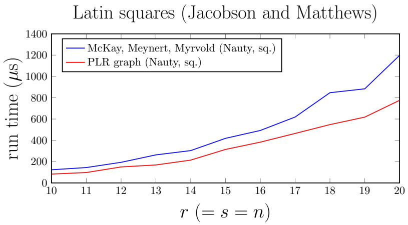


Figure: MMM method vs. PLR graph method, both using square entry invariants, for random Latin squares (10000 samples).

- I'm really surprised by this—the MMM method is the usual method.

Conclusion: Design goals...

To write a decent piece of code for computing autotopism groups of PLRs...

Conclusion: Design goals...

To write a decent piece of code for computing autotopism groups of PLRs...

- For very few entries, we have lots of symmetries, but we should be able to account for these mathematically.

Conclusion: Design goals...

To write a decent piece of code for computing autotopism groups of PLRs...

- For very few entries, we have lots of symmetries, but we should be able to account for these mathematically.
- For an intermediate number of entries, we can often eliminate computation using an entry invariant.

Conclusion: Design goals...

To write a decent piece of code for computing autotopism groups of PLRs...

- For very few entries, we have lots of symmetries, but we should be able to account for these mathematically.
- For an intermediate number of entries, we can often eliminate computation using an entry invariant.
- For a large number of entries, we may be best using the PLR graph method without invariants.

Conclusion: Design goals...

To write a decent piece of code for computing autotopism groups of PLRs...

- For very few entries, we have lots of symmetries, but we should be able to account for these mathematically.
- For an intermediate number of entries, we can often eliminate computation using an entry invariant.
- For a large number of entries, we may be best using the PLR graph method without invariants. (Maybe some theoretical work to reduce post-filtering?)

Conclusion: Design goals...

To write a decent piece of code for computing autotopism groups of PLRs...

- For very few entries, we have lots of symmetries, but we should be able to account for these mathematically.
- For an intermediate number of entries, we can often eliminate computation using an entry invariant.
- For a large number of entries, we may be best using the PLR graph method without invariants. (Maybe some theoretical work to reduce post-filtering?)
- For Latin squares, we're in another world...

Conclusion: Design goals...

To write a decent piece of code for computing autotopism groups of PLRs...

- For very few entries, we have lots of symmetries, but we should be able to account for these mathematically.
- For an intermediate number of entries, we can often eliminate computation using an entry invariant.
- For a large number of entries, we may be best using the PLR graph method without invariants. (Maybe some theoretical work to reduce post-filtering?)
- For Latin squares, we're in another world...
  - Some Latin squares have transitive autotopism groups (invariants are useless!).

Conclusion: Design goals...

To write a decent piece of code for computing autotopism groups of PLRs...

- For very few entries, we have lots of symmetries, but we should be able to account for these mathematically.
- For an intermediate number of entries, we can often eliminate computation using an entry invariant.
- For a large number of entries, we may be best using the PLR graph method without invariants. (Maybe some theoretical work to reduce post-filtering?)
- For Latin squares, we're in another world...
  - Some Latin squares have transitive autotopism groups (invariants are useless!).
  - Some Latin squares have large autotopism groups—computing this will be slow, even with an oracle.

Conclusion: Design goals...

To write a decent piece of code for computing autotopism groups of PLRs...

- For very few entries, we have lots of symmetries, but we should be able to account for these mathematically.
- For an intermediate number of entries, we can often eliminate computation using an entry invariant.
- For a large number of entries, we may be best using the PLR graph method without invariants. (Maybe some theoretical work to reduce post-filtering?)
- For Latin squares, we're in another world...
  - Some Latin squares have transitive autotopism groups (invariants are useless!).
  - Some Latin squares have large autotopism groups—computing this will be slow, even with an oracle. (Can we recognize these?)

Conclusion: Design goals...

To write a decent piece of code for computing autotopism groups of PLRs...

- For very few entries, we have lots of symmetries, but we should be able to account for these mathematically.
- For an intermediate number of entries, we can often eliminate computation using an entry invariant.
- For a large number of entries, we may be best using the PLR graph method without invariants. (Maybe some theoretical work to reduce post-filtering?)
- For Latin squares, we're in another world...
  - Some Latin squares have transitive autotopism groups (invariants are useless!).
  - Some Latin squares have large autotopism groups—computing this will be slow, even with an oracle. (Can we recognize these?)

It may be worthwhile re-implementing **Nauty**'s individualization-refinement method for this purpose.

Thank You!