Retransmission Permutation Arrays

Jeff Dinitz
University of Vermont

This talk is based on joint work with Doug Stinson, Maura Paterson and Ruizhong Wei.

Definition

A type 1 retransmission permutation array of order n (denoted type-1 RPA(n)) is an $n \times n$ array, say A, in which each cell contains a symbol from the set $\{1, \ldots, n\}$, such that the following properties are satisfied:

(i) every row of A contains all n symbols, and

(ii) for $1 \leq i \leq n$, the $i \times \lceil n/i \rceil$ rectangle in the upper left hand corner of A contains all n symbols.

An Example

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

$n = 10$

Lets check:

row latin ?

An Example

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

$n = 10$

Lets check:

row latin ?

An Example

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

$n = 10$

Lets check:

row latin ?

An Example

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

$n = 10$

Lets check:

row latin ?

An Example

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

$n = 10$

Lets check:

row latin ?

An Example

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

$n = 10$

Lets check:

row latin ?

An Example

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

$n = 10$

Lets check:

row latin ?

An Example

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

$n = 10$

Lets check:

row latin ?

An Example

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

$n = 10$

Lets check:

row latin ?

An Example

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

$n = 10$

Lets check:

row latin ?

An Example

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

$n = 10$

Lets check:

row latin ?

An Example

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

$n = 10$

Lets check:

row latin ?

An Example

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

$n = 10$

Lets check:

Left hand corner?

$i = 1$

1 x 10

An Example

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

$n = 10$

Lets check:

Left hand corner?

$i = 1$

1 x 10

An Example

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

$n = 10$

Lets check:

Left hand corner?

$i = 2$

2 x 5

An Example

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

$n = 10$

Lets check:

Left hand corner?

$i = 3$

3 x 4

An Example

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

$n = 10$

Lets check:

Left hand corner?

$i = 4$

4 x 3

An Example

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

$n = 10$

Lets check:

Left hand corner?

$i = 5$

5 x 2

An Example

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

$n = 10$

Lets check:

Left hand corner?

$i = 6$

6 x 2

An Example

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

$n = 10$

Lets check:

Left hand corner?

$i = 7$

7 x 2

An Example

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

$n = 10$

Lets check:

Left hand corner?

$i = 8$

8 x 2

An Example

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

$n = 10$

Lets check:

Left hand corner?

$i = 9$

9 x 2

An Example

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

$n = 10$

Lets check:

Left hand corner?

$i = 10$

10 x 1

An Example

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

$n = 10$

Lets check:

Left hand corner?

Motivation

Subject: design question
From: Michael Dinitz <mdinitz+@cs.cmu.edu>
Date: Sun, 21 Jun 2009 13:09:32 -0400
To: Dad <dinitz@cems.uvm.edu>

Hey Dad,

Here's that design question I mentioned. There are n symbols, which we want to place into an n x n box subject to the following:

1) Every row has every symbol (i.e. each row is a permutation)
2) For every integer k <= n, all of the symbols appear in the rectangle consisting of the first k columns and first \ceil{n/k} rows
3) For every integer k <= n, all of the symbols appear in the rectangle consisting of the last k columns and first \ceil{n/k} rows

Note the somewhat weird asymmetry in the problem: the first rows play a much more important role than the last rows. I only vaguely understand the motivation, but I think it's something like this. Some wireless systems are divided into "carriers", and every transmitter-receiver pair uses some consecutive n carriers to transmit. If someone else broadcasts on the exact same carriers then I'm screwed, but with reasonable probability I only have interference from one side of my interval, i.e. either my smallest r carriers are interfered with or my largest r carriers are interfered with. Now in the design, each row represents time, and k is the the number of carriers I use successfully. What I want is that no matter what k is, i.e. no matter who else chooses to broadcast as long as I get something through, it takes me the minimum number of rounds possible to transmit my message.

For example, suppose k is 1. Then I only get to use one carrier, either the smallest or the largest, and thus it takes me n rounds to get my entire message through. Requirement 2 of the design guarantees me that if I can use the smallest carrier it only takes me n rounds, and requirement 3 does the same for the largest carrier. If I can use the smallest k carriers, then I need to use at least \ceil{n/k} rounds, and requirements 2 and 3 guarantee that I achieve this. So in some sense this is an oblivious broadcasting algorithm: I don't know who's going to interfere with me, but whoever does I still get my message through as quickly as possible.

Have you seen anything like this before, or have any ideas? This version of the problem seemed to be something the networking guy was interested in, but the other theoreticians were more interested in variations of the problem that involved using coding schemes.

Hope you're having a great father's day!

Love,
Mike

Motivation

A type 1 retransmission permutation array of order *n* (denoted type-1 RPA(n)) is an $n \times n$ array, say *A*, in which each cell contains a symbol from the set $\{1, \ldots, n\}$, such that the following properties are satisfied:

(i) every row of A contains all n symbols, and

(ii) for $1 \leq i \leq n$, the $i \times \lceil n/i \rceil$ rectangle in the upper left hand corner of *A* contains all *n* symbols.

Motivation

A type 1 retransmission permutation array of order *n* (denoted type-1 RPA(n)) is an $n \times n$ array, say *A*, in which each cell contains a symbol from the set $\{1,\ldots,n\}$, such that the following properties are satisfied:

(i) every row of A contains all n symbols, and

(ii) for $1 \leq i \leq n$, the $i \times \lceil n/i \rceil$ rectangle in the upper left hand corner of *A* contains all *n* symbols.

- a type-2 array is one in which property (ii) instead holds for rectangles in the upper right corner of A.

Motivation

A type 1 retransmission permutation array of order *n* (denoted type-1 RPA(n)) is an $n \times n$ array, say *A*, in which each cell contains a symbol from the set $\{1,\ldots,n\}$, such that the following properties are satisfied:

(i) every row of A contains all n symbols, and

(ii) for $1 \leq i \leq n$, the $i \times \lceil n/i \rceil$ rectangle in the upper left hand corner of *A* contains all *n* symbols.

- a type-2 array is one in which property (ii) instead holds for rectangles in the upper right corner of A.
- a type-3 array is one in which property (ii) instead holds for rectangles in the lower left corner of A.
- a type-4 array is one in which property (ii) instead holds for rectangles in the lower right corner of A.

Motivation

A type 1 retransmission permutation array of order *n* (denoted type-1 RPA(n)) is an $n \times n$ array, say *A*, in which each cell contains a symbol from the set $\{1,\ldots,n\}$, such that the following properties are satisfied:

(i) every row of A contains all n symbols, and

(ii) for $1 \leq i \leq n$, the $i \times \lceil n/i \rceil$ rectangle in the upper left hand corner of *A* contains all *n* symbols.

- a type-2 array is one in which property (ii) instead holds for rectangles in the upper right corner of A.
- a type-3 array is one in which property (ii) instead holds for rectangles in the lower left corner of A.
- a type-4 array is one in which property (ii) instead holds for rectangles in the lower right corner of A.

So Mike is asking for a type-1,2 RPA

Latin RPA's

A retransmission permutation array A of order *n* is latin if every column of A contains all n symbols (column latin).

We denote a latin RPA of order *n* as an LRPA(*n*)

Another Example

A type-1,2,3,4 LRPA(4):

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

- An $r \times \lceil n/r \rceil$ rectangle is called basic if it does not contain an $r' \times \lceil n/r' \rceil$ rectangle where $r' < r$ and $\lceil n/r \rceil = \lceil n/r' \rceil$.

- In verifying property (ii), it suffices to consider only basic rectangles. The basic rectangles that must be verified in the above example have dimensions 1 × 4, 2 × 2 and 4 × 1.

Another Example

A type-1,2,3,4 LRPA(4):

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

- An $r \times \lceil n/r \rceil$ rectangle is called basic if it does not contain an $r' \times \lceil n/r' \rceil$ rectangle where $r' < r$ and $\lceil n/r \rceil = \lceil n/r' \rceil$.

- In verifying property (ii), it suffices to consider only basic rectangles. The basic rectangles that must be verified in the above example have dimensions 1 × 4, 2 × 2 and 4 × 1.

Another Example

A type-1,2,3,4 LRPA(4):

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

- An $r \times \lceil n/r \rceil$ rectangle is called basic if it does not contain an $r' \times \lceil n/r' \rceil$ rectangle where $r' < r$ and $\lceil n/r \rceil = \lceil n/r' \rceil$.

- In verifying property (ii), it suffices to consider only basic rectangles. The basic rectangles that must be verified in the above example have dimensions 1 × 4, 2 × 2 and 4 × 1.

Another Example

A type-1,2,3,4 LRPA(4):

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

- An $r \times \lceil n/r \rceil$ rectangle is called basic if it does not contain an $r' \times \lceil n/r' \rceil$ rectangle where $r' < r$ and $\lceil n/r \rceil = \lceil n/r' \rceil$.

- In verifying property (ii), it suffices to consider only basic rectangles. The basic rectangles that must be verified in the above example have dimensions 1 × 4, 2 × 2 and 4 × 1.

Another Example

A type-1,2,3,4 LRPA(4):

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

- An $r \times \lceil n/r \rceil$ rectangle is called basic if it does not contain an $r' \times \lceil n/r' \rceil$ rectangle where $r' < r$ and $\lceil n/r \rceil = \lceil n/r' \rceil$.

- In verifying property (ii), it suffices to consider only basic rectangles. The basic rectangles that must be verified in the above example have dimensions 1 × 4, 2 × 2 and 4 × 1.

Another Example

A type-1,2,3,4 LRPA(4):

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

- An $r \times \lceil n/r \rceil$ rectangle is called basic if it does not contain an $r' \times \lceil n/r' \rceil$ rectangle where $r' < r$ and $\lceil n/r \rceil = \lceil n/r' \rceil$.

- In verifying property (ii), it suffices to consider only basic rectangles. The basic rectangles that must be verified in the above example have dimensions 1 × 4, 2 × 2 and 4 × 1.

Another Example

A type-1,2,3,4 LRPA(4):

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

- An $r \times \lceil n/r \rceil$ rectangle is called basic if it does not contain an $r' \times \lceil n/r' \rceil$ rectangle where $r' < r$ and $\lceil n/r \rceil = \lceil n/r' \rceil$.

- In verifying property (ii), it suffices to consider only basic rectangles. The basic rectangles that must be verified in the above example have dimensions 1 × 4, 2 × 2 and 4 × 1.

Another Example

A type-1,2,3,4 LRPA(4):

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

- An $r \times \lceil n/r \rceil$ rectangle is called basic if it does not contain an $r' \times \lceil n/r' \rceil$ rectangle where $r' < r$ and $\lceil n/r \rceil = \lceil n/r' \rceil$.

- In verifying property (ii), it suffices to consider only basic rectangles. The basic rectangles that must be verified in the above example have dimensions 1 × 4, 2 × 2 and 4 × 1.

Another Example

A type-1,2,3,4 LRPA(4):

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

- An $r \times \lceil n/r \rceil$ rectangle is called basic if it does not contain an $r' \times \lceil n/r' \rceil$ rectangle where $r' < r$ and $\lceil n/r \rceil = \lceil n/r' \rceil$.

- In verifying property (ii), it suffices to consider only basic rectangles. The basic rectangles that must be verified in the above example have dimensions 1 × 4, 2 × 2 and 4 × 1.

Another Example

A type-1,2,3,4 LRPA(4):

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

- An $r \times \lceil n/r \rceil$ rectangle is called basic if it does not contain an $r' \times \lceil n/r' \rceil$ rectangle where $r' < r$ and $\lceil n/r \rceil = \lceil n/r' \rceil$.

- In verifying property (ii), it suffices to consider only basic rectangles. The basic rectangles that must be verified in the above example have dimensions 1 × 4, 2 × 2 and 4 × 1.

One More Example

A type-1,2,3,4 LRPA(8):

1	2	3	4	5	6	7	8
5	6	7	8	1	2	3	4
8	4	6	2	7	3	5	1
7	3	5	1	8	4	6	2
2	1	4	3	6	5	8	7
6	5	8	7	2	1	4	3
4	8	2	6	3	7	1	5
3	7	1	5	4	8	2	6

This array satisfies two symmetry properties:

- $a_{i,j} + a_{i,n+1-j} = n + 1$ where $n = 8$.
- $a_{j,i} = \pi(a_{i,j})$ where $\pi = (1)(2\ 5)(3\ 8)(4\ 7)(6)$.

Mike's motivation

Li, Liu, Tan, Viswanathan, and Yang published a paper entitled *Retransmission ≠ repeat: simple retransmission permutation can resolve overlapping channel collisions* (Eighth ACM Workshop on Hot Topics in Networks, 2009) in which they utilize type -1, 2 RPA(n) to resolve overlapping channel collisions.

Retransmission ≠ Repeat:
Simple Retransmission Permutation Can Resolve Overlapping Channel Collisions

Li Erran Li* Junliang Liu† Kun Tan†
Harish Viswanathan* Yang Richard Yang×

ABSTRACT

Collisions in overlapping channels are becoming an increasingly important problem in the deployment of high-speed wireless networks. In this paper, we present Remap, a simple, novel paradigm for handling collisions in overlapping OFDM channels. Remap introduces the novel concept of retransmission permutation that permutes the bit-to-subcarrier assignment after each retransmission, departing from the traditional, simply-repeat paradigm. Remap is simple to implement and able to exploit collision-free subcarriers to decode packets despite successive collisions in overlapping channels. We apply Remap to 802.11g to demonstrate that the diversity created by remapped packets can substantially improve decoding efficiency and improve wireless throughput. We implement our technique in software radio and demonstrate that it has potential to be deployed with simple software and firmware updates.

1. INTRODUCTION

As OFDM becomes the foundation of modern high-speed wireless networks due to its advantages such as lower symbol rate, effective usage of a large frequency band, and resistance to frequency-selective fading, collisions in overlapping OFDM channels become an increasingly important problem in the deployment of high-speed wireless networks. Specifically, in an OFDM network, each channel is allocated a set of subcarriers, and two channels overlap when the intersection of their sets of subcarriers is not empty. Consider 802.11g, which is becoming almost ubiquitously deployed in many residential neighborhood. With only 3 orthogonal channels with disjoint sets of subcarriers but a large number of access points in densely populated neighborhood, it is inevitable that many 802.11 access points in range of each other use overlapping channels, as observed by previous measurement studies (e.g., [1]). In [15], the authors show that partially overlapping channels may improve network throughput even in managed 802.11 networks, when the number of orthogonal channels is limited. Channel overlapping is also allowed in WiFi networks built on digital white spaces [2]. Although one may try to alleviate the shortage of orthogonal channels by using variable bandwidth channels, as advocated in [5, 9], bursty or time varying workload can pose a problem for channel width adaptation.

However, collisions in overlapping OFDM channels during contention and/or in the presence of hidden terminals are distinct from collisions in a single-channel setting. Consequently, recent progress (e.g., Zigzag Decoding [6]) on handling single-channel collisions does not directly apply.

In this paper, we present Remap, a simple, novel paradigm for handling collisions in OFDM networks with overlapping channels. Remap is different from the existing, passively repeat paradigm, and introduces a novel concept called retransmission permutation to permute the bit-to-subcarrier mapping after each retransmission. Retransmission permutation is a powerful diversity technique [17] that can recover frequency selective losses from subsequent retransmissions when there is no collision. When there are collisions, it in essence provides channel-width adaptation and allows bootstrapping of the decoding of collided packets that may otherwise be impossible to decode.

Specifically, the foundation of Remap is based on a simple observation and a simple idea. The observation is that when two packets transmitted in overlapping channels collide, only the subcarriers in the intersection of the two channels collide; the bits in other subcarriers are clean and can be collected. However, the non-colliding subcarriers do not contain complete packet information. The idea of Remap is to introduce structured permutation on the mapping from bits to subcarriers after each collision to create structured diversity. This diversity allows either independent decoding or bootstrapping other decoding techniques such as Zigzag decoding [6]. Integrating Remap with an existing system requires small changes to the OFDM physical layer as it involves only bit-to-subcarrier remapping.

In particular, we design 802.11g/Remap, which applies Remap to 802.11g to demonstrate its effectiveness. We show that by using the diversity created by remapped packets, an 802.11g receiver a can decode any packet P_a after 4 collisions with other transmissions in *adjacent* channels; the number of collisions reduces to 2 if the other transmissions are in *non-adjacent* channels. These numbers do not make any assumptions on the packets collided with P_a. If the packets collided with P_a are the same (i.e., both collisions are between P_a and another packet P_b), Remap can bootstrap decoding both packets with the bits on the collision-free subcarriers. Decoding both packets at a single AP is important so that the combiner (used in systems such as [16]) behind the AP can make use of the reception diversity. In contrast, without Remap, Zigzag cannot decode both packets. Furthermore, 802.11g/Remap is backward compatible with

*Bell Labs; †Microsoft Research Asia; ×Yale.

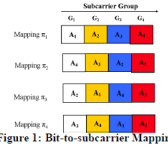


Figure 1: Bit-to-subcarrier Mapping.

802.11 MAC. Thus, 802.11g/Remap has potential to be deployed with simple software and firmware updates to the existing 802.11 networks.

We have implemented 802.11g/Remap using a software radio testbed. Our initial experimental results show that Remap works. The BER on decoding collision-free subcarriers is close to that of decoding collision-free transmissions, when the interference signal strength is smaller than the desired signal by at least 5 dB.

2. BASIC IDEA

802.11g Primer

We use 802.11g to illustrate our basic idea. In standard 802.11g, data bits are assigned to subcarriers. We denote the first group of 16 subcarrier frequencies of 802.11g as G_1, the next group as G_2, etc. Each 802.11g channel consists of 64 consecutive subcarrier frequencies [1]. Thus, the first channel C_1 of 802.11g consists of four groups: G_1, G_2, G_3, and G_4; channel C_2 consists of G_2, G_3, G_4, and G_5; channel C_3 consists of G_3, G_4, G_5, and G_6; channel C_4 consists of G_4, G_5, G_6, and G_7, etc. Note that C_1 overlaps with C_1, C_2, C_3, and C_4. We say that C_2 is an *adjacent channel* of C_1; C_3 and C_4 are *overlapping non-adjacent channels* of C_1 [2].

Assume that a sender uses channel C_1 to send a packet P consisting of data bits A_1, A_2, A_3, and A_4. Let the bit-to-subcarrier assignment be that $A_1 \rightarrow G_1, A_2 \rightarrow G_2, A_3 \rightarrow G_3$, and $A_4 \rightarrow G_4$. In other words, the bits A_1 are assigned to be carried by subcarrier group G_1, A_2 by G_2, A_3 by G_3, and A_4 by G_4. If the transmission for packet P is not successful, 802.11g retransmits the packet P where the bit-to-subcarrier assignment is the same.

Retransmission Permutation

The key novelty introduced by Remap is that during a retransmission, Remap uses a permutation scheduling of bit-to-subcarrier mapping.

Figure 1 is a schedule of bit-to-subcarrier mapping for 802.11g. As shown in the figure, for the original transmission, bits $A_1, \cdots, A_4$ are mapped to subcarrier groups $G_1, \cdots, G_4$ respectively. For the first retransmission, bits A_4, A_3, A_2, A_1 are mapped to subcarrier group $G_1, \cdots, G_4$ respectively. The third and fourth rows are for the second and third retransmissions. We cycle through these four mappings for more

[1] Only 48 subcarriers carry data bits

[2] Note that non-overlapping channels do not equal to orthogonal channels; due to imperfect filtering, guard bands are needed to achieve orthogonality. Our decoding technique is subject to adjacent channel interference.

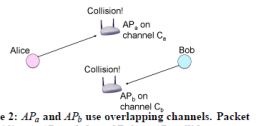


Figure 2: AP_a and AP_b use overlapping channels. Packet from Alice to AP_a and that of Bob to AP_b collide.

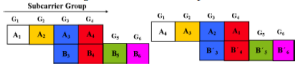


Figure 3: Subcarrier view of collisions: non-adjacent channels.

retransmissions. We only need 2 bits to encode the four mapping schemes. We can use the reserved bits in the SERVICE field of the PLCP header.

With the basic idea, now we demonstrate the benefits of Remap using a simple example. Consider two residential users, Alice and Bob, who use 802.11g to connect to access points AP_a and AP_b respectively. Let the channel between Alice and AP_a be C_a, and that between Bob and AP_b be C_b. Assume that the two channels are overlapping channels. Figure 2 shows the setting.

Due to hidden terminals or randomness, Alice may transmit a packet P_a to AP_a concurrently with Bob transmitting a packet P_b to AP_b, causing collisions at AP_a and AP_b. In absence of receiving an acknowledgment, Alice retransmits P_a, which may again collide with a transmission of Bob for packet P'_b. Note that P'_b may be different from P_b due to rescheduling. Without Remap, packet P_a cannot be decoded by the access point AP_a so long there are collisions.

Remap, however, allows decoding of collided packets. To illustrate our idea, we consider how AP_a decodes P_a. We will show that Remap allows AP_a to decode P_a after *at most* 3 retransmissions. We illustrate three cases.

Collisions in Non-adjacent Channels: Figure 3 considers the first case, when Alice and Bob use overlapping but non-adjacent channels (e.g., Alice uses C_1 and Bob uses C_3). The figure shows the collided packets in the frequency domain. We can check that AP_a can decode packet P_a after Alice retransmits the packet only once.

Collisions in Adjacent Channels: The worst-case of 3 retransmissions happens when Alice and Bob use adjacent channels (e.g., Alice uses C_1 and Bob uses C_2). Figure 4 shows the first transmission and one retransmission. After the two transmissions, bit blocks A_1 and A_4 will be recovered. Two

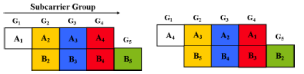


Figure 4: Subcarrier view of collisions: adjacent channels.

Mike's motivation

Li, Liu, Tan, Viswanathan, and Yang published a paper entitled *Retransmission ≠ repeat: simple retransmission permutation can resolve overlapping channel collisions* (Eighth ACM Workshop on Hot Topics in Networks, 2009) in which they utilize type -1, 2 RPA(n) to resolve overlapping channel collisions.

Suppose a message is divided into n pieces and broadcast using n consecutive groups (i.e., sets of carrier frequencies).

g_1	g_2	g_3	g_4
1	2	3	4

Mike's motivation

Li, Liu, Tan, Viswanathan, and Yang published a paper entitled *Retransmission ≠ repeat: simple retransmission permutation can resolve overlapping channel collisions* (Eighth ACM Workshop on Hot Topics in Networks, 2009) in which they utilize type -1, 2 RPA(n) to resolve overlapping channel collisions.

g_1	g_2	g_3	g_4	g_5	g_6
1	2	x_1	x_2	x_3	x_4

A second channels may overlap in an arbitrary number $j \leq n$ of groups.

(here $j = 2$ and pieces 3 and 4 of the message are lost)

Mike's motivation

Li, Liu, Tan, Viswanathan, and Yang published a paper entitled *Retransmission ≠ repeat: simple retransmission permutation can resolve overlapping channel collisions* (Eighth ACM Workshop on Hot Topics in Networks, 2009) in which they utilize type -1, 2 RPA(n) to resolve overlapping channel collisions.

We can now broadcast the second row in the RPA(4) to send the complete message (even if g_3 and g_4 are occupied)

g_1	g_2	g_3	g_4	g_5	g_6
1	2	x_1	x_2	x_3	x_4
4	3	x_1	x_2	x_3	x_4

Mike's motivation

Li, Liu, Tan, Viswanathan, and Yang published a paper entitled *Retransmission ≠ repeat: simple retransmission permutation can resolve overlapping channel collisions* (Eighth ACM Workshop on Hot Topics in Networks, 2009) in which they utilize type -1, 2 RPA(n) to resolve overlapping channel collisions.

A type-1, 2 RPA(n) gives a schedule for rebroadcasting messages in n "rounds" in such a way that all n pieces of a message are received in the minimum number of rounds, regardless of the overlap value, j.

g_1	g_2	g_3	g_4	g_5	g_6
1	2	x_1	x_2	x_3	x_4
4	3	x_1	x_2	x_3	x_4

Some related work

- A sudoku square is a type-1,2,3,4 LRPA(9)

  R.A. Bailey, P. Cameron and R. Connelly, Sudoku, gerechte designs, resolutions, affine space, spreads, reguli, and Hamming codes, *American Mathematical Monthly*, Volume115, Number 5, May 2008, pp 383–404.

Some related work

- A sudoku square is a type-1,2,3,4 LRPA(9)

  R.A. Bailey, P. Cameron and R. Connelly, Sudoku, gerechte designs, resolutions, affine space, spreads, reguli, and Hamming codes, *American Mathematical Monthly*, Volume115, Number 5, May 2008, pp 383–404.

- A gerechte design is a latin square of order n, where the cells are divided into *n* regions, each containing *n* cells, such that each symbol occurs once in each region.

  J. Courtiel, E. R. Vaughan, Gerechte Designs with Rectangular Regions arXiv:1104.0637v1, 2011.

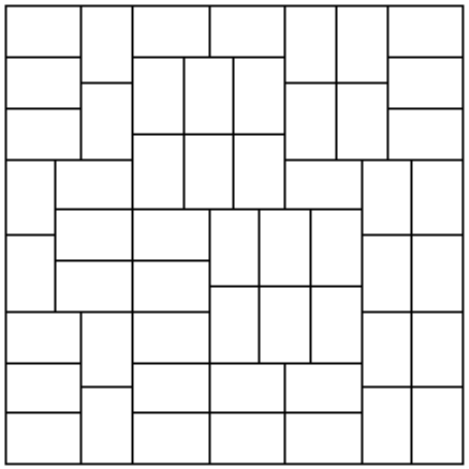


FIGURE 1. A gerechte framework of order 54 with 6×9 and 9×6 rectangular regions. (Individual cells are not shown.)

The authors prove for all positive integers s and t, any gerechte framework where each region is either an $s \times t$ rectangle or a $t \times s$ rectangle is realizable.

Some related work

C.J. Colbourn and K.E. Heinrich. Conflict-free access to parallel memories, *Journal of Parallel and DistributedComputing* 14 (1992), 193–200.

In this paper (and other related papers), fixed sized, arbitrarily positioned rectangles in a latin square are required to contain each symbol at most once.

Our main results

Type of array	Existence result
type-1 RPA(n)	all integers $n \geq 1$
type-1,2 RPA(n)	all integers $n \geq 1$

This is Mike's problem

Our main results

Type of array	Existence result
type-1 RPA(n)	all integers $n \geq 1$
type-1,2 RPA(n)	all integers $n \geq 1$
type-1,3 RPA(n)	all integers $n \geq 1$
type-1,4 RPA(n)	all integers $n \geq 1$
type-1,2,3,4 RPA(n)	all even integers $n \geq 1$
type-1,2,3,4 latin RPA(n)	even integers $n \leq 16, n = 36$
type-1,2,3,4 latin RPA(n)	odd integers $n \leq 9$

Constructing a type-1 RPA(10)

Suppose $n = 10$. The basic rectangles have dimensions 1×10, 2×5, 3×4, 4×3, 5×2 and 10×1.

We begin by filling in the 1×10 basic rectangle:

1	2	3	4	5	6	7	8	9	10

Constructing a type-1 RPA(10)

Suppose $n = 10$. The basic rectangles have dimensions 1×10, 2×5, 3×4, 4×3, 5×2 and 10×1.

We begin by filling in the 1×10 basic rectangle:

1	2	3	4	5	6	7	8	9	10

Next, we consider the 2×5 basic rectangle. We place the symbols 6, 7, 8, 9, 10 in the first five cells of the second row of this rectangle:

1	2	3	4	5	6	7	8	9	10

Constructing a type-1 RPA(10)

Suppose $n = 10$. The basic rectangles have dimensions 1×10, 2×5, 3×4, 4×3, 5×2 and 10×1.

We begin by filling in the 1×10 basic rectangle:

1	2	3	4	5	6	7	8	9	10

Next, we consider the 2×5 basic rectangle. We place the symbols 6, 7, 8, 9, 10 in the first five cells of the second row of this rectangle:

1	2	3	4	5	6	7	8	9	10

Constructing a type-1 RPA(10)

Next we deal with the 3 x 4 basic rectangle:

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

We need the symbols 5 and 10 in that rectangle

Constructing a type-1 RPA(10)

Next we deal with the 3 x 4 basic rectangle:

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

We move the symbols 5 and 10 to the first two cells in the third row.

Constructing a type-1 RPA(10)

Next is the 4 x 3 basic rectangle:

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

We move the symbols 4 and 9 to the first two cells in the fourth row.

Constructing a type-1 RPA(10)

Next is the 5 x 2 basic rectangle:

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

We move the symbols 3 and 8 to the first two cells in the fourth row.

Constructing a type-1 RPA(10)

Finally we get to the 1 x 10 basic rectangle:

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

move 2, 7,10, 9 and 8 to the last rows of the first column

Constructing a type-1 RPA(10)

The partial latin square below satisfies condition 2 of the definition of a type-1 RPA

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

Constructing a type-1 RPA(10)

We merely need to fill the remaining cells so that each row contains all the symbols 1—10.

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

Constructing a type-1 RPA(10)

We merely need to fill the remaining cells so that each row contains all the symbols 1—10.

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

It is easy to show that the above process works for all $n \geq 1$ since:

- Each basic rectangle contains each symbol at least once, and
- Each row in a basic rectangle contains no symbol twice.

Hence we get our first theorem.

Theorem:

For all integers $n \geq 1$, there exists a type-1 RPA(n).

Some extensions to other types

Theorem: If there exists a type-1 RPA(n), then there exists a type-1,3 RPA(n). (So these exist for all n)

Theorem: If there exists a type-1 RPA(n), then there exists a type-1,4 RPA(n). (So these exist for all n)

Some extensions to other types

Theorem: If there exists a type-1 RPA(n), then there exists a type-1,3 RPA(n). (So these exist for all n)

Theorem: If there exists a type-1 RPA(n), then there exists a type-1,4 RPA(n). (So these exist for all n)

Theorem: If n is even and there exists a type-1,2 RPA(n), then there exists a type-1,2,3,4 RPA(n).

Type-1, 2 RPA(n)

Suppose n is even.

We'll construct arrays $A = (a_{i,j})$ where for all $1 \leq i, j \leq n$ it holds that $a_{i,j} + a_{i,n+1-j} = n + 1$

Type-1, 2 RPA(n)

Suppose n is even.

We'll construct arrays $A = (a_{i,j})$ where for all $1 \leq i, j \leq n$ it holds that $a_{i,j} + a_{i,n+1-j} = n + 1$

Suppose we construct a type-1 RPA(n), ensuring that after the basic rectangles have been filled in, no row contains two symbols that sum to $n + 1$ (except for the first row, which is already a permutation of the n symbols).

Then we can easily fill in the rest of A to construct a type-1, 2 RPA(n):

1. For every filled cell (i, j) , we define $a_{i,n+1-j} = n + 1 - a_{i,j}$
2. At this point, no row contains any symbol more than once, so it is then a simple matter to complete each row to a permutation of the n symbols.

As an easy example we'll make a

type-1,2 RPA(8)

A type 1,2 RPA(8)

Suppose $n = 8$. The basic rectangles have dimensions

$1 \times 8,\ 2 \times 4,\ 3 \times 3,\ 4 \times 2,$ and 8×1

We begin by filling in the 1 × 8 basic rectangle:

1	2	3	4	5	6	7	8

A type 1,2 RPA(8)

Suppose $n = 8$. The basic rectangles have dimensions

$1 \times 8,\ 2 \times 4,\ 3 \times 3,\ 4 \times 2,$ and 8×1

We begin by filling in the 1 × 8 basic rectangle:

1	2	3	4	5	6	7	8

Next, we consider the 2×4 basic rectangle. We place the symbols 5, 6, 7, 8 in the first four cells of the second row of this rectangle, note that no two of these symbols sum to 9:

1	2	3	4	5	6	7	8
5	6	7	8				

A type 1,2 RPA(8)

Now we consider the 3×3 basic rectangle. We fill in the first two cell of the third row with the symbols 4 and 8 (note $4 + 8 \neq 9$):

1	2	3	4	5	6	7	8
5	6	7	8				
4	8						

A type 1,2 RPA(8)

Now we consider the 3×3 basic rectangle. We fill in the first two cell of the third row with the symbols 4 and 8 (note $4 + 8 \neq 9$):

1	2	3	4	5	6	7	8
5	6	7	8				
4	8						

Next look at the 4 × 2 basic rectangle. We have to fill in the symbols 3 and 7 (note that $3 + 7 \neq 9$):

1	2	3	4	5	6	7	8
5	6	7	8				
4	8						
3	7						

A type 1,2 RPA(8)

The last basic rectangle has size 8×1. It is completed by filling in the symbols 2,6,8,7 into the first cells of the last four rows

1	2	3	4	5	6	7	8
5	6	7	8				
4	8						
3	7						
2							
6							
8							
7							

A type 1,2 RPA(8)

Now, we “reflect” each row (using the bijection $x \longmapsto 9 - x$)

1	2	3	4	5	6	7	8
5	6	7	8	1	2	3	4
4	8					1	5
3	7					2	6
2							7
6							3
8							1
7							2

A type 1,2 RPA(8)

Finally, we fill in the remaining cells in each row so that each row is a permutation.

1	2	3	4	5	6	7	8
5	6	7	8	1	2	3	4
4	8	2	3	6	7	1	5
3	7	1	4	5	8	2	6
2	1	3	4	5	6	8	7
6	1	3	4	5	7	8	3
8	2	3	4	5	6	7	1
7	1	3	4	5	6	8	2

This may look easy, but it can't be done blindly without problems occurring. For example, here are the basic rectangles in the RPA(10) we constructed earlier.

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

If we reflect as we did for the RPA(8) (now using the bijection $x \longmapsto 11 - x$), we get

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

If we reflect as we did for the RPA(8) (now using the bijection $x \longmapsto 11 - x$), we get

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

Note that in row 5 both symbols 3 and 8 occur twice so it will not be possible to make this row a permutation ☹

Constructing a type1,2-RPA(n)

We fill in the cells of the square one basic rectangle at a time.

We will work to avoid the problem that we just had in the $n = 10$ case.

Once filled, basic rectangles will satisfy three properties:

Constructing a type1,2-RPA(n)

Property 1:

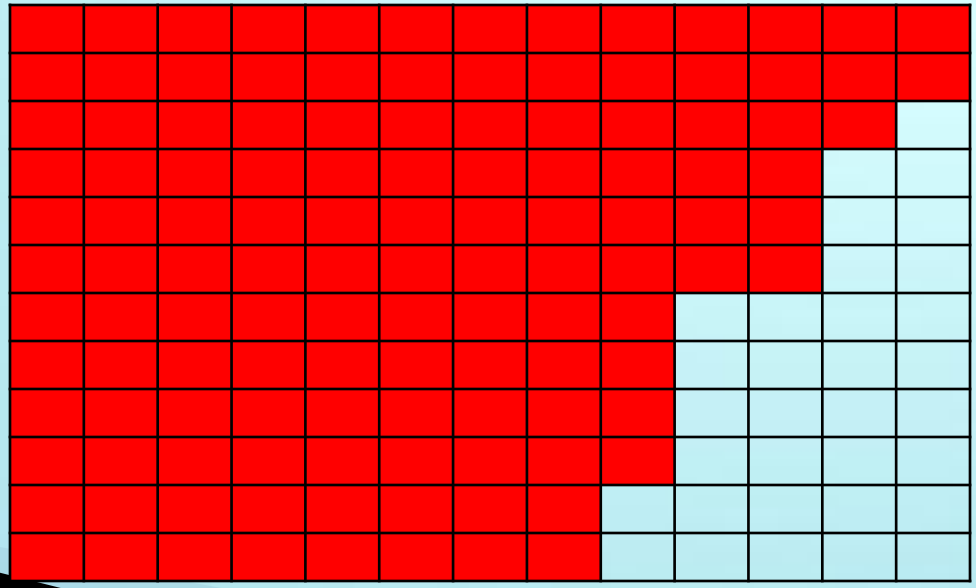
In a basic rectangle the number of filled cells per row is nonincreasing.

Constructing a type1,2-RPA(n)

Property 1:

In a basic rectangle the number of filled cells per row is nonincreasing.

So a filled basic rectangle may look like:



Constructing a type1,2-RPA(n)

Property 2:

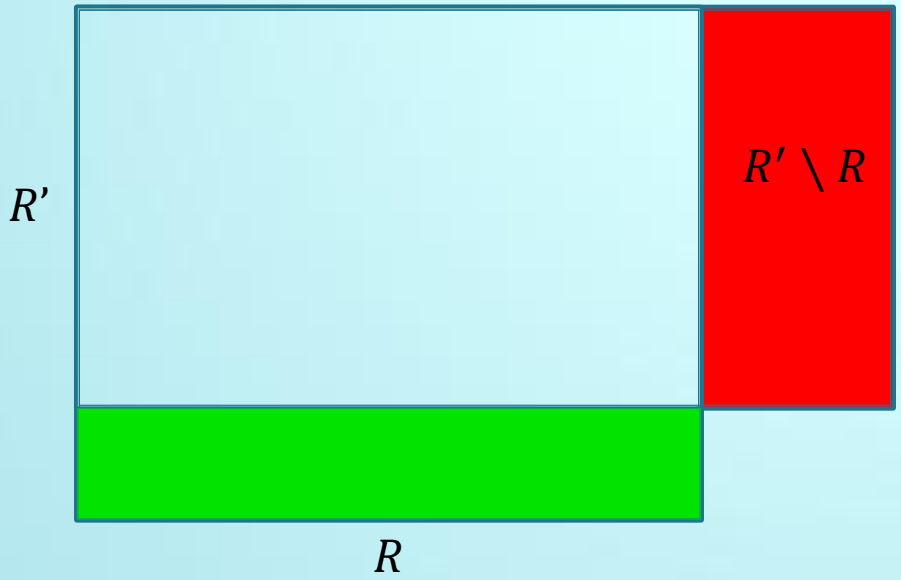
For any two symbols a, b in the same row of a basic rectangle R,

$$a + b \neq n + 1$$

Constructing a type1,2-RPA(n)

Property 3:

If R and R' are is a basic rectangles with R' following R when the basic rectangles are ordered by number of rows, then if a, b are two symbols in $R' \setminus R$, then $a + b \neq n + 1$



Constructing a type1,2-RPA(n)

The algorithm:

Step 1: Fill in the first basic rectangle, having size $1 \times n$ from left to right with $1, 2, \ldots, n$

Step 2 through b: (b = the number of basic rectangles)

For each k, $2 \leq k \leq b$ do the following:

- Let the k^{th} basic rectangle be denoted R and the previous one be R'
- Let S denote the set of symbols in $R' \setminus R$.
- Copy the symbols in S into the empty cells in R in such a way that properties 1 and 2 are satisfied.

Constructing a type1,2-RPA(n)

Copy the symbols in S into the empty cells in R in such a way that properties 1 and 2 are satisfied.

Constructing a type1,2-RPA(n)

- Copy the symbols in S into the empty cells in R in such a way that properties 1 and 2 are satisfied.

This is easy in practice, but much more complicated to prove in general. We need to ensure that we never place a symbol y from a red cell into a row that already contains the symbol $x = n + 1 - y$.

We use the following easy to read lemma:

Lemma 4.6. *Suppose $w > b_L \geq \cdots \geq b_1$ are positive integers and suppose d is a positive integer. Denote $b = \sum_{i=1}^{L} b_i$ and suppose that*

$$0 \leq t \leq (L+d)w - b. \tag{4}$$

Suppose $B_1, \ldots, B_L$ are pairwise disjoint sets such that $|B_i| = b_i$ for $1 \leq i \leq L$. Finally, suppose that $|B| = t$. Then there exists a partition

$$B = \left(\bigcup_{i=1}^{L} C_i\right) \bigcup \left(\bigcup_{i=1}^{d} D_i\right),$$

where the following properties are satisfied:

1. $w \geq b_L + |C_L| \geq b_{L-1} + |C_{L-1}| \geq \cdots \geq b_1 + |C_1| \geq |D_1| \geq \cdots \geq |D_d|$.
2. $C_i \cap B_i = \emptyset$ *for* $1 \leq i \leq L$.

Constructing a type1,2-RPA(n)

The algorithm:

Step 1: Fill in the first basic rectangle, having size $1 \times n$ from left to right with $1, 2, \ldots, n$

Step 2 through b: (b= the number of basic rectangles)

For each k, $2 \leq k \leq b$ do the following:

- Let the k^{th} basic rectangle be denoted R and the previous one be R'
- Let S denote the set of symbols in $R' \setminus R$.
- Copy the symbols in S into the empty cells in R in such a way that Properties 1 and 2 are satisfied.
- Perform a sequence of symbol exchanges within the rows of R so that Property 3 is satisfied.

That this is always possible can be proven using a certain "alternating path" graph-theoretic argument. (Details in the paper)

Constructing a type1,2-RPA(n)

The algorithm:

Step 1: Fill in the first basic rectangle, having size $1 \times n$ from left to right with $1, 2, \ldots, n$

Step 2 through b: (b= the number of basic rectangles)

For each k, $2 \leq k \leq b$ do the following:

- Let the k^{th} basic rectangle be denoted R and the previous one be R'
- Let S denote the set of symbols in $R' \setminus R$.
- Copy the symbols in S into the empty cells in R in such a way that Properties 1 and 2 are satisfied.
- Perform a sequence of symbol exchanges within each row of R so that Property 3 is satisfied.

Reflection step: For each nonempty cell (i, j) in rows $2, \ldots, n$ of A define $a_{n+1-j} = n + 1 - a_{i,j}$

Final Step: Fill in nonempty cells in every row to form a permutation

Constructing a type-1,2 RPA(10)

Suppose $n = 10$. The basic rectangles have dimensions 1×10, 2×5, 3×4, 4×3, 5×2 and 10×1.

We begin by filling in the 1×10 basic rectangle:

1	2	3	4	5	6	7	8	9	10

Constructing a type-1,2 RPA(10)

Suppose $n = 10$. The basic rectangles have dimensions 1×10, 2×5, 3×4, 4×3, 5×2 and 10×1.

We begin by filling in the 1×10 basic rectangle:

1	2	3	4	5	6	7	8	9	10

Next, we consider the 2×5 basic rectangle. We place the symbols 6, 7, 8, 9, 10 in the first five cells of the second row of this rectangle:

1	2	3	4	5	6	7	8	9	10

Constructing a type-1,2 RPA(10)

Suppose $n = 10$. The basic rectangles have dimensions 1×10, 2×5, 3×4, 4×3, 5×2 and 10×1.

We begin by filling in the 1×10 basic rectangle:

1	2	3	4	5	6	7	8	9	10

Next, we consider the 2×5 basic rectangle. We place the symbols 6, 7, 8, 9, 10 in the first five cells of the second row of this rectangle:

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

Constructing a type-1,2 RPA(10)

Next we deal with the 3 x 4 basic rectangle:

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

We need the symbols 5 and 10 in that rectangle

Constructing a type-1,2 RPA(10)

Next we deal with the 3 x 4 basic rectangle:

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

We move the symbols 5 and 10 to the first two cells in the third row.

(So far this is all the same as before)

Constructing a type-1,2 RPA(10)

Next is the 4 x 3 basic rectangle:

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

We move the symbols 4 and 9 to the first two cells in the fourth row.

Note that the filled cells in this basic rectangle satisfy Property 1.

Constructing a type-1,2 RPA(10)

Next is the 5 x 2 basic rectangle:

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

We now need to move the symbols 3,8 and 4 to the empty cells in the 5 x 2 rectangle. But $3 + 8 = 11$ so this violates Property 3.

Constructing a type-1,2 RPA(10)

Next is the 5 x 2 basic rectangle:

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

We now need to move the symbols 3,8 and 4 to the empty cells in the 5 x 2 rectangle. But $3 + 8 = 11$ so this violates Property 3. We do an exchange within row 1 to solve this problem.

Constructing a type-1,2 RPA(10)

Next is the 5 x 2 basic rectangle:

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

Since the exchange was in row 1 we also need to swap the 8 and the 10 to keep the reflection property.

Constructing a type-1,2 RPA(10)

Next is the 5 x 2 basic rectangle:

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

We move the symbols 1, 8, and 4 into this basic rectangle

Constructing a type-1,2 RPA(10)

The rest is straightforward:

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

first we complete the 1 x 10 basic rectangle

Constructing a type-1,2 RPA(10)

Then we “reflect”

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

Note that Property 2 ensures that this is a type-2 RPA

Constructing a type-1,2 RPA(10)

Finally we fill in the empty cells in each row

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

Constructing a type-1,2 RPA(n)

The technique described above can be modified to handle the case where n is odd.

We get our second theorem.

Theorem:

For all integers $n \geq 1$, there exists a type-1,2 RPA(n)

Constructing latin RPA

- Finding general constructions for *LRPA*s seems to be quite difficult (at least for us).
- We only have a few small examples at the present time (no infinite classes are known – even for type-1 LRPA(n)

Constructing latin RPA

- Finding general constructions for *LRPA*s seems to be quite difficult (at least for us).
- We only have a few small examples at the present time (no infinite classes are known – even for type-1 LRPA(n)
- We describe the method we used to construct type-1,2,3,4 LRPA(16) and type-1,2,3,4 LRPA(36), illustrating the technique by constructing a type-1,2,3,4 LRPA(16).

Constructing latin RPA

- Finding general constructions for *LRPA*s seems to be quite difficult (at least for us).
- We only have a few small examples at the present time (no infinite classes are known – even for type-1 LRPA(n)
- We describe the method we used to construct type-1,2,3,4 LRPA(16) and type-1,2,3,4 LRPA(36), illustrating the technique by constructing a type-1,2,3,4 LRPA(16).

Lemma

Let $n \geq 2$ be even, and suppose there exists an $\frac{n}{2} \times \frac{n}{2}$ latin square S with the property that for all i with $2 \leq i \leq \frac{n}{2}$, the $i \times \left\lceil \frac{n}{i} \right\rceil$ rectangle in the upper left hand corner of S contains each of the symbols from 1 to $\frac{n}{2}$ at least twice. Then there exists a type-1,2,3,4 LRPA(n).

Constructing latin RPA

Step 1

- Each of the $i \times \left\lceil \frac{n}{i} \right\rceil$ rectangles in the upper left hand corner of S contains each symbol x with $1 \leq x \leq \frac{n}{2}$ twice.
- By considering each such rectangle in turn and replacing appropriately chosen copies of x by $n + 1 - x$ we construct a new array S' for which each of the $i \times \left\lceil \frac{n}{i} \right\rceil$ rectangles in the upper left corner contain each of the symbols from 1 to n.

Constructing latin RPA

Step 1

- Each of the $i \times \left\lceil \frac{n}{i} \right\rceil$ rectangles in the upper left hand corner of S contains each symbol x with $1 \leq x \leq \frac{n}{2}$ twice.
- By considering each such rectangle in turn and replacing appropriately chosen copies of x by $n + 1 - x$ we construct a new array S' for which each of the $i \times \left\lceil \frac{n}{i} \right\rceil$ rectangles in the upper left corner contain each of the symbols from 1 to n.

- For each symbol x, we need to decide which occurrences of x to replace by $n + 1 - x$.
- We do this by constructing a certain bipartite graph whose vertices are the cells containing x, then 2-coloring the vertices of this graph.

Constructing latin RPA

Step 2

Now we let S' form the top left corner of A, and "reflect" it by applying the symmetry condition $a_{i,j} + a_{i,n+i-j} = n + 1$, to fill in the top right corner of A. Finally, we carry out a similar reflection vertically to fill in the rest of A. The result is an latin RPA that is symmetric under rotation through 180 degrees.

Constructing latin RPA

Step 2

Now we let S' form the top left corner of A, and "reflect" it by applying the symmetry condition $a_{i,j} + a_{i,n+i-j} = n + 1$, to fill in the top right corner of A. Finally, we carry out a similar reflection vertically to fill in the rest of A. The result is an latin RPA that is symmetric under rotation through 180 degrees.

We need an example.

Constructing a latin RPA(16)

The square below is an 8 × 8 latin square S with the required properties. Note that the shaded cells are cells that are contained in basic rectangles in the upper left corner of the resulting 16 × 16 latin RPA.

1	2	3	4	7	8	6	5
2	5	6	7	4	1	3	8
3	6	5	8	1	2	4	7
4	7	8	1	2	3	5	6
7	4	1	2	5	6	8	3
8	1	2	3	6	5	7	4
6	3	4	5	8	7	1	2
5	8	7	6	3	4	2	1

Constructing a latin RPA(16)

We now adjust the entries in the top left rectangles so that each basic rectangle contains all the numbers from 1 to 16:

1	2	3	4	10	9	11	12
15	5	6	7	13	16	14	8
14	11	12	8	16	2	4	7
13	10	9	16	2	3	5	6
7	4	16	2	5	6	8	3
8	1	2	3	6	5	7	4
6	3	4	5	8	7	1	2
12	9	7	6	3	4	2	1

Constructing a latin RPA(16)

Finally, we “reflect” the result to obtain a type-1,2,3,4 latin RPA(16)

1	2	3	4	10	9	11	12	5	6	8	7	13	14	15	16
15	5	6	7	13	16	14	8	9	3	1	4	10	11	12	2
14	11	12	8	16	2	4	7	10	13	15	1	9	5	6	3
13	10	9	16	2	3	5	6	11	12	14	15	1	8	7	4
7	4	16	2	5	6	8	3	14	9	11	12	15	1	13	10
8	1	2	3	6	5	7	4	13	10	12	11	14	15	1	9
6	3	4	5	8	7	1	2	15	16	10	9	12	13	14	11
12	9	7	6	3	4	2	1	16	15	13	14	11	10	8	5
5	8	10	11	14	13	15	16	1	2	4	3	5	7	9	12
11	14	13	12	9	10	16	15	2	1	7	8	5	4	3	6
9	1	15	14	11	12	10	13	4	7	5	6	3	2	16	8
10	13	1	15	12	11	9	14	3	8	6	5	2	16	4	7
4	7	8	1	15	14	12	11	6	5	3	2	16	9	10	13
3	6	5	9	1	15	13	10	7	4	2	16	8	12	11	14
2	12	11	10	4	1	3	9	8	14	16	13	7	6	5	15
16	15	14	13	7	8	6	5	12	11	9	10	4	3	2	1

Open problems

1. Does there exist a type- 1,2,3,4 RPA(n) for all odd positive integers n?

2. Find an infinite class of type-1 latin RPA's. Better yet, prove that for all positive integers n, there exists a type-1,2 latin RPA(n).

We conjecture that there exists a type-1,2,3,4 latin RPA(n) for all positive integers n.

Open problems

1. Does there exist a type- 1,2,3,4 RPA(n) for all odd positive integers n?

2. Find an infinite class of type-1 latin RPA's. Better yet, prove that for all positive integers n, there exists a type- 1,2 latin RPA(n).

3. What can be proven about the existence of gerechte designs?

Thanks!