Outline

1. The End of Course AD2

2. Combinatorial Optimisation
   - Constraint Problems
   - Solving Technologies
   - Modelling
   - Solving

3. Course AD3
   - Contents
   - New Concepts
   - Learning Outcomes
   - Organisation
Outline

1. The End of Course AD2

2. Combinatorial Optimisation
   - Constraint Problems
   - Solving Technologies
   - Modelling
   - Solving

3. Course AD3
   - Contents
   - New Concepts
   - Learning Outcomes
   - Organisation
In a decision problem we seek a ‘yes’ / ‘no’ answer to an existence question. An instance is given by its input data.

**Example (Travelling salesperson: Decision TSP)**

Given a map and $n$ cities, is there a route visiting each city once, returning to the starting city, and costing at most $c$?

A decision problem $R$ is:

- **in NP** if a witness to a ‘yes’ instance is checkable in time polynomial in the instance size: checking is in $P$;
- **NP-complete** if it is in NP and there is a reduction from each problem $Q$ in NP, transforming in polytime each instance of $Q$ into an instance of $R$ of the same answer.

It is believed that NP-complete problems are **intractable** (or: hard), requiring super-polynomial time to solve exactly.

**Example**

TSP is NP-complete as a witness is checkable in $O(n)$ time and NP-complete Hamiltonian-Cycle problem reduces to it.
In a satisfaction problem we seek a witness for ‘yes’ answer.

Example (Satisfaction TSP)

Given a map and \( n \) cities, find a route visiting each city once, returning to the starting city, and costing at most \( c \).

In an optimisation problem we seek an optimal witness according to some objective function for a ‘yes’ answer.

Example (Optimisation TSP)

Given a map and \( n \) cities, find a cheapest route visiting each city once and returning to the starting city.

In addition to decision problems that are at least as hard as every NP problem (as every NP problem reduces to them), satisfaction and optimisation problems with NP-complete decision versions are often also said to be NP-hard: they are unlikely to be easier than their decision versions.
Several courses at Uppsala University teach techniques for addressing NP-hard optimisation and satisfaction problems:

1TD184  Continuous Optimisation  (period 2)

1DL448  Modelling for Combinatorial Optimisation  (periods 3&1)

1DL441  Comb’l Opti. and Constraint Programming  (period 1–2)

1DL481  Algorithms and Data Structures 3  (period 3)

NP-hardness is not where the fun ends, but where it begins!
Example (Optimisation TSP over \( n \) cities)

A brute-force algorithm evaluates all \( n! \) candidate routes:

- A computer of today evaluates \( 10^6 \) routes / second:

<table>
<thead>
<tr>
<th>( n )</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>40 seconds</td>
</tr>
<tr>
<td>14</td>
<td>1 day</td>
</tr>
<tr>
<td>18</td>
<td>203 years</td>
</tr>
<tr>
<td>20</td>
<td>77k years</td>
</tr>
</tbody>
</table>

- Planck time is shortest useful interval: \( \approx 5.4 \cdot 10^{-44} \) s; a Planck computer would evaluate \( 1.8 \cdot 10^{43} \) routes / s:

<table>
<thead>
<tr>
<th>( n )</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>0.7 seconds</td>
</tr>
<tr>
<td>41</td>
<td>20 days</td>
</tr>
<tr>
<td>48</td>
<td>1.5 \cdot \text{age of universe}</td>
</tr>
</tbody>
</table>

The dynamic program by Bellman-Held-Karp “only” takes \( \mathcal{O}(n^2 \cdot 2^n) \) time: a computer of today takes a day for \( n = 27 \), a year for \( n = 35 \), the age of the universe for \( n = 67 \), and it beats \( n! \) on the Planck computer for \( n \geq 44 \).
Intelligent Search upon NP-Hardness

Do not give up but try to stay ahead of the curve: there is an instance size until which an exact algorithm is fast enough!

Concorde TSP Solver beats Bellman-Held-Karp exact algo: it uses approximation and stochastic local search algos, but it can sometimes prove the exactness (optimality) of its solutions. The largest instance it has solved exactly, in 136 CPU years in 2006, has 85,900 cities! ⚽️ Let the fun begin!
Outline

1 The End of Course AD2

2 Combinatorial Optimisation
   - Constraint Problems
   - Solving Technologies
   - Modelling
   - Solving

3 Course AD3
   - Contents
   - New Concepts
   - Learning Outcomes
   - Organisation
Optimisation is a science of service: to scientists, to engineers, to artists, and to society.
Outline

1. The End of Course AD2

2. Combinatorial Optimisation
   - Constraint Problems
   - Solving Technologies
   - Modelling
   - Solving

3. Course AD3
   - Contents
   - New Concepts
   - Learning Outcomes
   - Organisation
### 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 growth load: Every plot grows 3 grains.
2. Equal sample size: Every grain is grown in 3 plots.
3. Balance: Every grain pair is grown in 1 common plot.

**Instance:** 7 plots, 7 grains, 3 grains/plot, 3 plots/grain, 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><strong>barley</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>corn</strong></td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>millet</strong></td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>oats</strong></td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td><strong>rye</strong></td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td><strong>spelt</strong></td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td><strong>wheat</strong></td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
</tbody>
</table>

**Constraints to be satisfied:**

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

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

<table>
<thead>
<tr>
<th></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. \( \text{#doctors-on-call / day} = 1 \)
2. \( \text{#operations / workday} \leq 2 \)
3. \( \text{#operations / week} \geq 7 \)
4. \( \text{#appointments / week} \geq 4 \)
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>A</td>
<td>call</td>
<td>none</td>
<td>oper</td>
<td>none</td>
<td>oper</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>B</td>
<td>app</td>
<td>call</td>
<td>none</td>
<td>oper</td>
<td>none</td>
<td>none</td>
<td>call</td>
</tr>
<tr>
<td>C</td>
<td>oper</td>
<td>none</td>
<td>call</td>
<td>app</td>
<td>app</td>
<td>call</td>
<td>none</td>
</tr>
<tr>
<td>D</td>
<td>app</td>
<td>oper</td>
<td>none</td>
<td>call</td>
<td>oper</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>E</td>
<td>oper</td>
<td>none</td>
<td>oper</td>
<td>none</td>
<td>call</td>
<td>none</td>
<td>none</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 (Vehicle routing: Parcel delivery)

**Given** a depot with parcels for clients and a vehicle fleet, **find** which vehicle visits 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.
Application Areas

School timetabling

<table>
<thead>
<tr>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
</table>

Sports tournament design

Vehicle routing

Container packing
Applications in Programming and Testing

Robotic task sequencing

Sensor net configuration

Compiler design

Base station testing

New Concepts
Learning Outcomes
Organisation

Applications:
- Robotic task sequencing
- Sensor net configuration
- Compiler design
- Base station testing

Compiler Code Examples:
```
C Compiler
C++ Compiler
```

Base Station Components:
- Magnetic Modular Jack (RJ45)
- SFP+ Connector
- SFP Connector
- Wireless Infrastructure
Applications in Air Traffic Management

Demand vs capacity

Airspace sectorisation

Contingency planning

<table>
<thead>
<tr>
<th>Flow</th>
<th>Time Span</th>
<th>Hourly Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>From: Arland</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: Arland</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>
</tbody>
</table>

Workload balancing
Applications in Biology and Medicine

Phylogenetic supertree

Haplotype inference

Medical image analysis

Doctor rostering

- 18 -
Definitions

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.

A candidate solution to a constraint problem assigns to each variable a value within its domain; it is:

- feasible if all the constraints are satisfied;
- optimal if the objective function takes an optimal value.

The search space consists of all candidate solutions. A solution to a satisfaction problem is feasible. An optimal solution to an optimisation problem is feasible and optimal.
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!
Outline

1. The End of Course AD2

2. Combinatorial Optimisation
   - Constraint Problems
   - Solving Technologies

3. Course AD3
   - Contents
   - New Concepts
   - Learning Outcomes
   - Organisation
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.
Outline

1. The End of Course AD2

2. Combinatorial Optimisation
   - Constraint Problems
   - Solving Technologies
   - Modelling

3. Course AD3
   - Contents
   - New Concepts
   - Learning Outcomes
   - Organisation
### 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 growth load: Every plot grows 3 grains.
2. Equal sample size: Every grain is grown in 3 plots.
3. Balance: Every grain pair is grown in 1 common plot.

**Instance**: 7 plots, 7 grains, 3 grains/plot, 3 plots/grain, 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 growth load: Every plot grows 3 grains.
2. Equal sample size: Every grain is grown in 3 plots.
3. Balance: Every grain pair is grown in 1 common plot.

Instance: 7 plots, 7 grains, 3 grains/plot, 3 plots/grain, balance 1.
General term: balanced incomplete block design (BIBD).
Example (BIBD integer model: $\checkmark \sim 1$ and $\sim \sim 0$)

1 int: nbrBlocks; int: nbrVarieties;
2 set of int: Blocks = 1..nbrBlocks;
3 set of int: Varieties = 1..nbrVarieties;
4 int: blockSize; int: sampleSize; int: balance;
5 array[Varieties,Blocks] of var 0..1: BIBD;
6 solve satisfy;
7 constraint forall(b in Blocks)
8 (blockSize = sum(BIBD[..,b]));
9 constraint forall(v in Varieties)
10 (sampleSize = sum(BIBD[v,..]));
11 constraint forall(v1, v2 in Varieties where v1 < v2)
12 (balance = sum(b in Blocks) (BIBD[v1,b]*BIBD[v2,b]));

Example (Instance data for our AED)

1 nbrBlocks = 7; nbrVarieties = 7;
2 blockSize = 3; sampleSize = 3; balance = 1;
Reconsider the model fragment:

```
11 constraint forall(v1, v2 in Varieties where v1 < v2)
12   (balance = sum(b in Blocks)(BIBD[v1,b] * BIBD[v2,b]));
```

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

*For all two ordered varieties* \( v_1 \) *and* \( v_2 \), *the sum over all blocks* \( b \) *of the products* \( BIBD[v_1,b] \ast BIBD[v_2,b] \) *must equal* \( balance \)

The constraint is **not procedural**:

*For all two ordered varieties* \( v_1 \) *and* \( v_2 \), *we first add up, over all blocks* \( b \), *the products* \( BIBD[v_1,b] \ast BIBD[v_2,b] \), *and then we check whether that sum is equal to* \( balance \)

The latter reading is appropriate for solution **checking**, but solution **finding** performs no such procedural summation.
## Example (Idea for another BIBD model)

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

**Constraints to be satisfied:**

1. **Equal growth load:** Every plot grows 3 grains.
2. **Equal sample size:** Every grain is grown in 3 plots.
3. **Balance:** Every grain pair is grown in 1 common plot.
Example (BIBD set model: a block set per variety)

```plaintext
array[Varieties] of var set of Blocks: BIBD;

(blockSize =
   sum(v in Varieties)(bool2int(b in BIBD[v])));

(sampleSize = card(BIBD[v]));

(balance = card(BIBD[v] inter BIBD[w]));
```

Example (Instance data for our AED)

```plaintext
nbrBlocks = 7; nbrVarieties = 7;
blockSize = 3; sampleSize = 3; balance = 1;
```
### Example (Doctor rostering)

<table>
<thead>
<tr>
<th></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. $\#\text{doctors-on-call} / \text{day} = 1$
2. $\#\text{operations} / \text{workday} \leq 2$
3. $\#\text{operations} / \text{week} \geq 7$
4. $\#\text{appointments} / \text{week} \geq 4$
5. day off after operation day
6. ...

**Objective function** to be minimised:

- Cost: ...
### Example (Doctor rostering)

<table>
<thead>
<tr>
<th></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>none</td>
<td>oper</td>
<td>none</td>
<td>oper</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>Doctor B</td>
<td>app</td>
<td>call</td>
<td>none</td>
<td>oper</td>
<td>none</td>
<td>none</td>
<td>call</td>
</tr>
<tr>
<td>Doctor C</td>
<td>oper</td>
<td>none</td>
<td>call</td>
<td>app</td>
<td>app</td>
<td>call</td>
<td>none</td>
</tr>
<tr>
<td>Doctor D</td>
<td>app</td>
<td>oper</td>
<td>none</td>
<td>call</td>
<td>oper</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>Doctor E</td>
<td>oper</td>
<td>none</td>
<td>oper</td>
<td>none</td>
<td>call</td>
<td>none</td>
<td>none</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)

```plaintext
set of int: Days = 1..7;
set of int: Mon2Fri = 1..5;
enum Doctors = {Dr_A, Dr_B, Dr_C, Dr_D, Dr_E};
enum ShiftTypes = {app, call, oper, none};

array[Doctors, Days] of var ShiftTypes: Roster;

solve minimize ...; % plug in an objective function

constraint forall (d in Days)
  (count (Roster[..,d], call) = 1);
constraint forall (w in Mon2Fri)
  (count (Roster[..,w], oper) <= 2);
constraint count (Roster, oper) >= 7;
constraint count (Roster, app) >= 4;
constraint forall (D in Doctors)
  (regular (Roster[D,..], (oper none|app|call|none)*));
... % other constraints
```
Example (Sudoku)

1 array[1..9,1..9] of var 1..9: Sudoku;
2  ...  % load the hints
3 solve satisfy;
4 constraint forall(row in 1..9)
   (alldifferent(Sudoku[row,..]));
5 constraint forall(col in 1..9)
   (alldifferent(Sudoku[..,col]));
6 constraint forall(i,j in {1,4,7})
   (alldifferent(Sudoku[i..i+2,j..j+2]));
Modelling Languages

The following fully declarative modelling languages are powerful enough to encode NP-hard problems:

- **Boolean satisfiability solving (SAT):** satisfy a set of disjunctions of possibly negated Boolean variables.
- **Mixed integer programming (MIP):** satisfy a set of linear equalities (\(=\)) and inequalities (\(<, \leq, \geq, >\)), but not disequalities (\(\neq\)), over real-number variables and integer variables weighted by real-number constants, such that a linear objective function is optimised.
- **SAT modulo theories (SMT) & constraint programming (CP) do not have such small standardised low-level modelling languages, but enable the higher level of the previous sample models.**

☞ In course 1DL448: Modelling, we use such higher-level models to drive CP, MIP, SAT, SMT, ... solvers.
Outline

1. The End of Course AD2

2. Combinatorial Optimisation
   - Constraint Problems
   - Solving Technologies
   - Modelling
   - Solving

3. Course AD3
   - Contents
   - New Concepts
   - Learning Outcomes
   - Organisation
Solving Technologies

With general-purpose solvers, taking a model as input:

- Boolean satisfiability (SAT)
- SAT modulo theories (SMT)
- (Mixed) integer linear programming (IP & MIP)
- Constraint programming (CP)
- . . .

Methodologies, *usually without* modelling and solvers:

- Dynamic programming (DP)
- Greedy algorithms
- Stochastic local search (SLS)
- Genetic algorithms (GA)
- Ant colony optimisation (ACO)
- Approximation algorithms
- . . .
Solving Technologies

With general-purpose solvers, taking a model as input:

- Boolean satisfiability (SAT) in AD3
- SAT modulo theories (SMT) in AD3
- (Mixed) integer linear programming (IP & MIP) in AD3
- Constraint programming (CP) in 1DL441: COCP

... Methodologies, *usually without* modelling and solvers:

- Dynamic programming (DP) in 1DL231: AD2
- Greedy algorithms in 1DL231: AD2
- Stochastic local search (SLS) in AD3
- Genetic algorithms (GA)
- Ant colony optimisation (ACO)
- Approximation algorithms in AD3

...
Solvers

- **Black-box solvers** (for SAT, SMT, MIP, . . . ) have general-purpose search + inference + relaxation that is difficult to influence by the modeller.

- **Glass-box solvers** (for CP, . . . ) have general-purpose search + inference + relaxation that is easy to influence, if desired, by the modeller.

- **Special-purpose solvers** (for TSP, . . . ) exist for pure problems (without side constraints).
Correctness Is Not Enough for Models
Modelling is an Art

There are good & bad models for each constraint problem: AD3 and 1DL448: Modelling

- Different models of a problem may take different time on the same solver for the same instance.
- Different models of a problem may scale differently on the same solver for instances of growing size.
- Different solvers may take different time on the same instance on the same model.

Good modellers are worth their weight in gold!

Use solvers: based on decades of cutting-edge research, they are very hard to beat on exact solving.
Outline

1. The End of Course AD2
2. Combinatorial Optimisation
   - Constraint Problems
   - Solving Technologies
   - Modelling
   - Solving
3. Course AD3
   - Contents
   - New Concepts
   - Learning Outcomes
   - Organisation
Outline

1. The End of Course AD2

2. Combinatorial Optimisation
   - Constraint Problems
   - Solving Technologies
   - Modelling
   - Solving

3. Course AD3
   - Contents
   - New Concepts
   - Learning Outcomes
   - Organisation
Common thread = dealing with NP-hardness:

1. Stochastic local search (SLS)
2. Mixed integer programming (MIP)
3. Amortised analysis (CLRS3: Chapter 17)
4. Probabilistic analysis (Chapter 5)
5. Randomised algorithms: universal hashing, ...
6. Proving **NP-completeness** by reduction (Chapter 34)
7. Boolean satisfiability (SAT)
8. SAT modulo theories (SMT)
9. Approximation algorithms (Chapter 35)

**CLRS3 Textbook:**
Outline

1 The End of Course AD2

2 Combinatorial Optimisation
   - Constraint Problems
   - Solving Technologies
   - Modelling
   - Solving

3 Course AD3
   - Contents
   - New Concepts
   - Learning Outcomes
   - Organisation
In a **probabilistic algorithm analysis**, we use probability theory: knowing or assuming the distribution of the inputs, we compute the **average-case time** (as opposed to the worst-case time) of a deterministic algorithm.

**Example**

The brute-force string matching algorithm for finding all occurrences of a pattern $P$ of length $m$ within a text $T$ of length $n \geq m$ takes $O(n - m + 1)$ time on average when $P$ and $T$ are random strings, but this is a completely unreasonable assumption. (Chapter 32 in CLRS3)

Probabilistic analysis helps gain insight into a problem and helps design an efficient algorithm for it, when we have a reasonable assumption on the distribution of the inputs.
A randomised algorithm (as opposed to a deterministic algorithm) itself makes random choices, independently of the actual distribution of the inputs. We refer to the time of a randomised algorithm as expected time (not: average time).

### Examples

- A randomised algorithm by Karger-Klein-Tarjan (1993) computes in $O(V + E)$ expected time a minimum spanning tree (MST) of a connected undirected graph with vertex set $V$ and edge set $E$. (Chapter 23)

- A randomised algorithm computes in $O(m)$ expected time a prime number larger than $m$ for fingerprinting in the Rabin-Karp string matcher. (Chapter 32)

Many randomised algorithms have no worst-case input!
In an amortised analysis, we compute the worst-case time of a chain of data-structure operations, and average it over the operations. We refer to this time as an amortised time (as opposed to an average-case time, as no probability is used here, and the possibly non-tight worst-case time).

Examples

- A chain of \( m \) find-and-compress-paths or union-by-rank operations on disjoint sets of \( n \) items takes \( \mathcal{O}(m \cdot \lg^* n) \) time, where \( \lg^* n \leq 5 \) in practice. (Chapter 21)

- In a Fibonacci heap of \( n \) items, extracting a minimum takes \( \mathcal{O}(\lg n) \) amortised time, and decreasing a key takes \( \mathcal{O}(1) \) amortised time. (Chapter 19; not in AD2)

- Prim’s MST algorithm takes at worst \( \mathcal{O}(E + V \lg V) \) time when using a Fibonacci heap. (Chapter 23)
Dealing in polynomial time with (instances of) optimisation problems where brute-force or exact solving is too costly:

- A **greedy algorithm** builds a feasible solution variable by variable, making locally optimal choices in the hope of reaching an optimal solution. Greedy algorithms build either **provably optimal** solutions (for example, Prim’s MST algorithm and Dijkstra’s single-source shortest paths algorithm) or **at-best optimal** solutions.

- A **local search algorithm** repairs a possibly infeasible candidate solution, by reassigning some variables at every iteration, until an allocated resource (such as an iteration count or a time budget) is exhausted, in the hope of reaching a feasible or even optimal solution.

- An **approximation algorithm** for an NP-hard problem builds a feasible solution whose objective value is **provably** within a known factor of the optimum.

All techniques are orthogonal: there exist randomised local search algorithms, greedy approximation algorithms, etc.
Outline

1. The End of Course AD2

2. Combinatorial Optimisation
   - Constraint Problems
   - Solving Technologies
   - Modelling
   - Solving

3. Course AD3
   - Contents
   - New Concepts
   - Learning Outcomes
   - Organisation
In order to pass, the student must be able to:

- analyse NP-completeness of an algorithmic problem;
- use advanced algorithm analysis methods, such as amortised analysis and probabilistic analysis;
- use advanced algorithm design methods in order to approach hard algorithmic problems in a pragmatic way, such as by using:
  - randomised algorithms: universal hashing, …
  - approximation algorithms
  - local search: simulated annealing, tabu search, …
  - mixed integer programming (MIP)
  - Boolean satisfiability (SAT)
  - SAT modulo theories (SMT)
Outline

1. The End of Course AD2

2. Combinatorial Optimisation
   - Constraint Problems
   - Solving Technologies
   - Modelling
   - Solving

3. Course AD3
   - Contents
   - New Concepts
   - Learning Outcomes
   - Organisation
Course Organisation & Time Budget

Period 3: January to March, budget = 133.3 hours:

- 1 presentation, by student-chosen duo team on topic of their choice: budget = 15 hours/student (pass / fail)
- 11 lectures, including a mandatory guest lecture, plus 3 mandatory student-led presentation sessions: budget = 21 hours
- 2 teacher-chosen assignments with 3 help sessions, 1 grading session, and 1 solution session each, to be done in student-chosen duo team: budget = avg 30 hours/assignment/student (2 credits)
- 1 written closed-book exam of 3 hours, to be done alone: budget = 37.3 hours (3 credits)
- Prerequisites: Algorithms and Data Structures 2 (course 1DL231) or equivalent
Examination

Programming / modelling, experimenting, and reporting:

- Stochastic local search (SLS): Assignment 1
- Mixed integer programming (MIP): Assignment 1
- Boolean satisfiability (SAT): Assignment 2
- SAT modulo theories (SMT): Assignment 2

Exercises drawn from a list of questions at Student Portal:

- Amortised analysis and probabilistic analysis: exam
- Randomised algorithms: exam
- NP-completeness: 50% threshold at exam
- Approximation algorithms: exam
2 Assignment Cycles of 4 Weeks

Let $D_i$ be the deadline of Assignment $i$, with $i \in 1..2$:

- $D_i – 16$: publication & all needed material taught: start!
- $D_i – 14$: help session a: attend if you need it; no emails
- $D_i – 9$: help session b: attend if you need it; no emails
- $D_i – 2$: help session c: attend if you need it; no emails
- $D_i \pm 0$: submission, by 13:00 Swedish time on a Friday
- $D_i + 6$ by 16:00: initial score $a_{ij} \in \{0, 1, 2, 3, 4, 5\}$ points for each Problem $j$ of Assignment $i$, with $j \in 1..2$
- $D_i + 7$: teamwise oral grading session on Problems $j$ with $a_{ij} \in \{1, 2\}$: possibility of earning 1 extra point for final score; otherwise final score = initial score
- $D_i + 7 = D_{i+1} – 14$: solution session & help session a
Let $a_{ij}$ be final score on Problem $j$ of Asgmt $i$, with $i, j \in 1..2$:

- **20% threshold**: $\forall i, j \in 1..2 : a_{ij} \geq 20\% \cdot 5 = 1$
  No catastrophic failure on individual problems

- **30% threshold**: $\forall i : a_i = \sum_{j=1}^{2} a_{ij} \geq 30\% \cdot (2 \cdot 5) = 3$
  Can offset partial failure on problems or assignments

- **50% threshold**: $a = \sum_{i=1}^{2} a_i \geq 50\% \cdot (2 \cdot 2 \cdot 5) = 10$
  The formula for grades 3, 4, and 5 is at Student Portal

- **Worth going full-blast**: The assignment score $a$ is meshed with the exam score $e$ in order to determine the overall course grade, if $10 \leq a \leq 20$ and $10 \leq e \leq 20$; see the formula at Student Portal
Presentation

Topic:

- Pick a well-cited, well-published, and peer-reviewed academic paper on an algorithm or data structure within the exam topics of this course.

- Ask us, if need be.

Deadlines:

- Wed 14 Feb at 13:00: upload topic proposal
- Wed 21 Feb at 17:00: secure our topic approval
- Wed 28 Feb to Fri 2 Mar: present and upload slides

The length & order of presentations will be fixed in due time.
Assignment and Presentation Rules

Register **teams** by Sun 21 Jan at 23:59 at Student Portal:

- **Duo teams:** Two consenting partners sign up at portal.
- **Solo teams:** Apply to head teacher, who rarely agrees.
- **Random partner?** Assent to TA, else you’re bounced.

Other considerations:

- **Why (not) like this? Why no email reply?** See FAQ.
- **Partner swapping:** Allowed, but to be declared to TA.
- **Partner scores may differ** if no-show or passivity.
- **No freeloader:** Implicit honour declaration in reports that each partner can individually explain everything. Random checks will be made by us.
- **No plagiarism:** Implicit honour declaration in reports. Extremely powerful detection tools will be used by us. Suspected cases of using or providing will be reported.