



# Synthesizing **Minimal Tile Sets** for **Patterned DNA Self-Assembly**

Mika Göös & Pekka Orponen  
Aalto University (*School of Science and Technology*)

- 1 Previous Study
- 2 Problem Definition
- 3 Approach of Ma & Lombardi
- 4 Our Contributions

## Shapes modulo Scale

[Soloveichik & Winfree 2004]



Unsolvable

# Previous Study

## Shapes modulo Scale

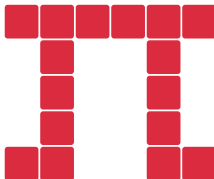
[Soloveichik & Winfree 2004]



Unsolvable

## Shapes

[Adleman et al. 2002]



NP-hard

# Previous Study

## Shapes modulo Scale

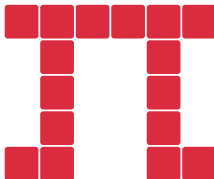
[Soloveichik & Winfree 2004]



Unsolvable

## Shapes

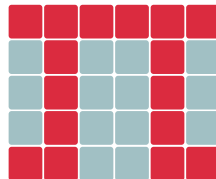
[Adleman et al. 2002]



NP-hard

## Patterns

[Ma & Lombardi 2008]

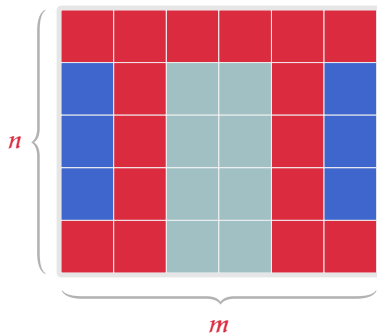


Not known?

# Pattern self-Assembly Tile set Synthesis (PATS)

## Input

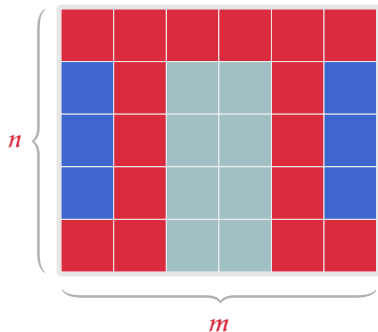
A  $k$ -colouring  $c : [m] \times [n] \rightarrow [k]$



# Pattern self-Assembly Tile set Synthesis (PATS)

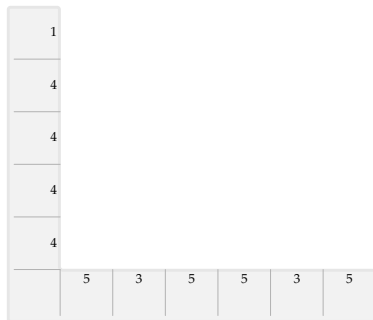
## Input

A  $k$ -colouring  $c : [m] \times [n] \rightarrow [k]$



## Output

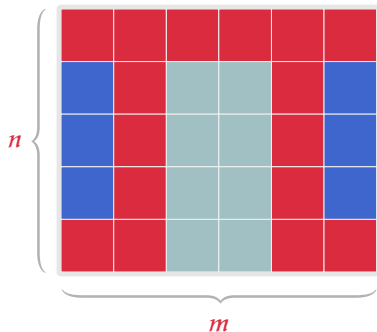
A Tile Assembly System  $\mathcal{T} = (T, S, s, 2)$



# Pattern self-Assembly Tile set Synthesis (PATS)

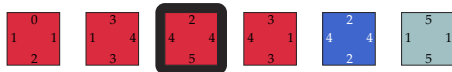
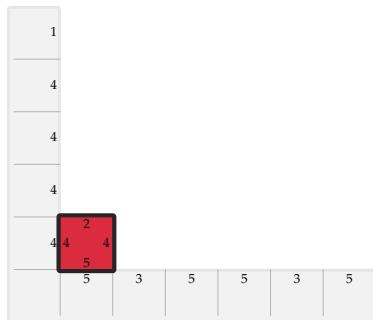
## Input

A  $k$ -colouring  $c : [m] \times [n] \rightarrow [k]$



## Output

A Tile Assembly System  $\mathcal{T} = (T, S, s, 2)$

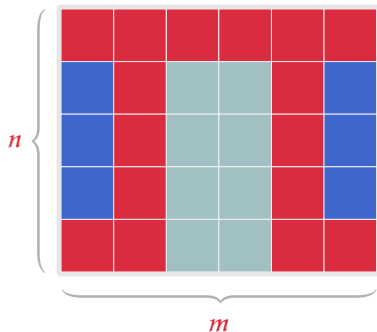




# Pattern self-Assembly Tile set Synthesis (PATS)

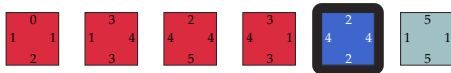
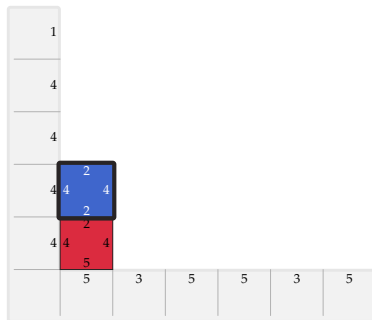
## Input

A  $k$ -colouring  $c : [m] \times [n] \rightarrow [k]$



## Output

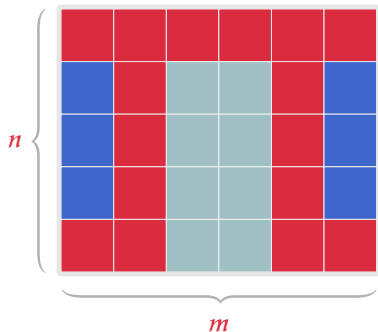
A Tile Assembly System  $\mathcal{T} = (T, S, s, 2)$



# Pattern self-Assembly Tile set Synthesis (PATS)

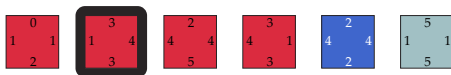
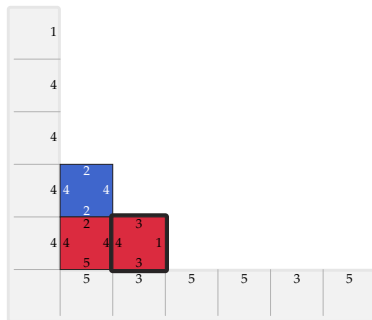
## Input

A  $k$ -colouring  $c : [m] \times [n] \rightarrow [k]$



## Output

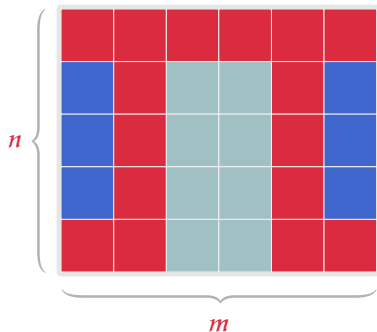
A Tile Assembly System  $\mathcal{T} = (T, S, s, 2)$



# Pattern self-Assembly Tile set Synthesis (PATS)

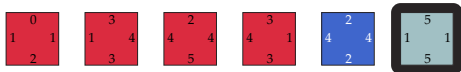
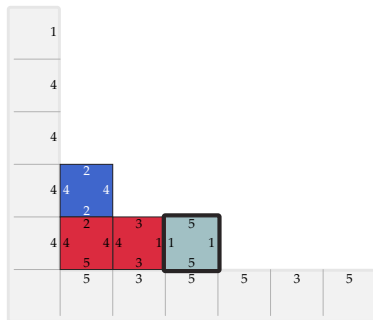
## Input

A  $k$ -colouring  $c : [m] \times [n] \rightarrow [k]$



## Output

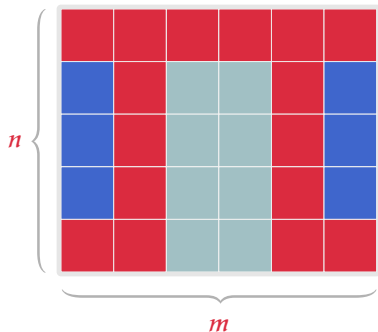
A Tile Assembly System  $\mathcal{T} = (T, S, s, 2)$



# Pattern self-Assembly Tile set Synthesis (PATS)

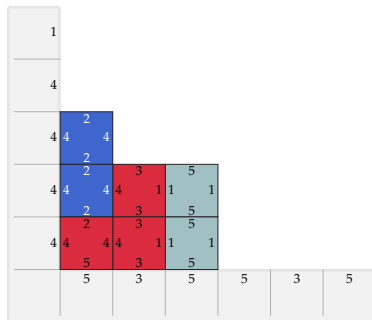
## Input

A  $k$ -colouring  $c : [m] \times [n] \rightarrow [k]$



## Output

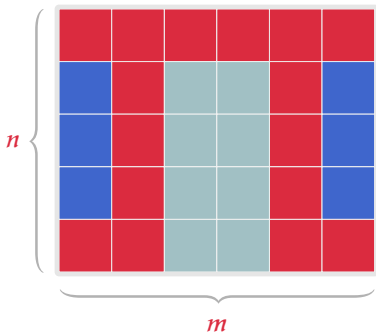
A Tile Assembly System  $\mathcal{T} = (T, S, s, 2)$



# Pattern self-Assembly Tile set Synthesis (PATS)

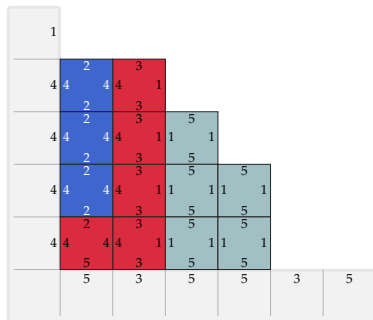
## Input

A  $k$ -colouring  $c : [m] \times [n] \rightarrow [k]$



## Output

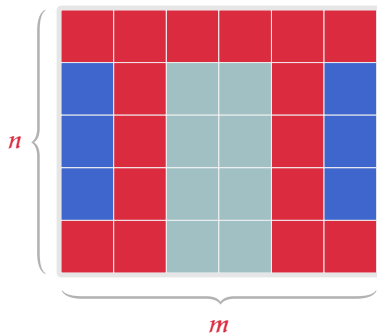
A Tile Assembly System  $\mathcal{T} = (T, S, s, 2)$



# Pattern self-Assembly Tile set Synthesis (PATS)

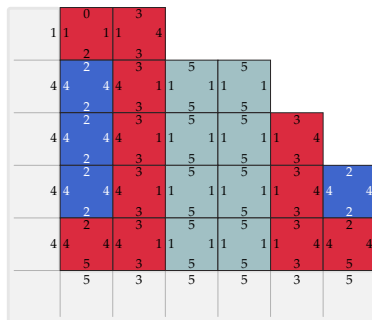
## Input

A  $k$ -colouring  $c : [m] \times [n] \rightarrow [k]$



## Output

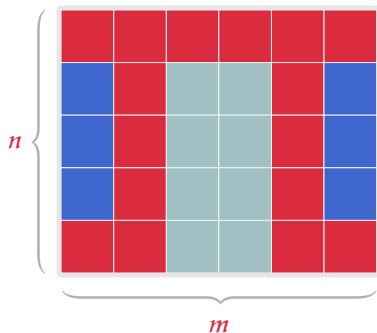
A Tile Assembly System  $\mathcal{T} = (T, S, s, 2)$



# Pattern self-Assembly Tile set Synthesis (PATS)

## Input

A  $k$ -colouring  $c : [m] \times [n] \rightarrow [k]$



## Output

A Tile Assembly System  $\mathcal{T} = (T, S, s, 2)$

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



# Pattern self-Assembly Tile set Synthesis (PATS)

**Given:** A  $k$ -colouring  $c : [m] \times [n] \rightarrow [k]$ .

**Find:** A tile assembly system  $\mathcal{T} = (T, \mathcal{S}, s, 2)$  s.t.

**P1.** The tiles in  $T$  have bonding strength 1.

**P2.** The domain of  $\mathcal{S}$  is  $[0, m] \times \{0\} \cup \{0\} \times [0, n]$  and all the terminal assemblies have the domain  $[0, m] \times [0, n]$ .

**P3.** There exists a colouring  $d : T \rightarrow [k]$  such that for each terminal assembly  $\mathcal{A} \in \text{Term } \mathcal{T}$  we have  $d(\mathcal{A}(x, y)) = c(x, y)$  for all  $(x, y) \in [m] \times [n]$ .



# Approach of Ma & Lombardi

	0	4	7	10	13	16
33	11	5	8	11	14	17
	2	6	9	12	15	18
21	19	22	24	26	28	30
	20	23	25	27	29	31
34	32	35	37	39	41	43
	33	36	38	40	42	44
47	45	48	50	52	54	56
	46	49	51	53	55	57
60	58	61	63	65	67	69
	59	62	64	66	68	70
	59	62	64	66	68	70

# Approach of Ma & Lombardi

	0	4	7	10	13	16
3	3 3	1 1	5 5	8 8	11 11	14 14
	2	6	9	12	15	18
21	21 19	19 22	22 24	24 26	26 28	28 30
	20	23	25	27	29	31
34	34 32	32 35	35 37	37 39	39 41	41 43
	33	36	38	40	42	44
47	47 45	45 48	48 50	50 52	52 54	54 56
	46	49	51	53	55	57
60	60 58	58 61	61 63	63 65	65 67	67 69
	59	62	64	66	68	70

# Approach of Ma & Lombardi

	0	4	7	10	13	16	
3	3	1 1	5 5	8 8	11 11	14 14	17
	2	6	9	12	15	18	
21	21	19 19	22 22	24 24	26 26	28 28	30
	20	23	25	27	29	31	
34	34	32 32	35 35	37 37	39 39	41 41	43
	33	36	38	40	42	44	
47	47	45 45	48 48	50 50	52 52	54 54	56
	46	49	51	53	55	57	
60	60	58 58	61 61	63 63	65 65	67 67	69
	59	62	64	66	68	70	

**To minimize Tile set size:**

- Merge glues
- Merge tiles

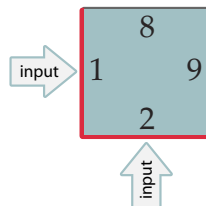
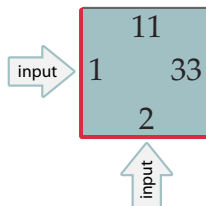
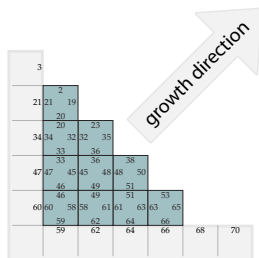
**If conflicts arise:**

- Continue merging!

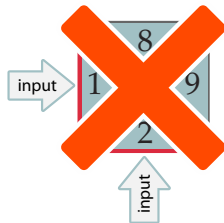
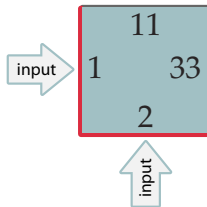
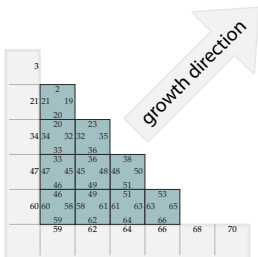
## We present

- 1 Extension of the work of Ma & Lombardi
- 2 Branch & Bound algorithm
- 3 Pruning heuristics

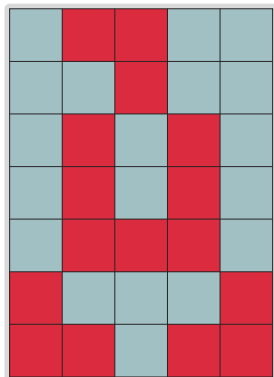
Lemma: Minimal solutions to the PATS problem are *deterministic*



Lemma: Minimal solutions to the PATS problem are *deterministic*



# Partition Centric View



# Partition Centric View

	0	2	0	2	2	
33	11	33	33	33	33	3
	2	0	4	0	0	
33	33	11	55	33	33	1
	0	2	2	0	2	
33	11	33	33	33	33	3
	2	0	0	4	0	
	2	0	0	4	0	
33	33	33	11	55	33	3
	0	4	2	2	0	
33	11	55	11	33	33	1
	2	2	2	0	2	
	2	2	2	0	2	
11	33	33	33	11	3	
	0	0	0	2	0	
	0	0	0	2	0	
33	33	33	11	33	3	
	4	4	2	0	4	
	4	4	2	0	4	

2	0
1 3	3 3
0	4

4	2
1 5	5 1
2	2

0	0
5 3	3 1
0	2

2
3 3
0



# Partition Centric View

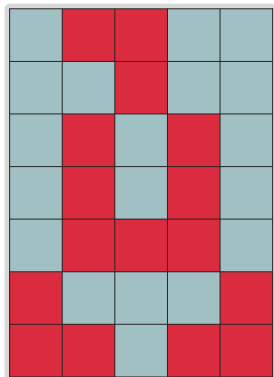
	0	2	0	2	2
33	11	33	33	33	3
	2	0	4	0	0
33	33	11	55	33	1
	0	2	2	0	2
33	11	33	33	33	3
	2	0	0	4	0
	2	0	0	4	0
33	33	33	11	55	3
	0	4	2	2	0
	0	4	2	2	0
33	11	55	11	33	1
	2	2	2	0	2
	2	2	2	0	2
11	33	33	33	11	3
	0	0	0	2	0
	0	0	0	2	0
33	33	33	11	33	3
	4	4	2	0	4
	4	4	2	0	4



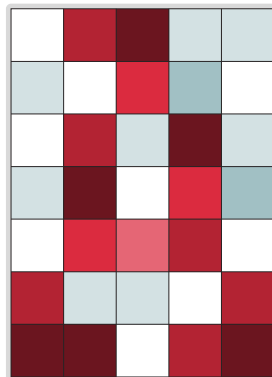
1	6	7	2	2
2	1	5	3	1
1	6	2	7	2
2	7	1	5	3
1	5	4	6	1
6	2	2	1	6
7	7	1	6	7

**Constructible** partition of  $[m] \times [n]$

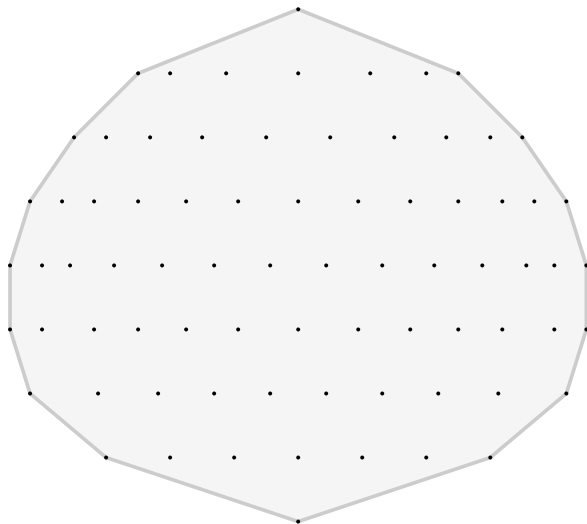
# Partition Centric View



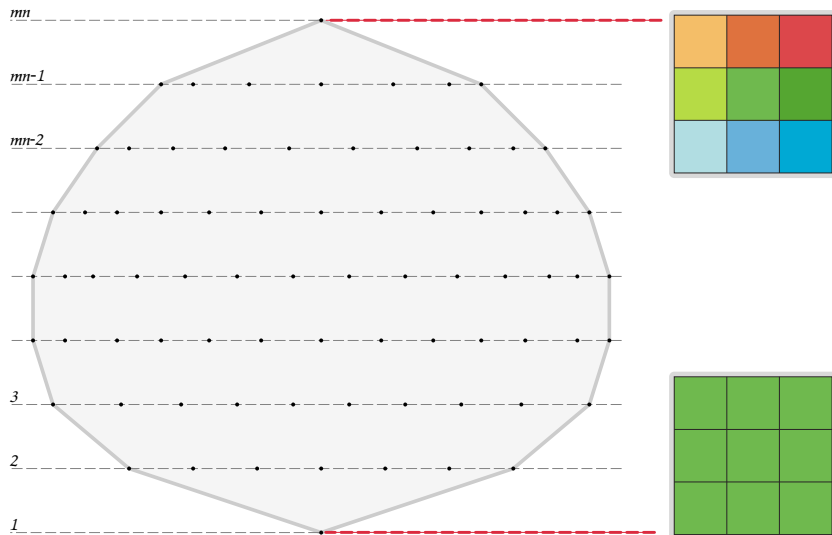
is  
**coarser**  
than



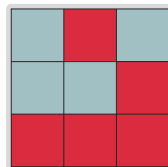
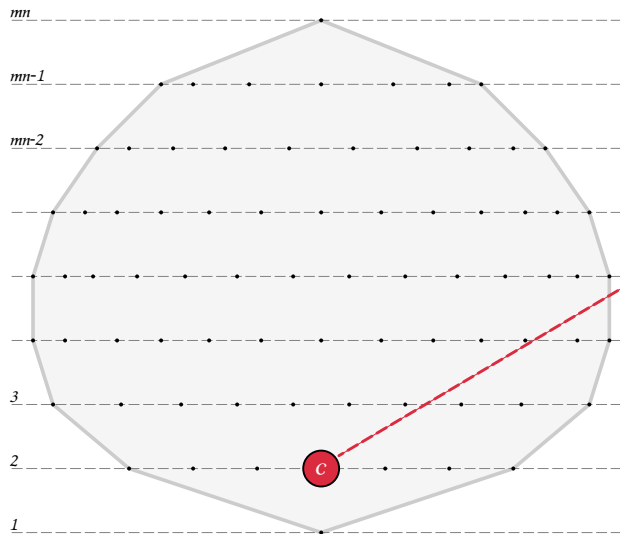
# Searching the Lattice of Partitions



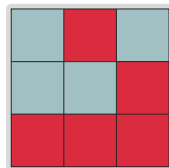
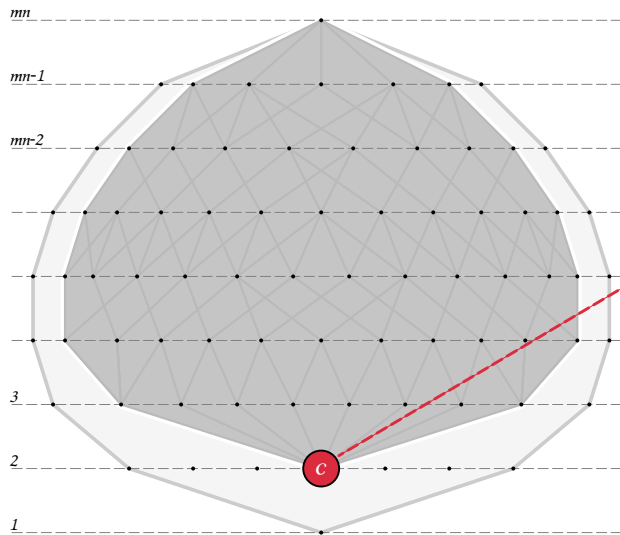
# Searching the Lattice of Partitions



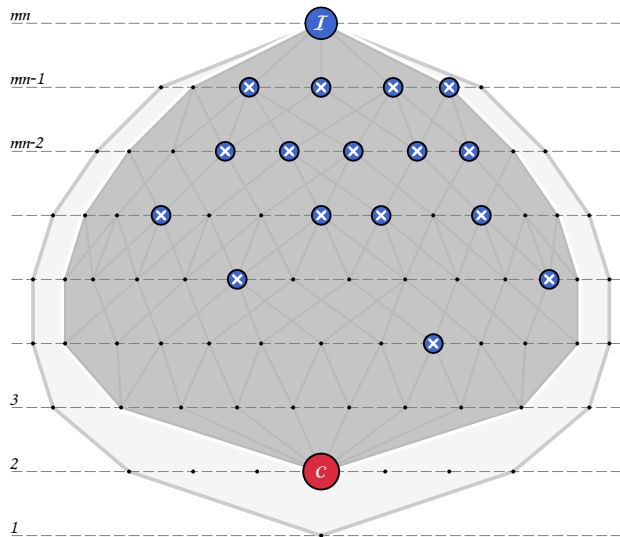
# Searching the Lattice of Partitions



# Searching the Lattice of Partitions

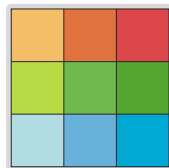
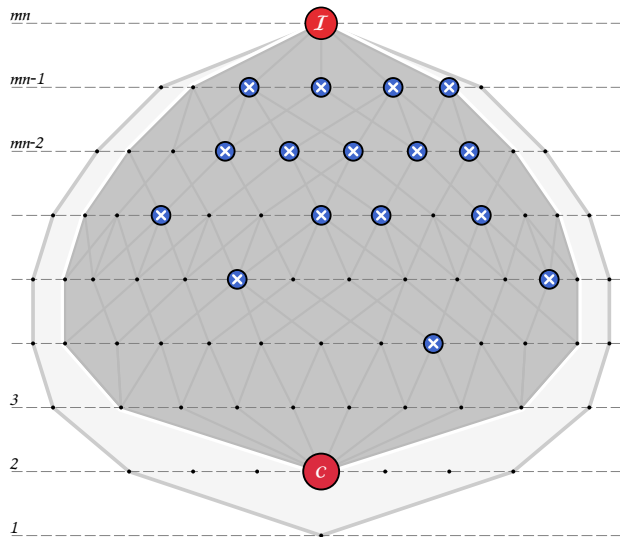


# Searching the Lattice of Partitions



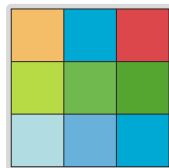
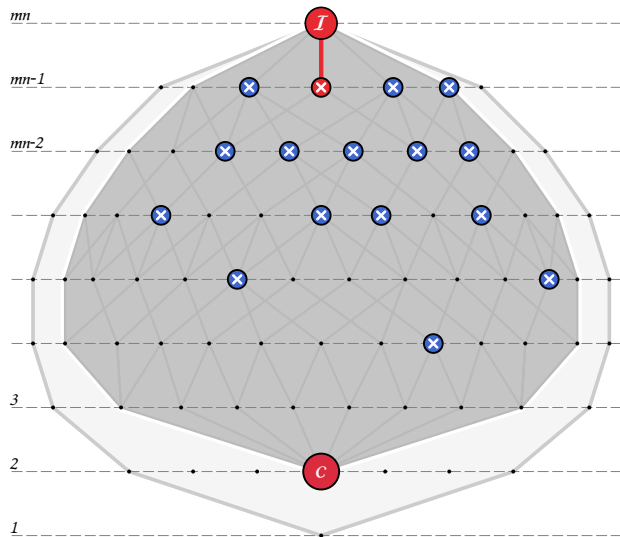
  
=  
**Constructible**  
partition

# Searching the Lattice of Partitions

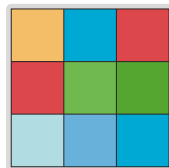
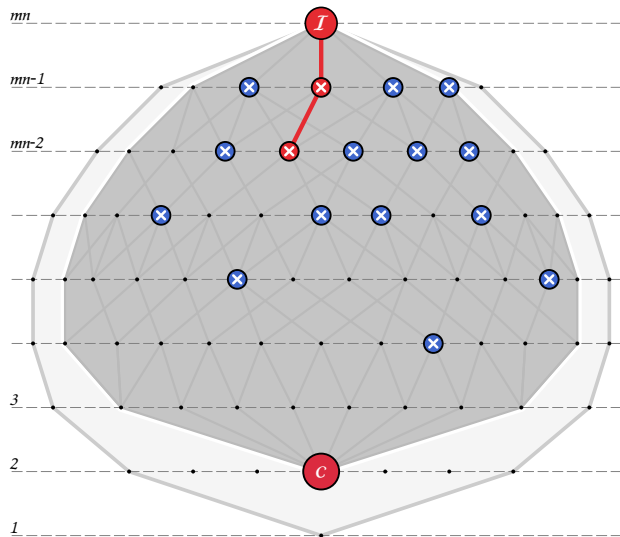




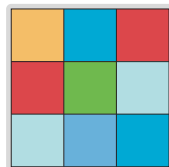
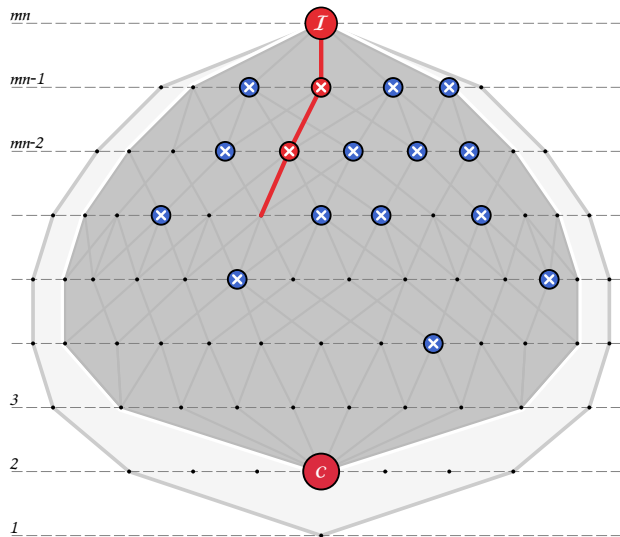
# Searching the Lattice of Partitions



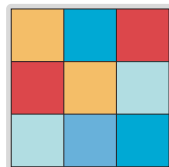
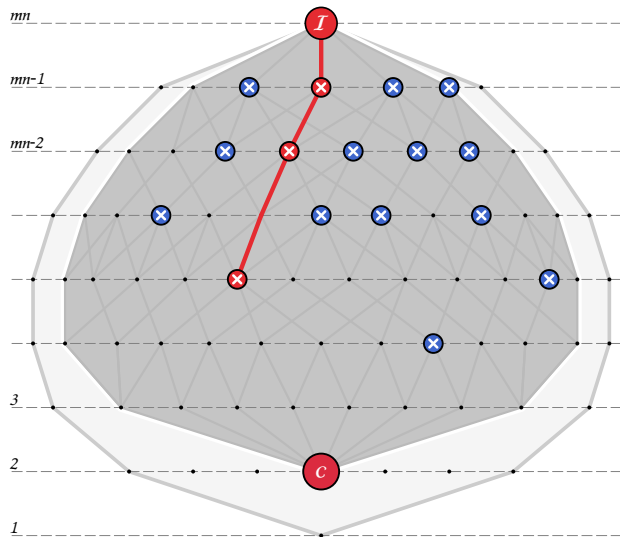
# Searching the Lattice of Partitions



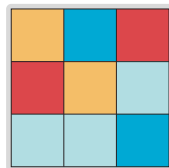
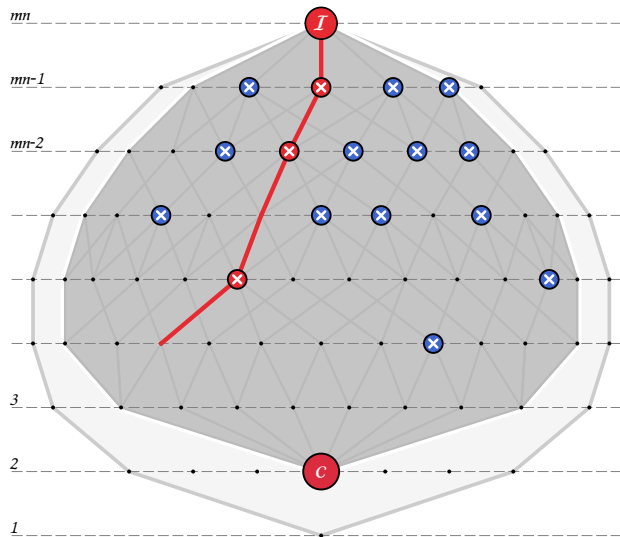
# Searching the Lattice of Partitions



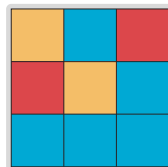
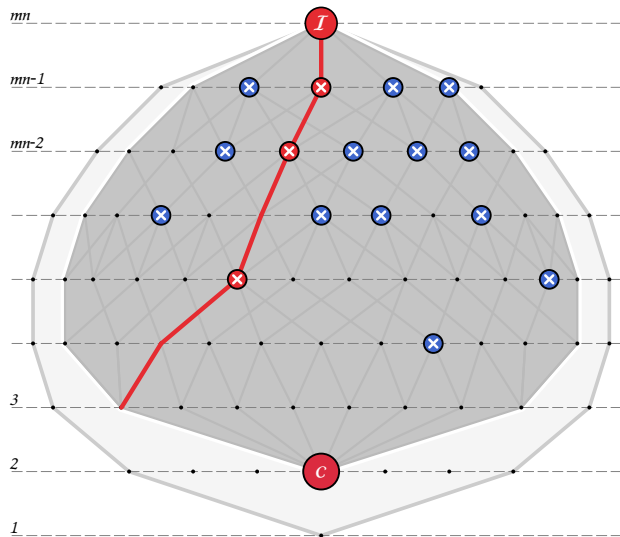
# Searching the Lattice of Partitions



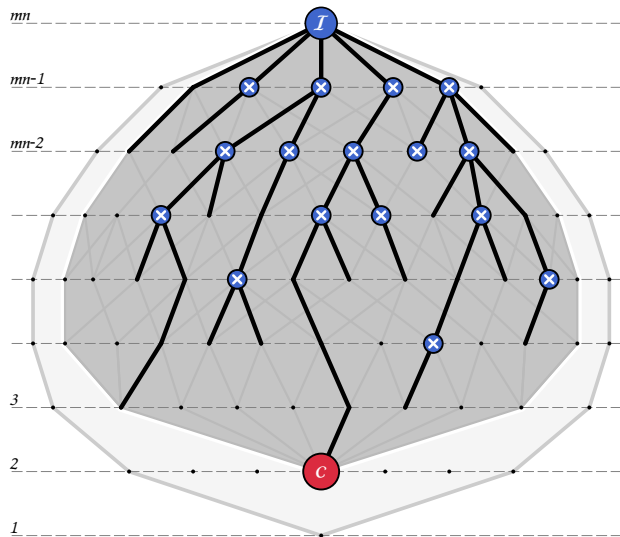
# Searching the Lattice of Partitions



# Searching the Lattice of Partitions



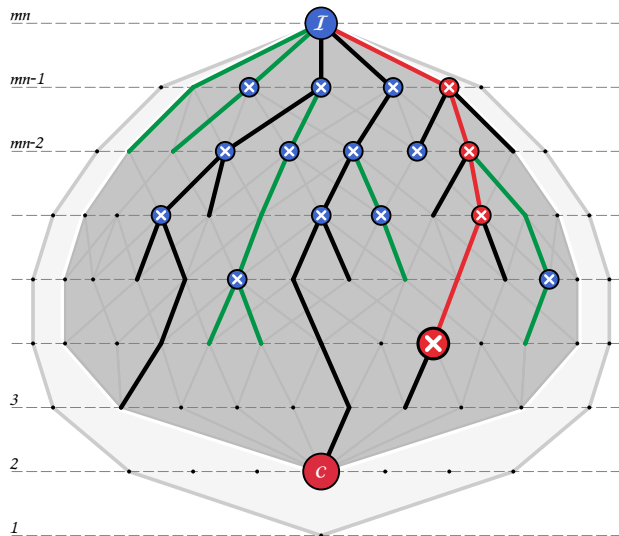
# Searching the Lattice of Partitions



## Our B&B algorithm

- 1 Node-disjoint search tree
- 2 Uses memory  $\text{poly}(mn)$
- 3 Branching *only* on **constructible** partitions

# Searching the Lattice of Partitions

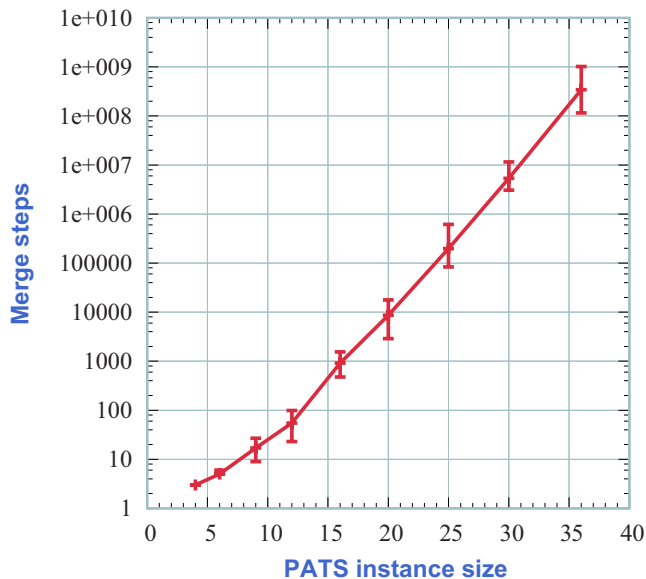


## Our B&B algorithm

- 1 Node-disjoint search tree
- 2 Uses memory **poly**( $mn$ )
- 3 Branching *only* on **constructible** partitions
- 4 Cheap **bounding** function



# Running time on random 2-coloured instances



$$\sim 2^{mn}$$

## PATS problem remains challenging

- Open Problems:
  - 1 Is it **NP**-hard?
  - 2 Faster algorithms?
  - 3 Generalize to infinite finite-period patterns
- PATS is of practical importance



# Thank you!