CS4402: Constraint Programming

Week 4, Lecture 2: Constraint Modelling – Thinking Abstractly (5)

With thanks to Alan Frisch

Nesting Inside Sets

Nesting Inside Sets

- Being asked to find a set of some other object is common, so it is worth considering how to model this type of problem.
- Now we must choose how to model the outer type (e.g. explicit vs occurrence model of sets) **as well** as the inner.

Nested Sets

Consider the following simple problem class:

- Given **m**, **n**.
- Find a cardinality-**m set of sets** of **n** digits such that …
- From what we have seen so far, we have three possibilities:
  1. An occurrence representation.
  2. Outer: Explicit. Inner: Occurrence.
  3. Outer: Explicit. Inner: Explicit.

Nesting Inside Sets: Occurrence

- Recall the occurrence representation of a fixed-cardinality set of digits:

	0	1	2	3	4	5	6	7	8	9
O	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1

- We have an index per possible element of the set.

Nesting Inside Sets: Occurrence

Can we take the same approach here?

- Given **m**, **n**.
- Find a cardinality-**m** **set of sets** of **n** digits such that…

Introduce an array indexed by the possible sets of **n** digits!

{1, 2, 3}	{1, 2, 4}	{1, 2, 5}	
0,1	0,1	0,1	…

(assuming **n** = 3)

This is often not feasible.
Typically, when dealing with nesting the outer layers are represented **explicitly**.

Nesting Inside Sets: Outer Explicit

- Recall the explicit representation of a fixed-cardinality set of digits:

E	1	2	3	4		n
	0..9	0..9	0..9	0..9	…	0..9

- Similarly to the sequence example, we extend the dimension of **E** according to the representation we choose for the inner set.
- We're also going to have to be careful to make sure the elements of the outer set are **distinct**.

Nesting Inside Sets: Explicit/Occurrence

- Given **m**, **n**.
- Find a cardinality-**m** **set of sets** of **n** digits such that…
- Let's consider an occurrence representation for the inner sets.

But what about **symmetry?**

EO	1	2	…	m
0	0,1	0,1		0,1
1	0,1	0,1		0,1
2	0,1	0,1		0,1
…				
9	0,1	0,1		0,1

Constraints:

- Sum(col i of **EO**) = **n** (forAll i in 1..**m**)
- Scalar-prod(col i of **EO**, col j of **EO**) $\neq$ **n** (forAll $\{i, j\}$ in 1..**m**)
Modelling alldiff on cols here.

Nesting Inside Sets: Explicit/Occurrence

- Given **m**, **n**.
- Find a cardinality-**m** **set of sets** of **n** digits such that…
- Let's consider an occurrence representation for the inner sets.

EO	1	2	…	m
0	0,1	0,1		0,1
1	0,1	0,1		0,1
2	0,1	0,1		0,1
…				
9	0,1	0,1		0,1

Constraints:

- Sum(col i of **EO**) = **n** (forAll *i* in 1..**m**)
- col i of **EO** $<_{lex}$ col i+1 of **EO**) (forAll *i* in 1..**m-1**)

Nesting Inside Sets: Explicit/Explicit

- Given **m**, **n**.
- Find a cardinality-**m set of sets** of **n** digits such that…
- Let's consider an occurrence representation for the inner sets.

EE	1	2	…	m
1	0..9	0..9		0..9
2	0..9	0..9		0..9
3	0..9	0..9		0..9
…				
n	0..9	0..9		0..9

Constraints:

- AllDiff on each column.
- col i of **EE** $<_{lex}$ col i+1 of **EE**) (forAll *i* in 1..**m-1**)
- More **symmetry?**

Nesting Inside Sets: Explicit/Explicit

- Given **m**, **n**.
- Find a cardinality-**m set of sets** of **n** digits such that…
- Let's consider an occurrence representation for the inner sets.

EE	1	2	…	m
1	0..9	0..9		0..9
2	0..9	0..9		0..9
3	0..9	0..9		0..9
…				
n	0..9	0..9		0..9

Constraints:

- ~~AllDiff on each column.~~
- col i of **EE** $<_{lex}$ col i+1 of **EE**) (forAll *i* in 1..**m-1**)
- More **symmetry? Yes: ascending order within each column**

Relations as Sets of Tuples

- Last time we looked at a couple of ways of modelling relations.
- We can also view relations as **sets of tuples**.
- Recall our example:
  - Find a relation R between sets
    A = {1, 2, 3} and B = {2, 3, 4} such that…
- What happens when we try and model this from the perspective of a set of tuples?

Relations as Sets of Tuples: Occurrence

- We have an array indexed by the possible tuples:

<1,2>	<1,3>	<1,4>	…
0,1	0,1	0,1	

- Basically same as the occurrence representation we came up with directly:

		B		
		2	3	4
A	1	0,1	0,1	0,1
	2	0,1	0,1	0,1
	3	0,1	0,1	0,1

Relations as Sets of Tuples: Explicit

- Find a relation R between sets
  A = {1, 2, 3} and B = {2, 3, 4} such that…
- Maximum number of tuples is 9. Invoke our bounded-cardinality set pattern:

	1	3	4	5	…
1	1..3	1..3	1..3	1..3	
2	2..4	2..4	2..4	2..4	

What about **symmetry**?
What if the relation allows fewer than the full 9 tuples?

The Social Golfers Problem

The Social Golfers Problem

- In a golf club there are a number of golfers who wish to play together in **g** groups of size **s**.
- Find a schedule of play for **w** weeks such that no pair of golfers play together more than once.

The Social Golfers Problem: Modelling

- In each week, we need to **partition** the golfers into groups.
  - A partition is a set of sets. No pair of inner sets have an element in common.
- What about the weeks?
  - A sequence? But what does the order matter?
  - A set.

- So we can think of the problem as finding a **set of partitions**.

Golfers: Representing the Outer set

- We have seen explicit and occurrence representations of sets.
- The set contains complex objects (partitions).
- Indexing an array by the possible partitions of golfers doesn't seem appealing.
- So let's try an explicit model:

1	2	3	4	…	w
?	?	?	?		?

Golfers: The Partitions

- In each week we want to partition the golfers into **g** groups of size **s**.
- That is, a set of cardinality **g** of sets of cardinality **s**.
- As per the previous discussion, probably sensible to represent the outer set explicitly.
- The inner set could be occurrence or explicit. Here we'll talk about an explicit/explicit representation.

Golfers: The Partitions

- Let **n** = number of golfers = **g** * **s**.

A Week		Group Size			
		1	2	…	s
Groups	1	1..n	1..n		1..n
	2	1..n	1..n		1..n
	3	1..n	1..n		1..n
	…				
	g	1..n	1..n		1..n

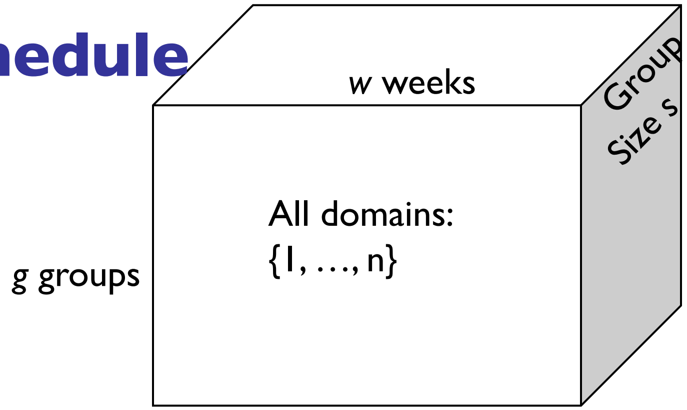
Since a week is a partition, what can we say about the elements of **week**?

What about **symmetry**?

A set of Partitions of Golfers

- If we put **week** into each slot of our set representation, we obtain a 3d array:

Schedule



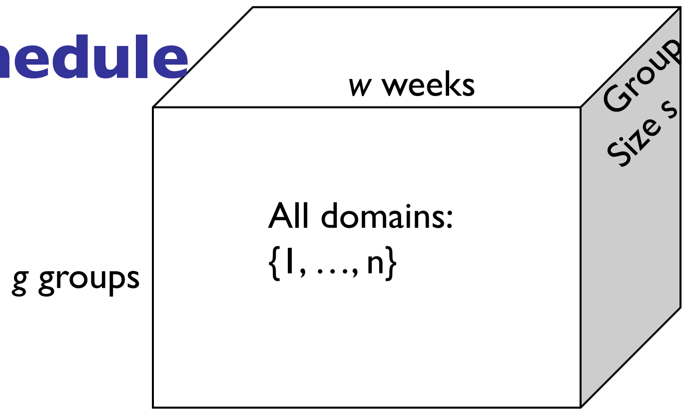
NB n = g x s is no of golfers

We can **order the weeks** lexicographically to counter the equivalence of assignments obtained by permuting the weeks.

A set of Partitions of Golfers

- Need to ensure no pair of golfers meet more than once.

Schedule



NB $n = g \times s$ is no of golfers

Equivalently: size of intersection of each pair of groups is at most 1. Invoking our intersection pattern:

Intersection 1..n

Switches 0, 1

Sum of switches is at most 1 (one)

Social Golfers

- Solution to the instance with 3 groups (size 3) over 3 weeks:

	3 weeks		
3 groups, size 3	[1, 2, 3]	[1, 4, 7]	[1, 5, 9]
	[4,5,6]	[2,5,8]	[2,6,7]
	[7,8,9]	[3,6,9]	[3,4,8]

We've missed a symmetry! Can you spot it?

Nesting Summary

- Modelling problems involving nested combinatorial objects can be quite tricky.
- Using the patterns we've been looking at can help you to do it **systematically**.
- It can also help in spotting **equivalence classes** of assignments as you introduce them.
  - Which can be substantially cheaper than trying to detect symmetry after the fact.

Viewpoints Redux

Viewpoints

- Fundamental to the formulation of any constraint model is the selection of a **viewpoint**:
  - A set of decision variables and domains sufficient to characterise the problem.
- From this choice, the rest of the model (I.e. the constraints) follows.

Viewpoint Selection

- We have seen that selecting a good viewpoint is essential:
  - Make the wrong choice and writing down the constraints can be very awkward, probably leading to poor solving behaviour.
- What if a viewpoint makes expressing **some** of the constraints easy?
  - Might want to keep this viewpoint and use **auxiliary variables** to express the remaining constraints.

Auxiliary Variables

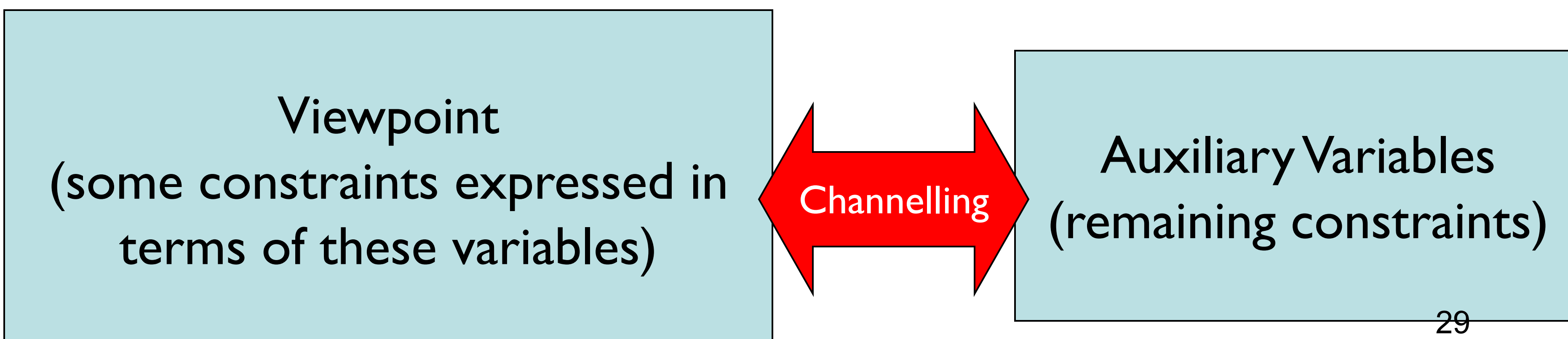
- Each variable in a CSP represents a choice that must be made to solve the problem being modelled.
- But by definition the original viewpoint is sufficient to represent all the choices in the original problem.
- These extra variables are **auxiliary** in the sense that they are not needed to characterise the problem.

Viewpoint
(some constraints expressed in terms of these variables)

Auxiliary Variables
(remaining constraints)

Channelling Constraints

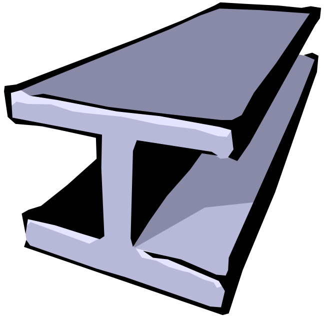
- The choices that the auxiliary variables represent are also represented by the original viewpoint.
- We must make sure that the two sets of variables are **consistent**.
  - **Channelling** constraints:



Auxiliary Variables Example

Steel Mill Slab Design

Decision Variables: The Order Matrix

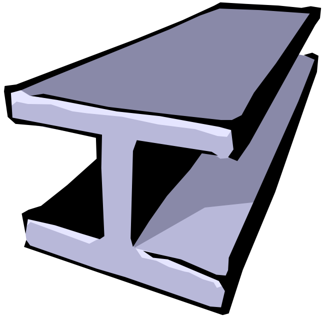


		Orders			
		a	b	c	d
Slabs	1	0	0	1	1
	2	0	1	0	0
	3	1	0	0	0
	4…	0	0	0	0

Can you see the symmetry?

What kind of abstract variable does this matrix represent?

Auxiliary Variables: The Colour Matrix

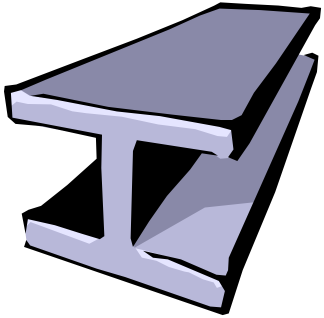


		Colours			
		red	green	blue	orange
Slabs	1	0	0	1	1
	2	0	1	0	0
	3	1	0	0	0
	4…	0	0	0	0

Channelling constraints?

What kind of abstract variable does this matrix represent?

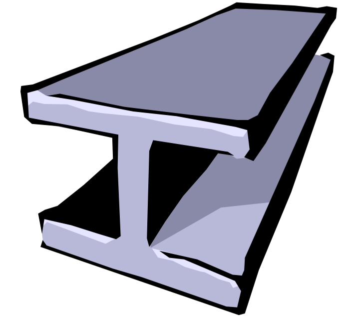
Auxiliary Variables: The Colour Matrix

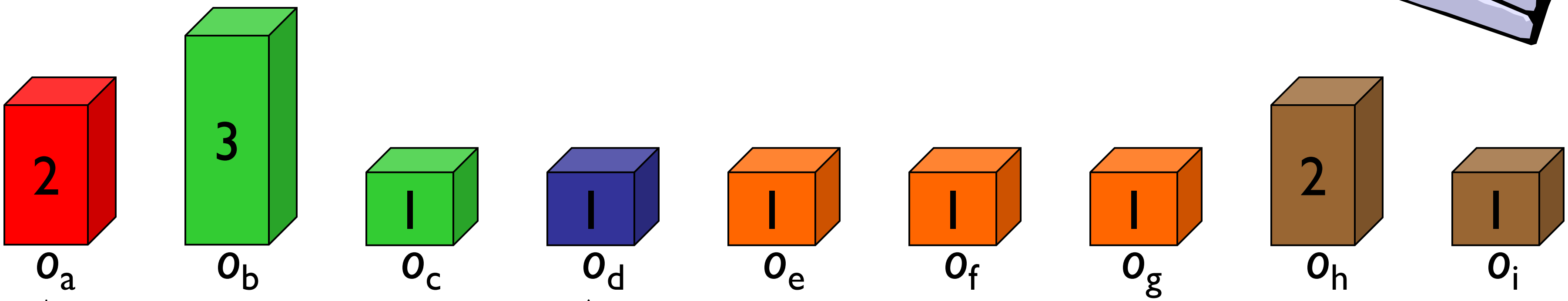


		Colours			
		red	green	blue	orange
Slabs	1	0	0	1	1
	2	0	1	0	0
	3	1	0	0	0
	4…	0	0	0	0

If Orders[i,j] = 1 Then Colours[colour_of(i), j] = 1

Example

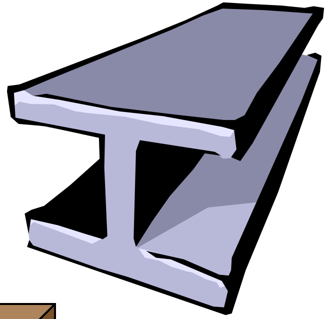


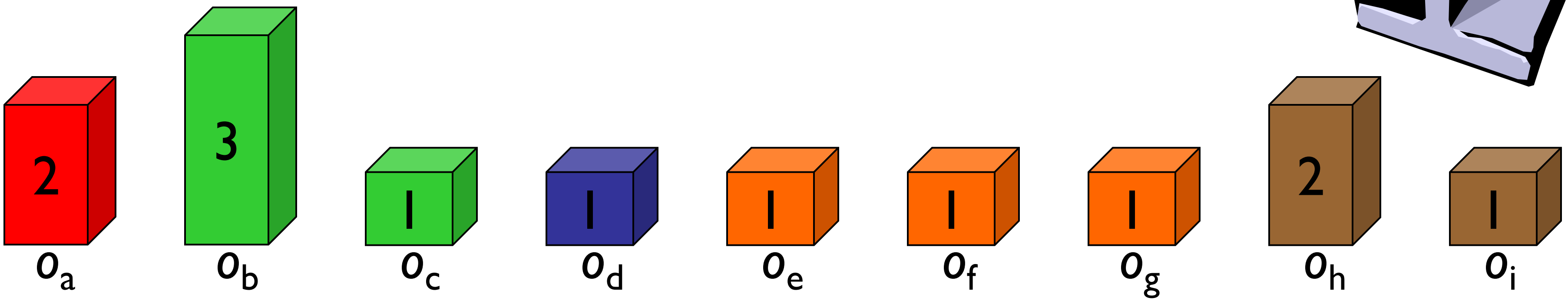


	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*
1	1								
2				1					
…									

	Red	Green	Blue	Orange	Brown
1	1				
2			1		
…					

Solution





$s_1 = 4, s_2 = 3, s_3 = 3, s_4 = 3, s_i = 0 \ (5 \leq i \leq 9)$

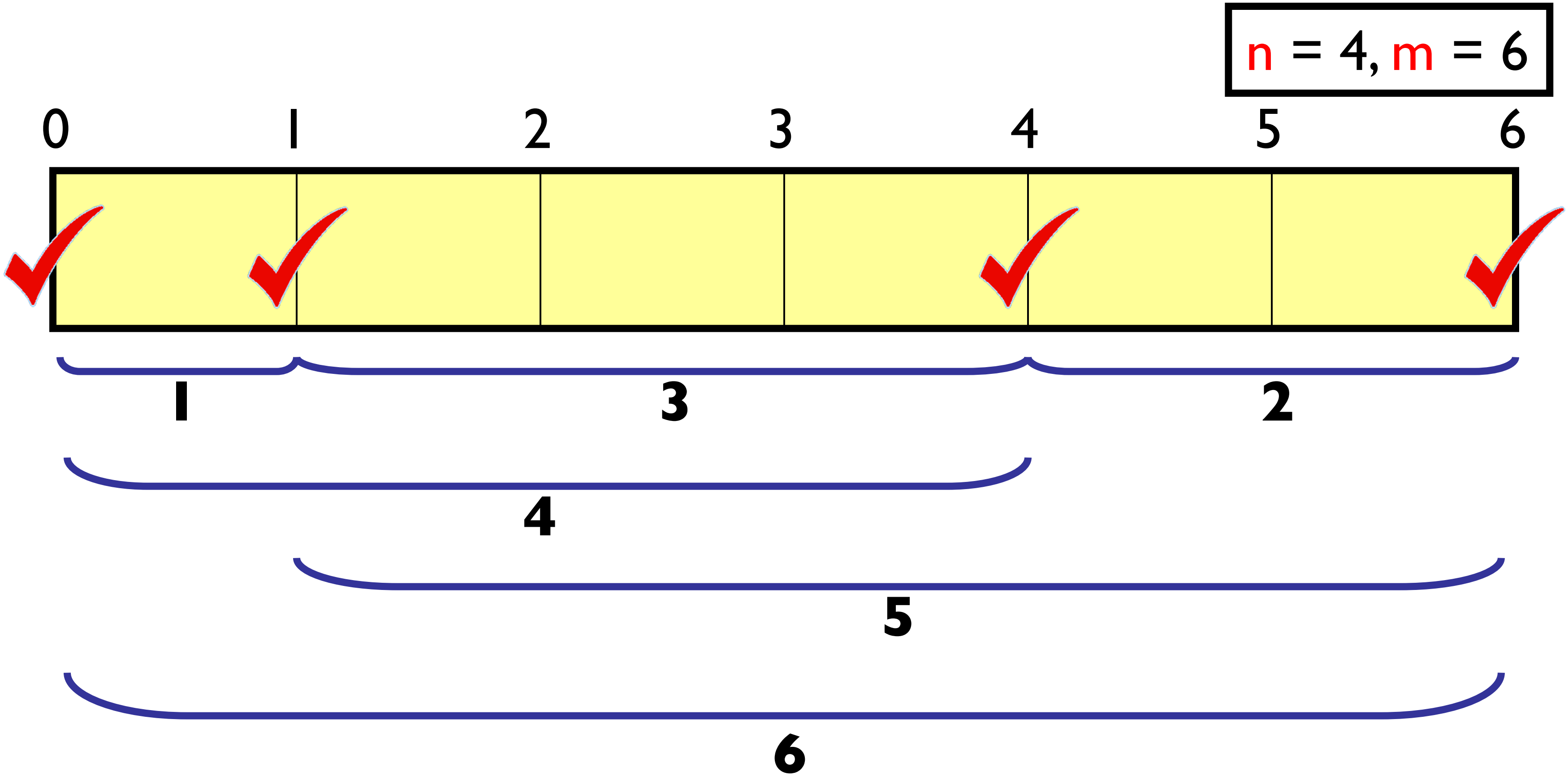
	o_a	o_b	o_c	o_d	o_e	o_f	o_g	o_h	o_i
1	0	0	0	0	0	0	**1**	**1**	**1**
2	**1**	0	**1**	0	0	0	0	0	0
3	0	**1**	0	0	0	0	0	0	0
4	0	0	0	**1**	**1**	**1**	0	0	0
…	0	0	0	0	0	0	0	0	0

	Red	Green	Blue	Orange	Brown
1	0	0	0	**1**	**1**
2	**1**	**1**	0	0	0
3	0	**1**	0	0	0
4	0	0	**1**	**1**	0
…	0	0	0	0	0

Auxiliary Variables Example

Golomb Ruler

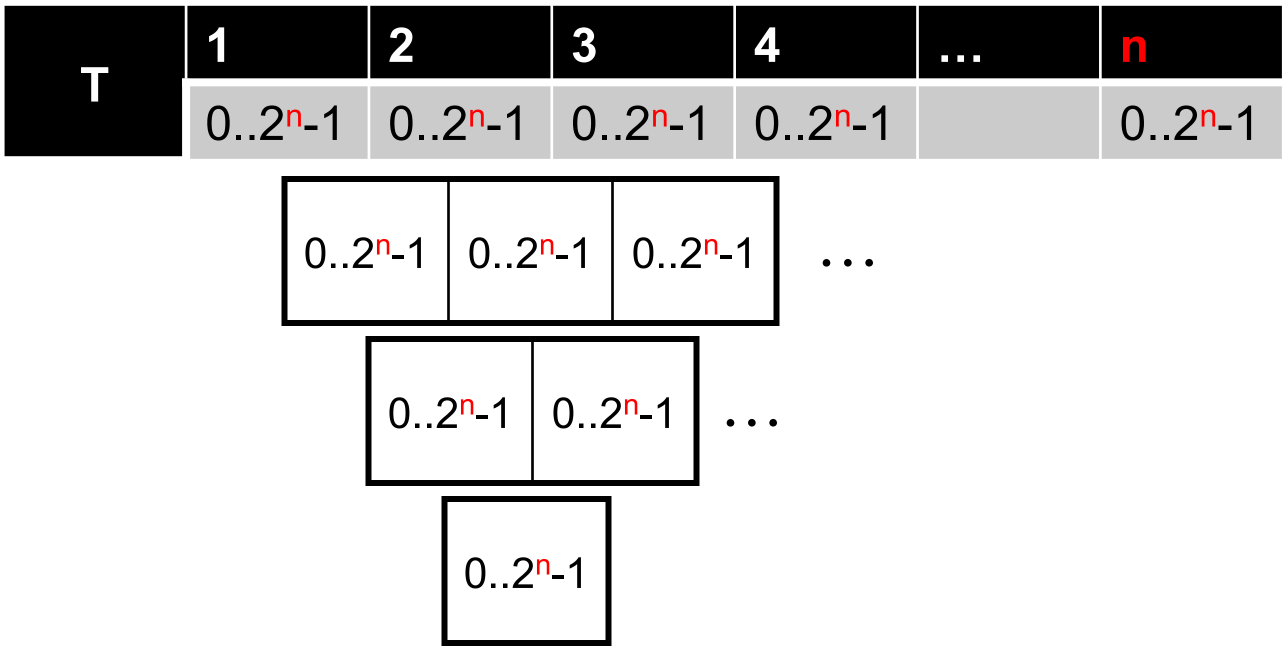
Golomb Ruler: Example



All Inter-tick Distances are Distinct

- This description immediately suggests the use of an all-different constraint.
- But all-different works on variables, not on pairs of variables.
- Solution?

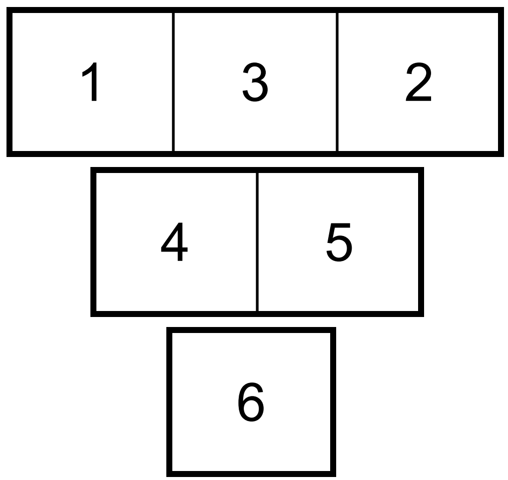
Distance Variables



- Introduce an auxiliary variable per distance.
- Channelling constraint: dij = T[i] - T[j]
- Distinct distances: alldifferent(dij)

Distance Variables

T	1	2	3	4
	0	1	4	6



- A solution when n = 4.