Thinking Abstractly About Constraint Modelling

Ian Miguel
ianm@cs.st-andrews.ac.uk

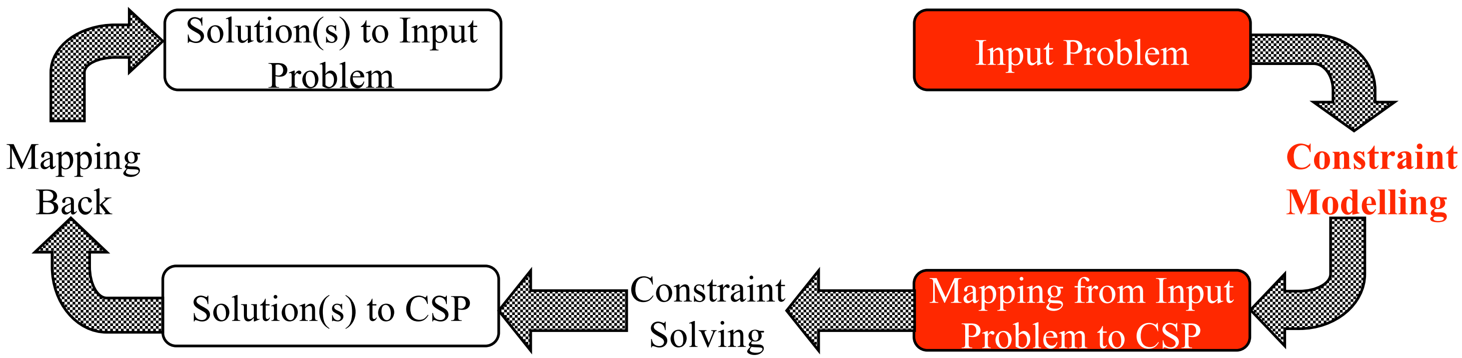
Constraints

A Brief Recap

Constraint Solving

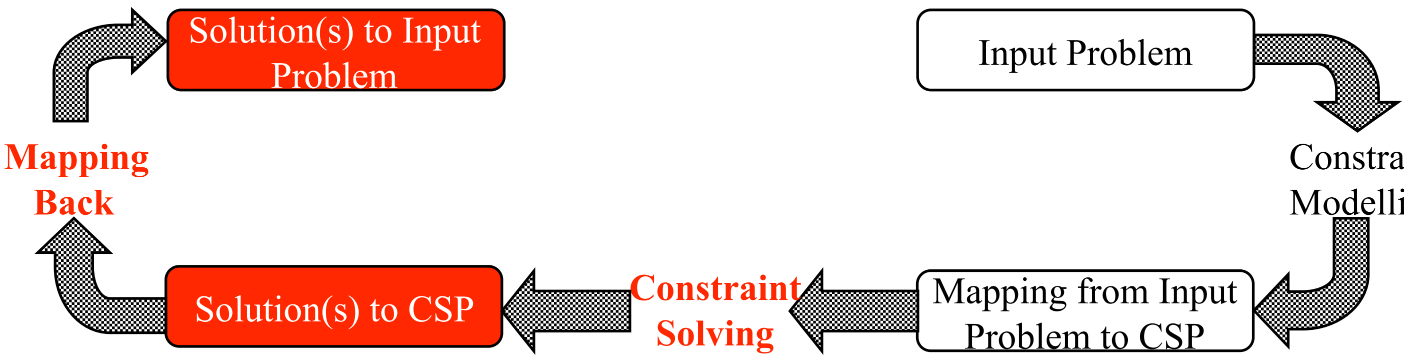
- Offers an efficient means of finding solutions to combinatorial problems.
- A **constraint model** is a description of a combinatorial problem in a format suitable for input to a **constraint solver**.
- Constraint solver searches for solutions to the problem automatically.

Constraint Modelling & Solving



- A constraint model is a description of a combinatorial problem in terms of a **constraint satisfaction problem** (CSP).
  - The features of a given problem are mapped onto the features of a CSP.

Constraint Modelling & Solving



- The CSP is input to a constraint solver, which produces a solution (or solutions).
- The model is used to map the solution(s) back onto the original problem.

Assumptions

1. Input problem and constraint model are **finite**.
2. The problem to be modelled is known completely to the modeller.
   - In practice, **knowledge elicitation** and iterative modelling may be required.

Constraint Satisfaction Problems

- A finite-domain constraint satisfaction problem comprises:
  - A finite set of **decision variables**.
  - For each decision variable, a finite **domain** of potential values.
  - A finite set of **constraints** on the decision variables.

Example

- Find three digits that sum to 23.
- We want to model this problem as a CSP.
- So we must choose appropriate **variables**, **domains**, and **constraints**.

Decision Variables & Domains: Viewpoints

- A decision variable corresponds to a **choice** that must be made in solving a problem.
- **Values** in the domain of a decision variable correspond to the various **options** for this choice.
- A decision variable is **assigned** a value from its domain.
  - Equivalently, the choice associated with that variable is made.
- **A viewpoint**: a set of variables and domains sufficient to characterise the problem.

Example

- Find three digits that sum to 23.
- Decision variables: x_1, x_2, x_3.
- Domains: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}.

Constraints

- A constraint has a **scope**:
  - The subset of the decision variables it involves.
  - The **arity** of the constraint is the cardinality of this subset.
- Of the possible combinations of assignments to the variables in its scope, a constraint specifies:
  - Which are allowed.
    Assignments that **satisfy** the constraint.
  - Which are disallowed.
    Assignments that **violate** the constraint.

Example

- Find three digits that sum to 23.
- Decision variables: x_1, x_2, x_3.
- Domains: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}.
- Constraints: $x_1 + x_2 + x_3 = 23$

Thinking Abstractly

Recognising & Exploiting Patterns
in Constraint Problems

Thinking Abstractly

- When viewed **abstractly**, many combinatorial problems that we wish to tackle with constraint solving exhibit **common features**.
- Abstractly: **above** the level at which constraint modelling decisions are made.

Thinking Abstractly

- Example: Many problems require us to find combinatorial objects such as:
  - (Multi-)sets
  - Relations
  - Functions
- Typically, these are **not** supported directly by constraint solvers.
- So we need to **model** them as constrained collections of more primitive objects.

Exploiting Patterns

- By:
  - Recognising these commonly-occurring patterns, and
  - Developing corresponding **modelling patterns** for representing and constraining these combinatorial objects,

  we can **reduce effort** required when modelling a new problem.

Overview

- We will look at a number of individual patterns.
- We will then look at how these patterns can be **combined** to model more complex problems.

Sequences

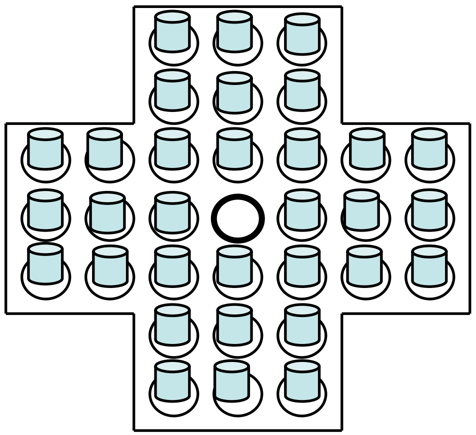
Sequences

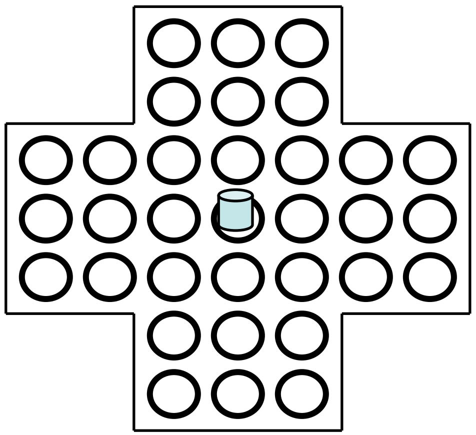
- A sequence is an **ordered list of elements**.
  - In the sense that a sequence has a first element, a second element, etc.
  - Repetition is allowed.
- Examples:
  - 0, 1, 1, 2, 3, 5, 8,13.
  - Turn right, drive ¼ mile, turn right, drive ½ mile, turn left.

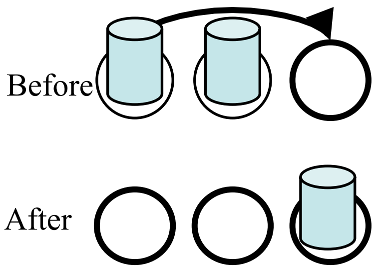
Where does the Sequence Pattern Occur?

www.csplib.org

- Planning Problems:
  - Find a **sequence of actions** to transform an initial state into a goal state.
  - Example: Peg Solitaire (CSPLib 38).







Where does the Sequence Pattern Occur?

- Scheduling Problems:
  - Travelling Salesperson

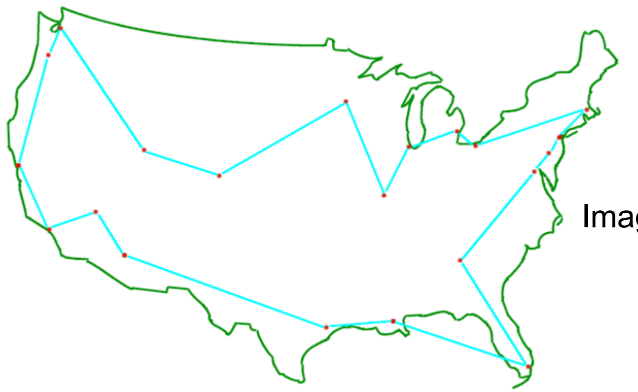


Image from www.jimloy.com

  - Car Sequencing (see CSPLib 1).

Where does the Sequence Pattern Occur?

- Communications:
  - Low Autocorrelation Binary Sequences (CSPLib 5).
- Mathematics:
  - Langford’s Problem (CSPLib 24).
  - Error-Correcting Codes (CSPLib 36).
- Puzzles:
  - Magic Sequences (CSPLib 19).

Fixed-length Sequences

- Problems of the form:
  - Given **n**,
  - Find a sequence of objects of length **n**,
  - Such that …

Fixed-length Sequences

- Example (Magic Sequence, CSPLib 19):
  - Given **n**.
  - Find a sequence **S** of integers s_0, …, s_n
  - Such that there are s_i occurrences of **i** in **S** for each **i** in 0, …, **n**.
  - If **n** = 9, a solution is:
    - 6, 2, 1, 0, 0, 0, 1, 0, 0, 0

Fixed-length Sequences

- Problems of the form:
  - Given **n**,
  - Find a sequence of objects of length **n**, such that …
- Most straightforward model: use an array of decision variables indexed 1..**n**. Domains are the objects to be found.
- Example, find a sequence of **n** digits:

DigitsArray	1	2	3	4		**n**
	0..9	0..9	0..9	0..9	…	0..9

Fixed-length Sequences

- Example (Magic Sequence, CSPLib 19):
  - Given **n**.
  - Find a sequence **S** of integers $s_0, \ldots, s_n$
  - Such that there are s_i occurrences of **i** in **S** for each **i** in 0, …, **n**.

MagicSequence	0	1	2	3		**n**
	0..**n**	0..**n**	0..**n**	0..**n**	…	0..**n**

Constraints:
Forall **i** in 0..n .
No. of occurrences of **i** in MagicSequence of **i** is MagicSequence[**i**]

Bounded-length Sequences

- Problems of the form:
  - Given **n**,
  - Find a sequence of objects of length **at most n**,
  - Such that …

Bounded-length Sequences

- Example (Kiselman Semigroup Problem):
  - Given **n**.
  - Find a sequence of integers drawn from 1..**n**
  - Such that between every pair of occurrences of an integer **i** there exists an integer greater than **i** and an integer less than **i**.
  - If **n** = 3, a solution is 2, 3, 1, 2
  - We are usually interested in counting the solutions for a given **n**.

Bounded-length Sequences

- Kiselman Semigroup Problem:
  - Given **n**.
  - Find a sequence of integers drawn from 1..**n**
  - Such that between every pair of occurrences of an integer **i** there exists an integer greater than **i** and an integer less than **i**.

Notice:
- There can be at most 1 occurrence of 1 and **n**.
- There can be at most 2 occurrences of 2 and **n**-1.
- There can be at most 4 occurrences of 3 and **n**-2.

So, given **n**, we can derive a **maximum sequence length**:
- For even **n**: $1+2+4+8+\ldots+2^{n/2-1} = 2^{n/2+1}-2$
- Similarly for odd **n**.

Bounded-length Sequences

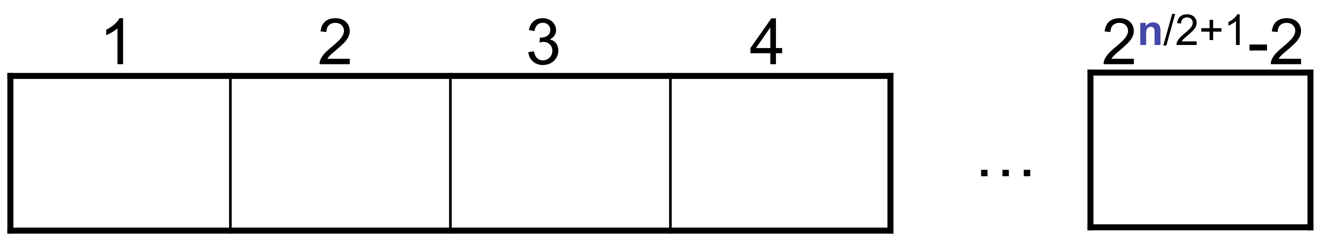
- Kiselman Semigroup Problem:
  - Given **n**.
  - Find a sequence of integers drawn from 1..**n**
  - Such that between every pair of occurrences of an integer **i** there exists an integer greater than **i** and an integer less than **i**.

Given **n**, we can derive a **maximum sequence length**:

- For even **n**: $1+2+4+8+\ldots+2^{n/2-1} = 2^{n/2+1}-2$
- Similarly for odd **n**.

Again, we can use an array indexed 1.. $2^{n/2+1}-2$:

KisSequence



Bounded-length Sequences

- Kiselman Semigroup Problem:
  - Given **n**.
  - Find a sequence of integers drawn from 1..**n**
  - Such that between every pair of occurrences of an integer **i** there exists an integer greater than **i** and an integer less than **i**.

Again, we can use an array indexed 1.. $2^{n/2+1}$-2:

KisSequence

1	2	3	4	…	$2^{n/2+1}-2$

Problem: What if a solution has length less than $2^{n/2+1}$-2?

Example: The empty sequence is always a solution to this problem.

Bounded-length Sequences

- Kiselman Semigroup Problem:
  - Given **n**.
  - Find a sequence of integers drawn from 1..**n**
  - Such that between every pair of occurrences of an integer **i** there exists an integer greater than **i** and an integer less than **i**.

Problem: What if a solution has length less than $2^{n/2+1}-2$?
Example: The empty sequence is always a solution to this problem.
Solution: Use a **dummy value** in the domain.

KisSequence

1	2	3	4		$2^{n/2+1}-2$
0..n	**0..n**	**0..n**	**0..n**	…	**0..n**

Bounded-length Sequences

- Kiselman Semigroup Problem:
  - Given **n**, find a sequence of integers drawn from 1..**n**
  - Such that between every pair of occurrences of an integer **i** there exists an integer greater than **i** and an integer less than **i**.

KisSequence

1	2	3	4	…	$2^{n/2+1}-2$
0..n	**0..n**	**0..n**	**0..n**	…	**0..n**

Constraints:

Forall i in 1.. $2^{n/2+1}-2$. Forall j in i+1 .. $2^{n/2+1}-2$.

If

(KisSequence[i] = KisSequence[j] $\neq$ 0)

Then exists k, l in i+1..j-1 .

(KisSequence[i] < KisSequence[k]) and

(KisSequence[i] > KisSequence[l])

Bounded-length Sequences

- So, for **n** = 4 and the solution 1, 2 the variables might be assigned:

KisSequence

1	2	3	4	5	6
1	**2**	**0**	**0**	**0**	**0**

Problem: They might also be assigned

KisSequence

1	2	3	4	5	6
1	**0**	**0**	**2**	**0**	**0**

Adding the dummy value has created **equivalence classes** of assignments

Bounded-length Sequences

- Adding the dummy value has created **equivalence classes** of assignments.
- Solution: choose a **canonical element** from each class.
- E.g. all 0s must appear at the end of the sequence:
  - Forall i in 1.. $2^{n/2+1}-3$.
    If (KisSequence[i] = 0) Then (KisSequence[i + 1] = 0)

KisSequence

1	2	3	4	5	6
1	2	0	0	0	0

KisSequence

1	2	3	4	5	6
1	0	0	2	0	0

Something to Note

- It is **very common** when modelling an abstract object to **introduce equivalences** during modelling.
- Need to be aware of this happening, and of the measure used to counter it.

Unbounded Sequences

- For the Kiselman problem, we were able to bound the sequence length (relatively) straightforwardly.
- For some problems either:
  - We cannot derive a bound.
  - Any bound we can derive is so weak as to be useless.

Unbounded Sequences

- For some problems either:
  - We cannot derive a bound.
  - Any bound we can derive is so weak as to be useless.
- This is often the case when modelling planning problems.
  - Difficult to tell how many actions are going to be needed to achieve the goal state.

Unbounded Sequences

- Solution: solve a series of CSPs, incrementally **increasing the length** of the sequence.
- i.e. Try and find a solution for a sequence of length 1.
  - If no solution, try length 2.
  - If no solution, try length 3 …

Permutations

- Some problems involve finding a sequence of elements where:
  - The elements in the sequence are known
  - Their arrangement is not.
- I.e. find a **permutation** of the sequence.
- E.g. The Travelling Salesman Problem
  - Given a network of cities, known distances between every pair of cities, and a starting city.
  - Find shortest route that visits all points, returns to start.

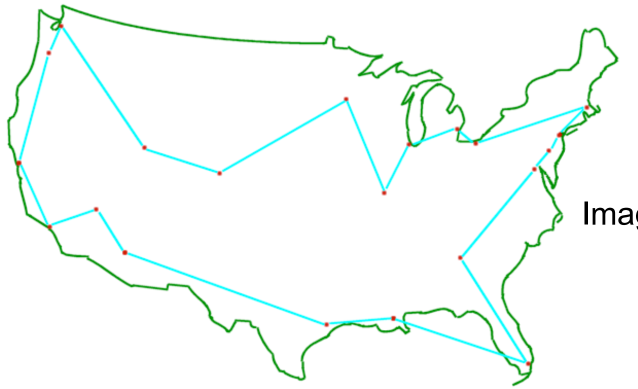


Image from www.jimloy.com

Permutations: First Viewpoint

- Assume that the elements of the permutation are distinct.
- First viewpoint is as fixed-length.
  If permutation contains elements a, …, f:

Perm1	1	2	3	4	5	6
	a..f	a..f	a..f	a..f	a..f	a..f

Constraint: All-different(Perm1)

Permutations: Second Viewpoint

- Alternatively, we know the elements that appear in the sequence.
- So we can index by those elements:

Perm2

a	b	c	d	e	f
1..6	1..6	1..6	1..6	1..6	1..6

Constraint: All-different(Perm2)

- Domain values represent the position in the sequence an element is in. So "badcef" would be:

Perm2

a	b	c	d	e	f
2	1	4	3	5	6

Permutations: Which Viewpoint to Choose?

- Depends on the constraints on the permutation.
- Example: a and b must be adjacent.

Perm1	1	2	3	4	5	6
	a..f	a..f	a..f	a..f	a..f	a..f

If Perm1[1] = a Then Perm1[2] = b
If Perm1[6] = a Then Perm1[5] = b
Forall i in 2..5 . If Perm1[i] = a Then
Perm1[i-1] = b or Perm1[i+1] = b

(and vice versa)

Permutations: Which Viewpoint to Choose?

- Depends on the constraints on the permutation.
- Example: a and b must be adjacent.

Perm2

a	b	c	d	e	f
1..6	1..6	1..6	1..6	1..6	1..6

$| \text{Perm2}[a] - \text{Perm2}[b] | = 1$

Permutations: Which Viewpoint to Choose?

- Depends on the constraints on the permutation.
- Example: The first three letters of the sequence must form an English word.

	1	2	3	4	5	6
Perm1	a..f	a..f	a..f	a..f	a..f	a..f

Just need a table constraint on the first three variables in Perm1 that allows “bad”, “cad”, “fad”, …

Permutations: Which Viewpoint to Choose?

- Depends on the constraints on the permutation.
- Example: The first three letters of the sequence must form an English word.

Perm2	a	b	c	d	e	f
	1..6	1..6	1..6	1..6	1..6	1..6

Horrible: (Perm2[a] = 1 and Perm2[c] = 2 and Perm2[e] = 3)
or …

Sequences: Summary

- Fixed-length.
- Bounded-length.
- Unbounded.
- Permutations.
- Try some of the problems from CSPLib!

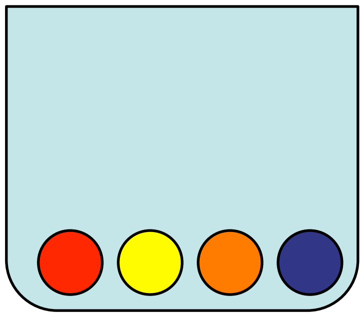
Sets

Sets

- A collection of distinct objects.
  - {1, 2, 3}.
  - {red, green, blue}.
- Not arranged in any particular order.
- Yes, I know: many solvers support set variables.
  - Some don't (e.g. Minion).
  - This pattern is still very useful when considering **combinations** of objects.

Where does the Set Pattern Occur?

- Packing Problems:
  - Can often represent a container (e.g. a bin) as a set of objects.

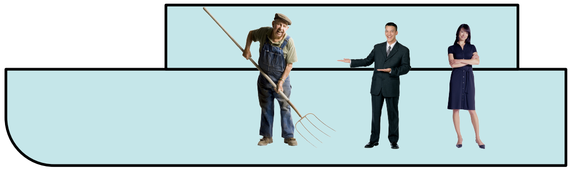


See also:

- Steel Mill Slab Design (CSPLib 38)
- Rack Configuration (CSPLib 31)

Where does the Set Pattern Occur?

- Scheduling:
  - E.g. Progressive Party Problem (CSPLib 13)
  - Timetable a party at a yacht club.
  - Certain boats designated hosts, crews of remaining boats in turn visit the host boats for successive half-hour periods.
  - View a boat as a **set** of crews.



See also: Social Golfers (CSPLib 10)

Where does the Set Pattern Occur?

- Mathematics:
  - E.g. Steiner Triple Systems (CSPLib 44)
  - Given n, find a set of n(n-1)/6 triples of elements from 1,…,n such that any pair of triples have at most one common element.
  - If n = 7:
    - {{1, 2, 3}, {1, 4, 5}, {1, 6, 7}, {2, 4, 6}, {2, 5, 7}, 3, 4, 7}, {3, 5, 6}}
  - This is a **set of sets** (the triples).

See also: Golomb Ruler (CSPLib 6).

Fixed-cardinality Sets

Consider the following simple problem class:

- Given **n** and **s**, Find a **set** of **n** digits that sum to **s**.
- To model this problem, we need to decide how to represent this set.
- We will look at two different ways (there are many more):
  - The Explicit representation.
  - The Occurrence representation.

Fixed-cardinality Sets: Explicit Representation

- Given **n** and **s**, Find a **set** of **n** digits that sum to **s**.
- Introduce **E**, an array of decision variables indexed by 1..**n**. Domain of each is 0..9:

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

- Constraints:

AllDifferent(**E**).

Sum(**E**) = **s**.

Fixed-cardinality Sets: Explicit Representation

- So the set {1, 3, 5, 7} might be represented:

	1	2	3	4
E	1	3	5	7

- However, it might also be represented:

	1	2	3	4
E	7	3	5	1

- Once again, a modelling step has introduced **equivalence classes** of assignments.

Fixed-cardinality Sets: Explicit Representation

- So the set {1, 3, 5, 7} might be represented:

	1	2	3	4
E	1	3	5	7

	1	2	3	4
E	7	3	5	1

- Again we need to choose a **canonical element** from each class.
- Obvious choice is to require **ascending order**.
- So, we can replace AllDifferent(E) with E[1] < E[2] < …

Fixed-cardinality Sets: Occurrence Representation

- Given **n** and **s**, Find a **set** of **n** digits that sum to **s**.
- Introduce **O**, an array of 0/1 decision variables indexed by 0..9:

	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

- Constraints:

$$\text{Sum}(\mathbf{E}) = \mathbf{n}.$$

$$\mathbf{O}[1] + 2\mathbf{O}[2] + 3\mathbf{O}[3] + \ldots + 9\mathbf{O}[9] = \mathbf{s}.$$

Notice: This representation did **not** introduce equivalence classes.

Explicit vs. Occurrence Representations

- What if we want to say:
  "If 5 is in the set then so is 4"?
- Explicitly:

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

Forall j in 1..n .
  If (E[j] = 5) Exists i in 1..j . E[i] = 4

Notice how I exploit ascending order here

Explicit vs. Occurrence Representations

What if we want to say:

"If 5 is in the set then so is 4"?

Occurrence Rep:

	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

If **O**[5] = 1 Then **O**[4] = 1

Explicit vs. Occurrence Representations

- What if we want to say:
  - “The difference between every pair of elements is not equal to the assignment to a variable **d**”?
- Explicitly:

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

Forall i in 1..**n** . Forall j in i+1..**n** .

E[j] - **E**[i] $\neq$ **d**

Notice how I exploit ascending order here

Explicit vs. Occurrence Representations

- What if we want to say:
  - "The difference between every pair of elements is not equal to the assignment to a variable **d**"?
- Occurrence Rep:

	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

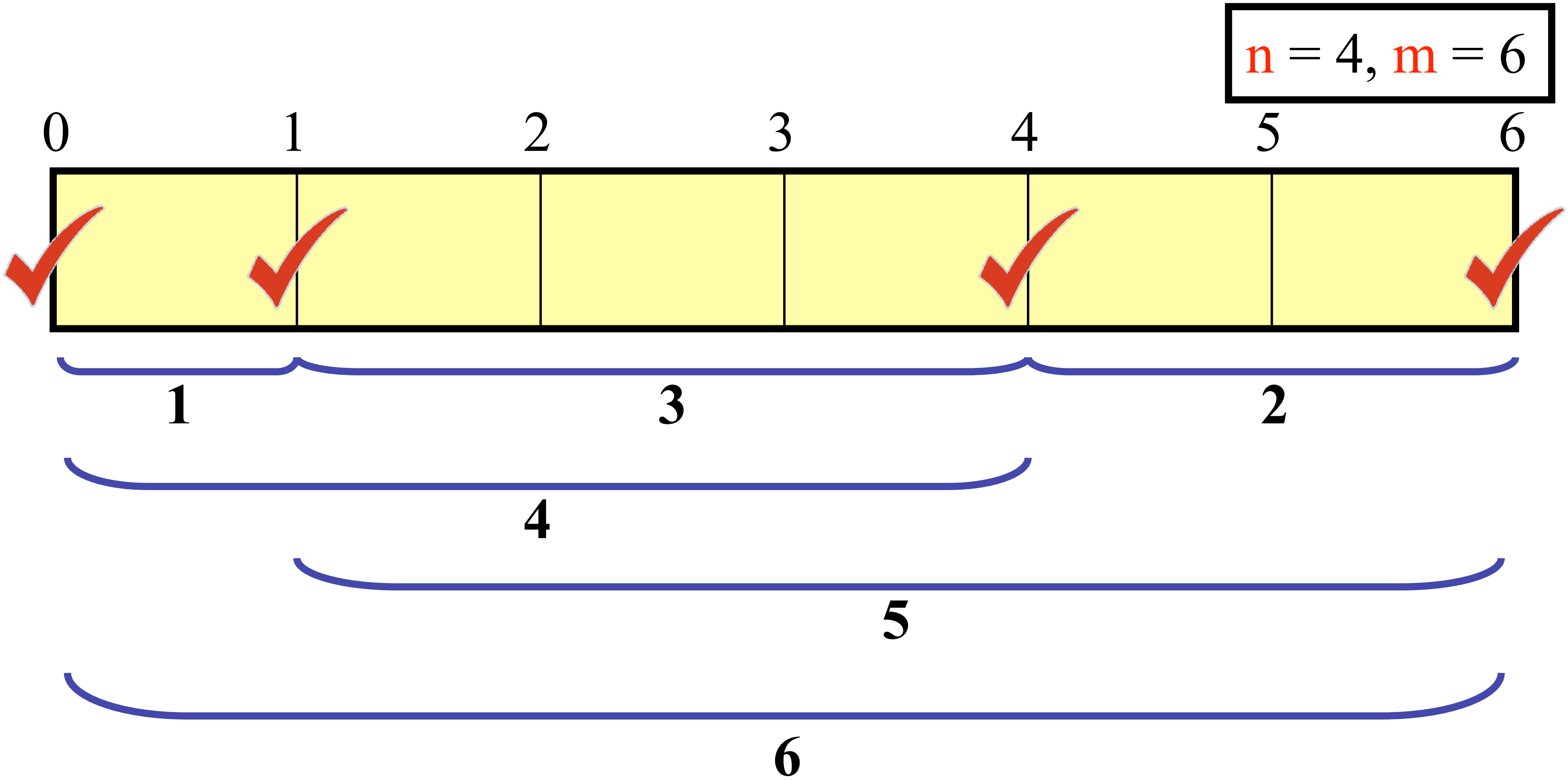
Add constraints of the form:

- **d** = 1 Then $((\mathbf{O}[0] \times \mathbf{O}[1]) + (\mathbf{O}[1] \times \mathbf{O}[2]) + \ldots = 0)$
- **d** = 2 Then …

The Golomb Ruler Problem

- Applications: x-ray crystallography, radio antenna placement.
- Given:
  - A positive integer ***n***.
- Find:
  - A set of ***n*** integer ticks on a ruler of length ***m***.
- Such that:
  - All inter-tick distances are distinct.
- Minimising:
  - ***m***.

Golomb Ruler: Example



Modelling the Golomb Ruler

- Requires finding a **set** of ticks.
- Which of the two representations shall we use?

- The constraints need direct access to the values in the set: let's try the **explicit** representation.

	1	2	3	4		n
T	$0..n^2$	$0..n^2$	$0..n^2$	$0..n^2$	…	$0..n^2$

Ascending order: T[1] < T[2] < … < T[n]

Modelling the Golomb Ruler

	1	2	3	4		n
T	$0..n^2$	$0..n^2$	$0..n^2$	$0..n^2$	…	$0..n^2$

- All inter-tick distances are distinct:
  - $\mathbf{T}[j] - \mathbf{T}[i] \neq \mathbf{T}[k] - \mathbf{T}[l]$
    for each $\{i, j\}$, $\{k, l\}$ drawn from $1..\mathbf{n}$,
    such that $\{i, j\} \neq \{k, l\}$, $i < j$, $k < l$
    again,exploiting ascending order.

Modelling the Golomb Ruler

	1	2	3	4		n
T	0..n^2	0..n^2	0..n^2	0..n^2	…	0..n^2

- Objective:
  - Minimise(T[n])

    Again, exploiting ascending order.

Modelling the Golomb Ruler

- **A Challenge**:
- Can you see how to model this problem using the occurrence representation?

Bounded-cardinality Sets: Occurrence

- Given **n** and **s**, Find a **set** of **at most n** digits that sum to **s**.

	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

- Constraints:

$$\text{Sum}(\mathbf{E}) \leq \mathbf{n}.$$

$$\mathbf{O}[1] + 2\mathbf{O}[2] + 3\mathbf{O}[3] + \ldots + 9\mathbf{O}[9] = \mathbf{s}.$$

Bounded-cardinality Sets: Explicit

- Given **n** and **s**, Find a **set** of **at most n** digits that sum to **s**.
- Explicit?

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

- Constraints:

  E in ascending order.

Constraints on Sets

- Sets appear very frequently in problems we wish to model.
- It's worth looking at how to model some other common constraints on them:
  - Intersection
  - Union
  - Subset…

Set Intersection: Occurrence Rep

- Model **A** ∩ **B** = **C**.
  - **A**, **B** are sets of digits (cardinality 5).

	0	1	2	3	4	5	6	7	8	9	
$\mathbf{O_A}$	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	$\mathrm{Sum}(\mathbf{O_A}) = 5$

	0	1	2	3	4	5	6	7	8	9	
$\mathbf{O_B}$	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	$\mathrm{Sum}(\mathbf{O_B}) = 5$

	0	1	2	3	4	5	6	7	8	9	
$\mathbf{O_C}$	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	Sum?

Constraint: $\mathbf{O_C}[i] = \mathbf{O_A}[i] \times \mathbf{O_B}[i]$ (for each i in 0..9)

Set Intersection: Explicit Rep

- Model **A** ∩ **B** = **C**.
  - **A**, **B** are sets of digits (cardinality 5).

	1	2	3	4	5
E_A	0..9	0..9	0..9	0..9	0..9

	1	2	3	4	5
E_B	0..9	0..9	0..9	0..9	0..9

	1	2	3	4	5
E_C	0..9	0..9	0..9	0..9	0..9

	1	2	3	4	5
S_C	0, 1	0, 1	0, 1	0, 1	0, 1

E_A, E_B, E_C in ascending order.

What does the intersection constraint look like?

Set Union: Occurrence Rep

- Model **A** ∪ **B** = **C**.
  - **A**, **B** are sets of digits (cardinality 5).

O_A

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

$\mathrm{Sum}(O_A) = 5$

O_B

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

$\mathrm{Sum}(O_B) = 5$

O_C

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

Sum?

Constraint:

$O_C[i] = (O_A[i] = 1 \lor O_B[i] = 1)$ (for each i in 0..9)

OR

$O_C[i] = \max(O_A[i], O_B[i])$ (for each i in 0..9)

Set Union: Explicit Rep

- Model **A** ∪ **B** = **C**.
  - **A**, **B** are sets of digits (cardinality 5).

E_A

1	2	3	4	5
0..9	0..9	0..9	0..9	0..9

E_A, E_B, E_C in ascending order.

E_B

1	2	3	4	5
0..9	0..9	0..9	0..9	0..9

E_C

1	2	3	4	5	6	7	8	9	10
0..9	0..9	0..9	0..9	0..9	0..9	0..9	0..9	0..9	0..9

S_C

1	2	3	4	5	6	7	8	9	10
0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1

What does the union constraint look like?

Subset: Occurrence Rep

- Model **A** $\subseteq$ **B**.
  - **B** is a set of digits (cardinality 5).

	0	1	2	3	4	5	6	7	8	9	
$\mathbf{O_B}$	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	$\mathrm{Sum}(\mathbf{O_B}) = 5$

	0	1	2	3	4	5	6	7	8	9	
$\mathbf{O_A}$	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	Sum?

Subset constraint: $\mathbf{O_A}[i] \leq \mathbf{O_B}[i]$ (foreach i in 0..9)

Subset: Explicit Rep

- Model **A** ⊆ **B**.
  - **B** is a set of digits (cardinality 5).

E_A, E_B in ascending order.

	1	2	3	4	5
E_B	0..9	0..9	0..9	0..9	0..9

	1	2	3	4	5
E_A	0..9	0..9	0..9	0..9	0..9

	1	2	3	4	5
S_A	0, 1	0, 1	0, 1	0, 1	0, 1

What does the subset constraint look like?

Combined Representations

- Obviously, can have combinations of the two representations (e.g. O_A, O_B and E_C).
- Constraints on original sets have to be modelled appropriately.

Sets: Summary

- We’ve seen that the explicit representation often looks much worse when the set is of bounded cardinality.
- The explicit representation will make a comeback when we look at **nested** objects (e.g. sets of sets).

Multisets

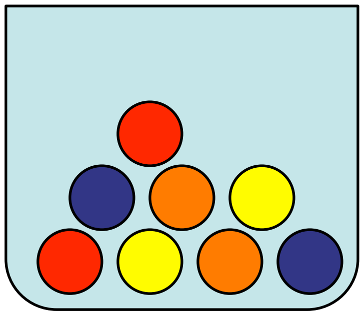
(in brief)

Multisets

- A collection of distinct objects.
- Repetition allowed.
- Not arranged in any particular order.
  - {1, 2, 3, 1}.
  - {red, green, blue, green, red}.

Where does the Multiset Pattern Occur?

- In many places that the set pattern occurs. For example, in some packing problems (e.g. Vellino's problem) it is appropriate to view the containers as multisets:



Multisets: Explicit

- Find a multiset of n/at most n digits:
- Explicit fixed-cardinality:

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

- Bounded-cardinality:

No AllDiff needed

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

	1	2	3	4		n
Switches	0,1	0,1	0,1	0,1	…	0,1

Are we introducing **equivalent assignments** here?

Multisets: Occurrence

- Find a multiset of **n**/at most **n** digits:
- Occurrence representation:

	0	1	2	3	4	5	6	7	8	9
O	0…**n**	0…**n**	0…**n**	0…**n**	0…**n**	0…**n**	0…**n**	0…**n**	0…**n**	0…**n**

- Larger domain allows multiple occurrences.
- Fixed-cardinality **n**: Sum is **n**.
- Bounded-cardinality **n**: Sum is at most **n**.

Relations

Relations

- Assign truth values to tuples of values.
  - Constraints are relations.
- Example:
  - Set **P** is {Bill, Bert, Tom}.
  - Binary relation likes on **P** x **P** might assign <Bill, Bert> and <Bert, Tom> true (to mean Bill likes Bert and Bert likes Tom), and false to the other combinations.

Where Does the Relation Pattern Occur?

- Combinatorial Design:
  - BIBDs (CSPLib 28)
- Cellular Frequency Assignment
  - Assignment of frequencies to transmitters (see Van Hentenryck '99)
- Rostering
  - Assignment of staff to shifts.

Relations: Occurrence Representation

- Find a relation R between sets
  A = {1, 2, 3} and B = {2, 3, 4}
  such that each element of A is related to at least one element of B."

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

Relations: Projection

- A common operation on relations. We project a relation onto one or more of its arguments. Result: a relation of reduced arity.
- Example:
  - Set **P** is {Bill, Bert, Tom}.
  - Relation likes on **P** x **P** = {<Bill, Bert>, <Bert, Tom>, <Bert, Bill>}.
  - Project likes onto “Bert” (first position): {Tom, Bill}

Relations: Projection

- Find a relation R between sets
  A = {1, 2, 3} and B = {2, 3, 4}
  such that **each element of A is related to at least one element of B**."
- So projection of R onto each element of A has size at least 1.

A

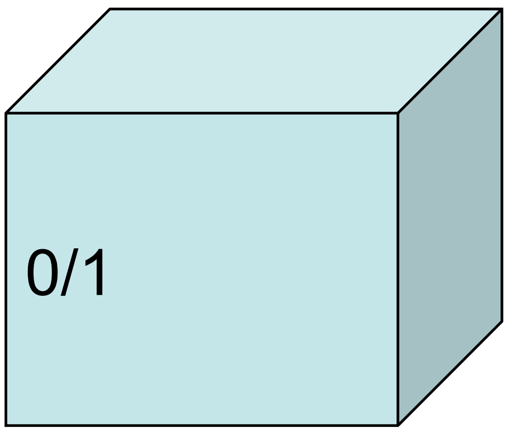
B		1	2	3
	2	0, 1	0, 1	0, 1
B	3	0, 1	0, 1	0, 1
	4	0, 1	0, 1	0, 1

Constraint: Summation on the columns

Relations: Occurrence Representation

- What about *k*-ary relations?

k-dimensional matrices



Relations: Other Representations

- Consider binary case, A × B.
  (A = {1, 2, 3}, B = {2, 3, 4}).
- Introduce a matrix indexed by elements of A and by 1 .. |B|:

Why did I add 0?

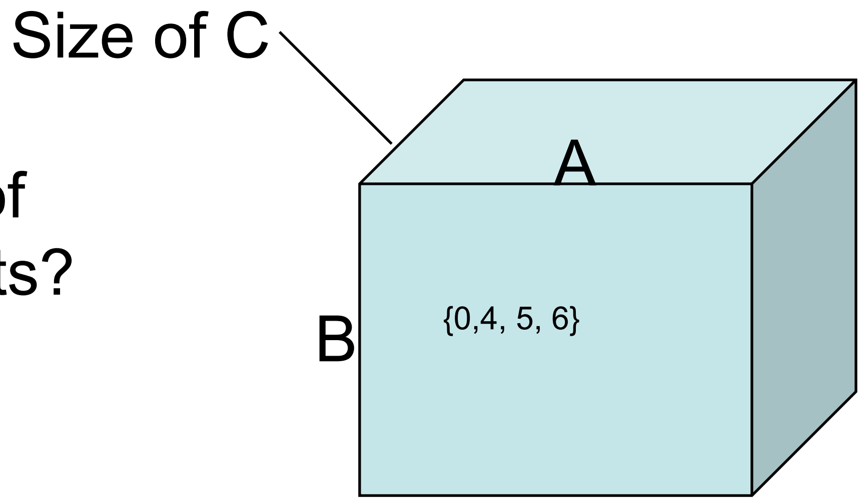
Do I need to add constraints?

		A		
		1	2	3
	1	{0,2,3,4}	{0,2,3,4}	{0,2,3,4}
1..|B|	2	{0,2,3,4}	{0,2,3,4}	{0,2,3,4}
	3	{0,2,3,4}	{0,2,3,4}	{0,2,3,4}

Relations: Other Representations

- Consider ternary case, $A \times B \times C$.
  ($A = \{1, 2, 3\}$, $B = \{2, 3, 4\}$, $C = \{4, 5, 6\}$).
- Introduce a 3d matrix indexed by elements of A and B, and size of C. Domain is C and 0.

Can you think of
Other viewpoints?



Would a 2d array indexed by A and B, with C as domain work?

Relations: Other Representations

- We will return to the natural “explicit” representation of a relation when considering modelling nested combinatorial objects.

BIBD: Specification

- Given:
  - A 5-tuple of positive integers: $\langle \boldsymbol{v}, \boldsymbol{b}, \boldsymbol{r}, \boldsymbol{k}, \boldsymbol{\lambda} \rangle$.
- Find:
  - An assignment, associating each of ***v*** objects to ***b*** blocks.
- Such that:
  - Each block contains ***k*** distinct objects.
  - Each object occurs in exactly ***r*** different blocks.
  - Every two distinct objects occur together in exactly λ blocks.

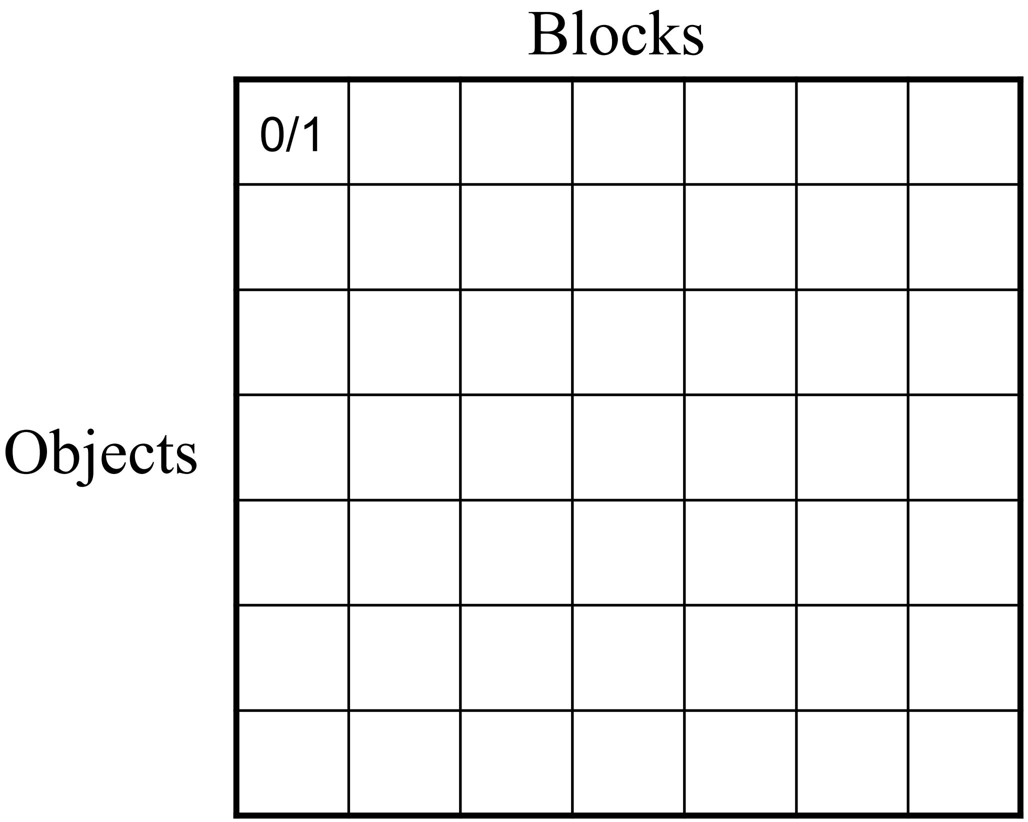
Applications: cryptography, experimental design.

Modelling the BIBD

- An abstract view of the decision variable:
  - A **relation** on **blocks** × **objects**.

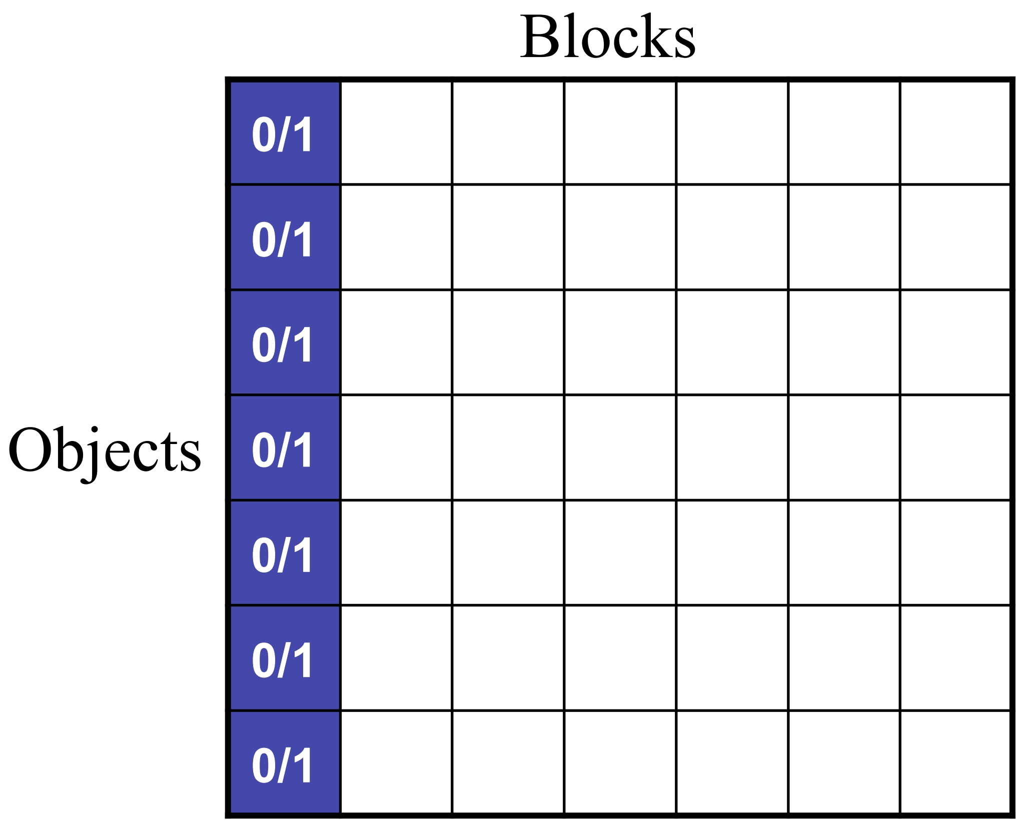
BIBD: Occurrence Model

- Constraints can now be stated easily on **rows** and **columns**.



BIBD: Occurrence Model

- Each block contains ***k*** distinct objects.
  - Foreach ***i*** in **blocks**, sum of ***i***th column is ***k***.
- So each instance has ***b*** sum constraints.



BIBD: Occurrence Model

- Each object occurs in exactly ***r*** different blocks.
  - Foreach ***i*** in **objects**, sum of ***i***th row is ***r***.
  - Again, each instance has ***v*** sum constraints.

Blocks

	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Objects							

BIBD: Occurrence Model

- Every 2 distinct objects occur together λ blocks.
    - Foreach {***i***, ***j***} in **objects**, scalar product of ***i***th and ***j***th rows is λ.

Blocks

Objects

0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1

BIBD: Example

- $\langle 7, 7, 3, 3, 1 \rangle$

Blocks

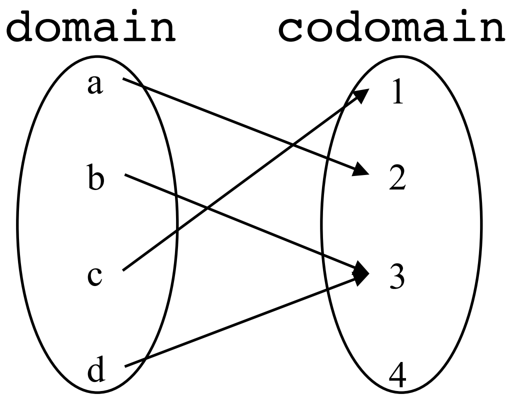
	0	0	0	0	1	1	1
	0	0	1	1	0	0	1
	0	1	0	1	0	1	0
Objects	0	1	1	0	1	0	0
	1	0	0	1	1	0	0
	1	0	1	0	0	1	0
	1	1	0	0	0	0	1

… but did we introduce **equivalence classes** of assignments here?

Functions

Functions

- A function *f* is a binary relation on two sets:
  - a **`domain`** and a **`codomain`**.
- Has the property that each element of the `domain` is related to at most one element of the `codomain`:
  - its **image**.
- We write *f*(x) = y to mean that the image of x under the function *f* is y.



Where does the Function Pattern Occur?

- Timetabling:
  - E.g. Balanced Academic Curriculum Problem (CSPLib 30).
  - Find a function from courses to periods in the timetable.

Where does the Function Pattern Occur?

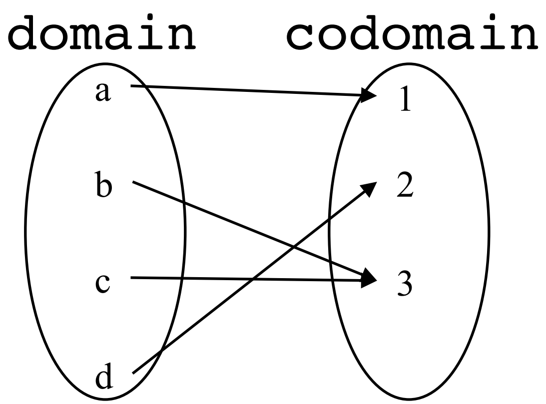
- Graph Colouring:
  - Find a function from the vertices of a graph to a set of colours

Where does the Function Pattern Occur?

- Warehouse Location.
  - Find a function from stores to warehouses to indicate which store is supplied by which warehouse.

Total Functions: Explicit

- In a **total** function, every element of the `domain` has an image in the `codomain`.



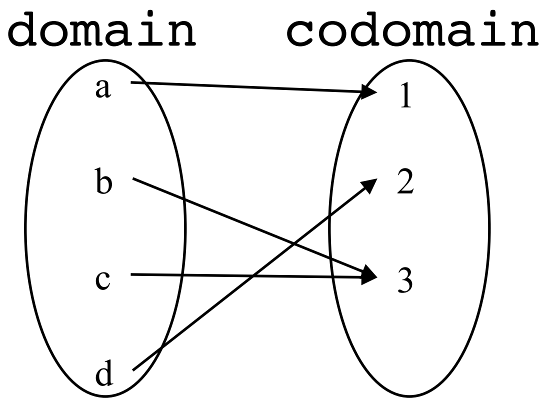
- Use a variable per element of the `domain`, domain of each is the `codomain`.

TotalFn

a	b	c	d
1,2,3	1,2,3	1,2,3	1,2,3

Total Functions: Occurrence

- In a **total** function, every element of the `domain` has an image in the `codomain`.



- Use a 2d array of 0/1 variables, indexed by the `domain` and `codomain`.

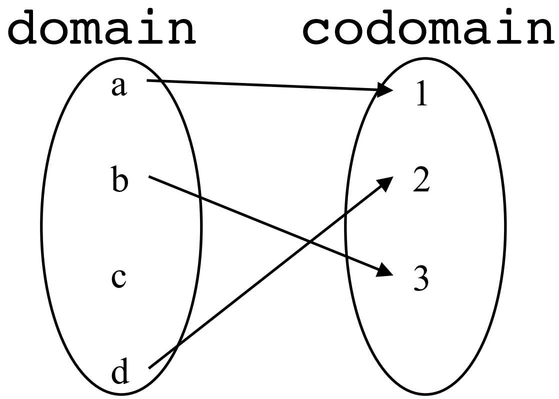
TotalFn

	a	b	c	d
1	0/1	0/1	0/1	0/1
2	0/1	0/1	0/1	0/1
3	0/1	0/1	0/1	0/1

Sum of each col is 1

Partial Functions: Explcit

- In a **partial** function, some elements of the `domain` have no image in the `codomain`.



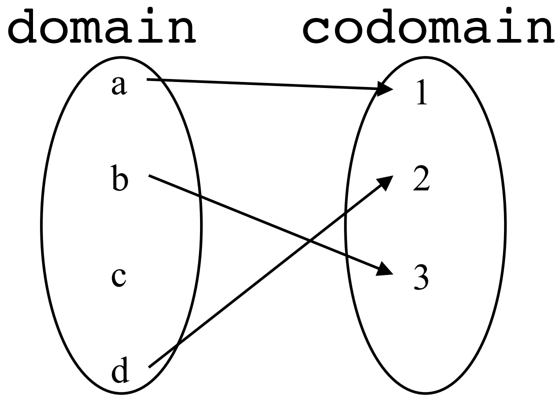
- Use a variable per element of the `domain`, domain of each is the `codomain` **and** a dummy element:

PartialFn

a	b	c	d
0,1,2,3	0,1,2,3	0,1,2,3	0,1,2,3

Partial Functions: Occurrence

- In a **partial** function, some elements of the `domain` have no image in the `codomain`.



- Use a 2d array of 0/1 variables, indexed by the `domain` and `codomain`.

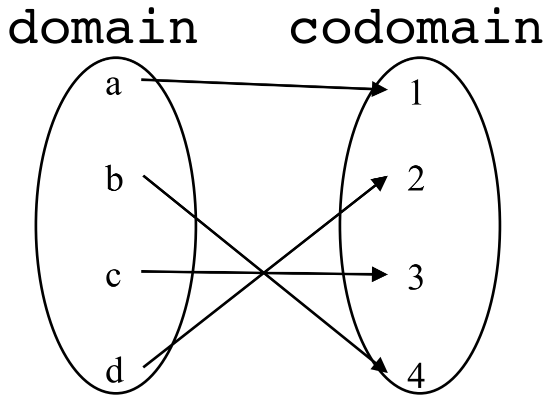
PartialFn

	a	b	c	d
1	0/1	0/1	0/1	0/1
2	0/1	0/1	0/1	0/1
3	0/1	0/1	0/1	0/1

Sum of each col **≤ 1**

Injections: Explicit

- The images of two distinct elements of the `domain` under an **injective** function are distinct

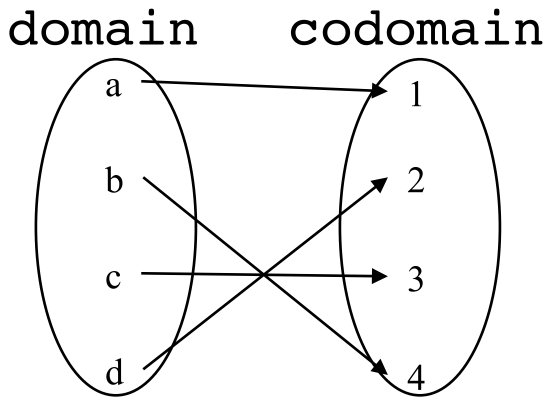


- If **total**, just need to add a constraint to our basic model of a function: allDifferent(injection)

Injection	a	b	c	d
	1,2,3,4	1,2,3,4	1,2,3,4	1,2,3,4

Injections: Occurrence

- The images of two distinct elements of the `domain` under an **injective** function are distinct



- If **total**, just need to add constraints to our basic model of a function:

Injection

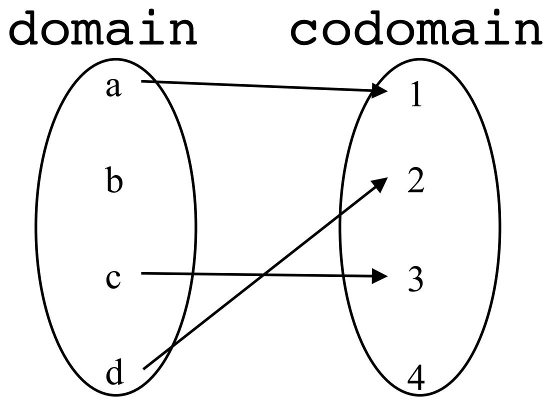
	a	b	c	d
1	0/1	0/1	0/1	0/1
2	0/1	0/1	0/1	0/1
3	0/1	0/1	0/1	0/1
4	0/1	0/1	0/1	0/1

Sum of each col: **1**

Sum of each row: **≤1**

Partial Injections

- The images of two distinct elements of the `domain` under an **injective** function are distinct



- If **partial**, explicit model is messy, but 0/1 model is easy:

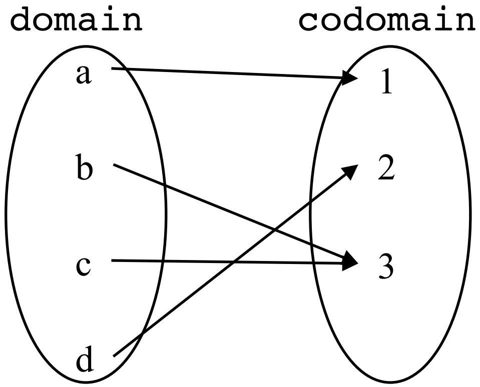
Injection

	a	b	c	d
1	0/1	0/1	0/1	0/1
2	0/1	0/1	0/1	0/1
3	0/1	0/1	0/1	0/1
4	0/1	0/1	0/1	0/1

Sum of each col: **≤1**
Sum of each row: **≤1**

Surjections: Explicit

- A function is **surjective** if every element of the `codomain` is the image of some element of the `domain`.



- Can modify our explicit total/partial models of a function to model surjections:

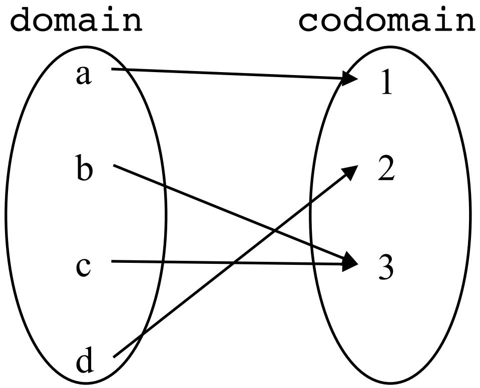
Surjection

a	b	c	d
1,2,3	1,2,3	1,2,3	1,2,3

Must exist indices where each of 1, 2, 3 appear.

Surjections: Occurrence

- A function is **surjective** if every element of the `codomain` is the image of some element of the `domain`.



- Can modify our 0/1 total/partial models of a function to model surjections easily:

Surjection

	a	b	c	d
1	0/1	0/1	0/1	0/1
2	0/1	0/1	0/1	0/1
3	0/1	0/1	0/1	0/1

Sum of each row: **≥ 1**

Modelling Bijections

- A **bijection** is both an **injection** and a **surjection**.
- In fact, we've seen this already when looking at permutations.
- Can you formulate a 0/1 model?

Summary

- There are a number of patterns prevalent in many combinatorial problems.
- We’ve seen some of these and some alternative ways of modelling them.
- You can invoke these patterns when modelling a new problem.
- Beware of introducing equivalence classes of assignments, and the steps needed to avoid this.