Combinatorial Optimisation using Constraint Programming
(course 1DL441, 10 credits)

Solving Problems that Are Larger than the Universe!

Pierre Flener

ASTRA Research Group
on Combinatorial Optimisation
Uppsala University
Sweden

Autumn semester, periods 1 and 2
(Version of 29th September 2017)
Optimisation is a science of service: to scientists, to engineers, to artists, and to society.
### Example (Agricultural experiment design)

<table>
<thead>
<tr>
<th></th>
<th>plot1</th>
<th>plot2</th>
<th>plot3</th>
<th>plot4</th>
<th>plot5</th>
<th>plot6</th>
<th>plot7</th>
</tr>
</thead>
<tbody>
<tr>
<td>barley</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>corn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>millet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>oats</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rye</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spelt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wheat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Constraints to be satisfied:**

1. **Equal sample size:** Every grain is grown in 3 plots.
2. **Equal growth load:** Every plot grows 3 grains.
3. **Balance:** Every grain pair is grown in 1 common plot.

**Instance:** 7 grains, 7 plots, 3 plots/grain, 3 grains/plot, balance 1.
Example (Agricultural experiment design)

<table>
<thead>
<tr>
<th></th>
<th>plot1</th>
<th>plot2</th>
<th>plot3</th>
<th>plot4</th>
<th>plot5</th>
<th>plot6</th>
<th>plot7</th>
</tr>
</thead>
<tbody>
<tr>
<td>barley</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>corn</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>millet</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>oats</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>rye</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>spelt</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>wheat</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
</tbody>
</table>

**Constraints to be satisfied:**

1. Equal sample size: Every grain is grown in 3 plots.
2. Equal growth load: Every plot grows 3 grains.
3. Balance: Every grain pair is grown in 1 common plot.

**Instance:** 7 grains, 7 plots, 3 plots/grain, 3 grains/plot, balance 1.
Example (Doctor rostering)

<table>
<thead>
<tr>
<th>Doctor</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doctor A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Doctor B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Doctor C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Doctor D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Doctor E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Constraints to be satisfied:**

1. #doctors-on-call / day = 1
2. #operations / workday ≤ 2
3. #operations / week ≥ 7
4. #appointments / week ≥ 4
5. Day off after operation day
6. ...

**Objective function to be minimised:**

- Cost: ...
Example (Doctor rostering)

<table>
<thead>
<tr>
<th>Doctor</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doctor A</td>
<td>call</td>
<td>–</td>
<td>oper</td>
<td>–</td>
<td>oper</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Doctor B</td>
<td>app</td>
<td>call</td>
<td>–</td>
<td>oper</td>
<td>–</td>
<td>–</td>
<td>call</td>
</tr>
<tr>
<td>Doctor C</td>
<td>oper</td>
<td>–</td>
<td>call</td>
<td>app</td>
<td>app</td>
<td>call</td>
<td>–</td>
</tr>
<tr>
<td>Doctor D</td>
<td>app</td>
<td>oper</td>
<td>–</td>
<td>call</td>
<td>oper</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Doctor E</td>
<td>oper</td>
<td>–</td>
<td>oper</td>
<td>–</td>
<td>call</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

**Constraints to be satisfied:**

1. \#doctors-on-call / day $= 1$
2. \#operations / workday $\leq 2$
3. \#operations / week $\geq 7$
4. \#appointments / week $\geq 4$
5. Day off after operation day
6. ...

**Objective function to be minimised:**

- Cost: ...
### Example (Financial investment instrument design)

<table>
<thead>
<tr>
<th></th>
<th>Acer</th>
<th>Apple</th>
<th>Dell</th>
<th>HP</th>
<th>IBM</th>
<th>Sony</th>
<th>Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basket 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basket 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basket 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basket 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basket 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basket 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basket 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Constraint to be satisfied:**

1. Equal basket size: Every basket contains 3 shares.

**Objective function to be minimised:**

- Risk: Maximum observed overlap of any basket pair.
**Example (Financial investment instrument design)**

<table>
<thead>
<tr>
<th>Basket</th>
<th>Acer</th>
<th>Apple</th>
<th>Dell</th>
<th>HP</th>
<th>IBM</th>
<th>Sony</th>
<th>Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>✓</td>
<td>—</td>
</tr>
<tr>
<td>5</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>—</td>
<td>—</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>7</td>
<td>—</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>✓</td>
<td>—</td>
</tr>
</tbody>
</table>

**Constraint** to be **satisfied**:

1. Equal basket size: Every basket contains 3 shares.

**Objective function** to be **minimised**:

- Risk: Maximum observed overlap of any basket pair.
Example (Vehicle routing: parcel delivery)

**Given** a depot with a vehicle fleet and parcels for clients, **find** which vehicle brings which parcel to which client when.

**Constraints** to be **satisfied**:

1. All parcels are delivered on time.
2. No vehicle is overloaded.
3. Driver regulations are respected.
4. . . .

**Objective function** to be **minimised**:

- Cost: the total fuel consumption and driver salary.

Example (Travelling salesperson: optimisation TSP)

**Given** a map and cities, **find** a **shortest** route visiting each city once and returning to the starting city.
Applications in Air Traffic Management

Demand vs capacity

Airspace sectorisation

Contingency planning

Workload balancing

<table>
<thead>
<tr>
<th>Flow</th>
<th>Time Span</th>
<th>Hourly Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>From: Arlanda</td>
<td>00:00 – 09:00</td>
<td>3</td>
</tr>
<tr>
<td>To: west, south</td>
<td>09:00 – 18:00</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>18:00 – 24:00</td>
<td>2</td>
</tr>
<tr>
<td>From: Arlanda</td>
<td>00:00 – 12:00</td>
<td>4</td>
</tr>
<tr>
<td>To: east, north</td>
<td>12:00 – 24:00</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example (Air traffic demand-capacity balancing)

Reroute flights, in height and speed, so as to balance the workload of air traffic controllers in a multi-sector airspace:

Periods 1 & 2
Course 1DL441 - Combinatorial Optimisation using Constraint Programming
Example (Airspace sectorisation)

**Given** an airspace split into \( c \) cells, and a targeted number \( s \) of sectors.

**Find** a colouring of the cells into \( s \) connected convex sectors, with minimal imbalance of the workloads of their air traffic controllers.

There are \( s^c \) possible colourings, but very few optimally satisfy the constraints: is intelligent search necessary?
Applications in Biology and Medicine

Phylogenetic supertree

Haplotype inference

Medical image analysis

Doctor rostering

Phytophthora adelinae
Eucalyptus minor
Meganaphyes camptodes
Eucalyptus pauciflora
Polygorgon maritimum
OICOMOEA CROMOPHORA
THAILANDIUM BUDDLEI
Euphorbia capitata
Punica granatum orange
Punica granatum red
Punica granatum purple
Punica granatum pink
Punica granatum
tomato
Phenolphoto insperata

Mother
T C A G A C
Father
T G A C T C
Child

Haplotypes
Genotype
Homzygous
Heterozygous
Example (What supertree is maximally consistent with several given trees that share some species?)

Oceanodroma castro
Hydronates pelagicus
Macronectes giganteus
Fulmarus glacialoides
Fulmarus glacialis
Bulweria bulwerii
Procellaria cinerea
Calonectris diomedea
Puffinus assimilis
Puffinus puffinus
Puffinus yelkouan
Puffinus mauretanicus
THALASSARCHE BULLERI
Thalassarche chrysostoma
Phoebetria fusca
Phoebetria palpebrata
Phoebastria albatrus
Phoebastria immutabilis
Diomedea amsterdamsensis
DIOMEDEA EPOMOPHORA

Pygoscelis adeliae
Eudyptula minor
Megadyptes antipodes
Eudyptes pachyrhynchus
Pelagodroma marina
DIOMEDEA EPOMOPHORA
THALASSARCHE BULLERI
Daption capense
Pelecanoides georgicus
Pachyptila vittata
Pachyptila turtur
Procellaria westlandica
Puffinus griseus
Puffinus huttoni
Pterodroma inexpectata
Pterodroma cookii
Example (Haplotype inference by pure parsimony)

**Given** $n$ child genotypes, with homo- & heterozygous sites:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C / G</td>
<td>T</td>
<td>C</td>
<td>A / T</td>
<td>C</td>
</tr>
</tbody>
</table>


<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A / T</td>
<td>G</td>
<td>T</td>
<td>C / G</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>


<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
<td>T</td>
<td>C</td>
<td>T</td>
<td>C</td>
</tr>
</tbody>
</table>


<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>G</td>
<td>T</td>
<td>C</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>


<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>G</td>
<td>T</td>
<td>G</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>


**find** a minimal set of (at most $2 \cdot n$) parent haplotypes:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
<td>T</td>
<td>C</td>
<td>T</td>
<td>C</td>
</tr>
</tbody>
</table>


<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>G</td>
<td>T</td>
<td>C</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>


<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>G</td>
<td>T</td>
<td>G</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>


**so that** each given genotype conflates 2 found haplotypes.
Applications in Programming and Testing

**Constraint Problems**

**Optimisation**

**Constraint Programming (CP)**

**Success Stories**

**Course Organisation**

---

**Robot-task sequencing**

---

**Sensor-net configuration**

---

**Compiler design**

---

**Base-station testing**

---

Applications in Programming and Testing

**Robot-task sequencing**

---

**Sensor-net configuration**

---

**Compiler design**

---

**Base-station testing**

---
Other Application Areas

### School timetabling

<table>
<thead>
<tr>
<th>Time</th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:00</td>
<td>MTE02</td>
<td>LAB1502</td>
<td>NF1220</td>
<td>NF1220</td>
<td>NF1220</td>
</tr>
<tr>
<td></td>
<td>Group 1</td>
<td>Data</td>
<td>Group 3</td>
<td>Group 7</td>
<td>Group 7</td>
</tr>
<tr>
<td>9:00</td>
<td>MTE02</td>
<td>LAB1502</td>
<td>NF1220</td>
<td>NF1220</td>
<td>NF1220</td>
</tr>
<tr>
<td></td>
<td>Group 1</td>
<td>Data</td>
<td>Group 3</td>
<td>Group 7</td>
<td>Group 7</td>
</tr>
<tr>
<td>10:00</td>
<td>CSE020</td>
<td>PRACT01</td>
<td>CSE020</td>
<td>CSE020</td>
<td>CSE020</td>
</tr>
<tr>
<td></td>
<td>Alg 1</td>
<td>Pract</td>
<td>Alg 1</td>
<td>Pract</td>
<td>Pract</td>
</tr>
<tr>
<td>11:00</td>
<td>PRACT01</td>
<td>CSE020</td>
<td>CSE020</td>
<td>CSE020</td>
<td>CSE020</td>
</tr>
<tr>
<td></td>
<td>Pract</td>
<td>Alg 1</td>
<td>Pract</td>
<td>Alg 1</td>
<td>Pract</td>
</tr>
<tr>
<td>12:00</td>
<td>LUNCH</td>
<td>LUNCH</td>
<td>LUNCH</td>
<td>LUNCH</td>
<td>LUNCH</td>
</tr>
<tr>
<td>13:00</td>
<td>LUNCH</td>
<td>LUNCH</td>
<td>LUNCH</td>
<td>LUNCH</td>
<td>LUNCH</td>
</tr>
<tr>
<td>14:00</td>
<td>LUNCH</td>
<td>LUNCH</td>
<td>LUNCH</td>
<td>LUNCH</td>
<td>LUNCH</td>
</tr>
<tr>
<td>15:00</td>
<td>LUNCH</td>
<td>LUNCH</td>
<td>LUNCH</td>
<td>LUNCH</td>
<td>LUNCH</td>
</tr>
</tbody>
</table>

### Sports tournament design

Security: SQL injection?

### Other Application Areas

- School timetabling
- Sports tournament design
- Security: SQL injection?
- Container packing
Definition

In a constraint problem, values have to be found for all the unknowns, called variables (in the mathematical sense) and ranging over given sets called domains, so that:

- All the given constraints on the variables are satisfied.
- Optionally: A given objective function on the variables has an optimal value: minimal cost or maximal profit.

Definition

A candidate solution to a constraint problem assigns to each variable a value within its domain. The search space consists of all candidate solutions.
Search spaces are often larger than the universe!
Many important real-life problems are NP-hard and can only be solved exactly & fast enough by intelligent search, unless $P = NP$:

NP-hardness is not where the fun ends, but where it begins!
A solving technology offers methods and tools for:

what: **Modelling** constraint problems in declarative language.

and / or

how: **Solving** constraint problems *intelligently*:

- **Search**: Explore the space of candidate solutions.
- **Inference**: Reduce the space of candidate solutions.
- **Relaxation**: Exploit solutions to easier problems.

A solver is a software that takes a model as input and tries to solve the modelled problem.

Combinatorial (= discrete) optimisation covers satisfaction and optimisation problems, for variables over discrete sets. The ideas in this course extend to continuous optimisation, to soft optimisation, and to stochastic optimisation.
### Example (Agricultural experiment design, AED)

<table>
<thead>
<tr>
<th></th>
<th>plot1</th>
<th>plot2</th>
<th>plot3</th>
<th>plot4</th>
<th>plot5</th>
<th>plot6</th>
<th>plot7</th>
</tr>
</thead>
<tbody>
<tr>
<td>barley</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>corn</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>millet</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>oats</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>rye</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>spelt</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>wheat</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
</tbody>
</table>

**Constraints to be satisfied:**

1. Equal sample size: Every grain is grown in 3 plots.
2. Equal growth load: Every plot grows 3 grains.
3. Balance: Every grain pair is grown in 1 common plot.

**Instance:** 7 grains, 7 plots, 3 plots/grain, 3 grains/plot, balance 1.

**General term:** balanced incomplete block design (BIBD).
Example (Agricultural experiment design, AED)

<table>
<thead>
<tr>
<th></th>
<th>plot1</th>
<th>plot2</th>
<th>plot3</th>
<th>plot4</th>
<th>plot5</th>
<th>plot6</th>
<th>plot7</th>
</tr>
</thead>
<tbody>
<tr>
<td>barley</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>corn</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>millet</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>oats</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>rye</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>spelt</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>wheat</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Constraints** to be satisfied:

1. Equal sample size: Every grain is grown in 3 plots.
2. Equal growth load: Every plot grows 3 grains.
3. Balance: Every grain pair is grown in 1 common plot.

**Instance**: 7 grains, 7 plots, 3 plots/grain, 3 grains/plot, balance 1.

General term: balanced incomplete block design (BIBD).
Modelling

Example (BIBD integer model: $\checkmark \iff 1$ and $\dashv \iff 0$)

\begin{verbatim}
int: nVarieties, nBlocks, sampleSize, blockSize, balance
set of int: Varieties = 1..nVarieties, Blocks = 1..nBlocks
array[Varieties, Blocks] of var 0..1 : BIBD

satisfy
forall v in Varieties : sampleSize = count(BIBD[v, *], 1)
forall b in Blocks : blockSize = count(BIBD[*, b], 1)
forall v1, v2 in Varieties where v1 < v2 : balance =
    count(b in Blocks)(BIBD[v1, b] \cdot BIBD[v2, b], 1)
\end{verbatim}

Example (Instance data for our AED)

nVarieties = 7, nBlocks = 7,

sampleSize = 3, blockSize = 3, balance = 1
Reconsider the model fragment:

\[
\text{forall } v_1, v_2 \text{ in } \text{Varieties where } v_1 < v_2 : \text{balance} = \text{count}(b \text{ in } \text{Blocks})(\text{BIBD}[v_1, b] \cdot \text{BIBD}[v_2, b], 1)
\]

This constraint is **declarative** (and by the way non-linear), so read it using only the verb “to be” or synonyms thereof:

for all two ordered varieties \(v_1\) and \(v_2\),
the count of blocks \(b\) whose product
\(\text{BIBD}[v_1, b] \cdot \text{BIBD}[v_2, b]\) is 1
must equal \(\text{balance}\)

The constraint is **not procedural**:

for all two ordered varieties \(v_1\) and \(v_2\),
we first count the blocks \(b\) whose product
\(\text{BIBD}[v_1, b] \cdot \text{BIBD}[v_2, b]\) is 1,
and then we check whether that count is \(\text{balance}\)

The latter reading is appropriate for solution **checking**, but solution **finding** performs no such procedural summation.
Constraint Programming Technology

Constraint programming (CP) offers methods and tools for:

what: Modelling constraint problems in a high-level language.

and

how: Solving constraint problems intelligently by:

• either default search upon pushing a button
• or systematic search guided by user-given strategies
• or local search guided by user-given (meta-)heuristics
• or hybrid search

plus inference, called propagation, but little relaxation.

Slogan of CP:

Constraint Program = Model [ + Search ]
CP Solving = Propagation + Search

A CP solver conducts search interleaved with propagation:

Each constraint has a propagator.
Propagation of one Constraint: Propagator

Example

Consider the constraint \textit{CONNECTED([C_1, \ldots, C_n])}, which enforces max one stretch per colour among the \(n\) variables.

From

\[ \ldots ? \ldots \]
\[ \quad \text{black} \quad \text{red} \quad ? \quad ? \quad \text{red} \quad \text{yellow} \quad \ldots ? \ldots \]

the \textit{CONNECTED([C_1, \ldots, C_n])} constraint infers

\[ \text{no redyellow} \quad \quad \text{no redblack} \]

Propagation is the elimination of the impossible values from the current domains of the variables, and thereby accelerates otherwise blind search.
Is CP a Silver Bullet for NP-Hard Problems?

No! CP solvers are complementary to those of:

- Operations research (OR):
  - linear programming (LP)
  - integer linear programming (IP)
  - mixed integer linear programming (MIP)
  - non-linear programming (NLP)
  - ...

- Local search: tabu search, simulated annealing, ...

- Boolean satisfiability (SAT), modulo theories (SMT)

- ...

This leads to hybrid solving technologies!
Success Stories by CP Users and Contributors:

Success stories: CP = technology of choice in scheduling, configuration, personnel rostering, timetabling, ...
Course Contents

- Consistency
- Propagation
- Systematic search
- Local search
- Global constraints: DISTINCT, LINEAR, ELEMENT, REGULAR, etc
- Modelling, including symmetry detection and breaking
- Set variables and set constraints
- Scheduling
- Guest lectures: applications, extensions, etc

Check the previous course instance for an idea of what the course may look like this time.
Learning Outcomes

In order to pass, the student must be able to:

- describe the basic underlying concepts of CP
- describe how a generic CP solver works
- model a combinatorial problem using CP
- devise suitable search strategies
- compare CP programs for a combinatorial problem
- evaluate impact of redundant variables or constraints
- detect and break (some of the) symmetries in a CP
- enhance a CP solver with an additional constraint
- outline other combinatorial optimisation technologies
Course Organisation

- Periods 1 and 2 of the autumn semester
- 22 lectures, in English
- No textbook required
- C++ programming with the free Gecode.org library:
  - 3 assignments, to be done in pairs (2 credits)
  - 3 project parts, to be done in pairs (3 credits)
  in $6 \cdot 3 = 18$ help sessions and 6 solution sessions
- 1 closed-book exam, to be done alone (5 credits)
- Prerequisites: able to define or learn basic concepts in algebra, combinatorics, logic, graph theory, and set theory; able to implement basic search algorithms