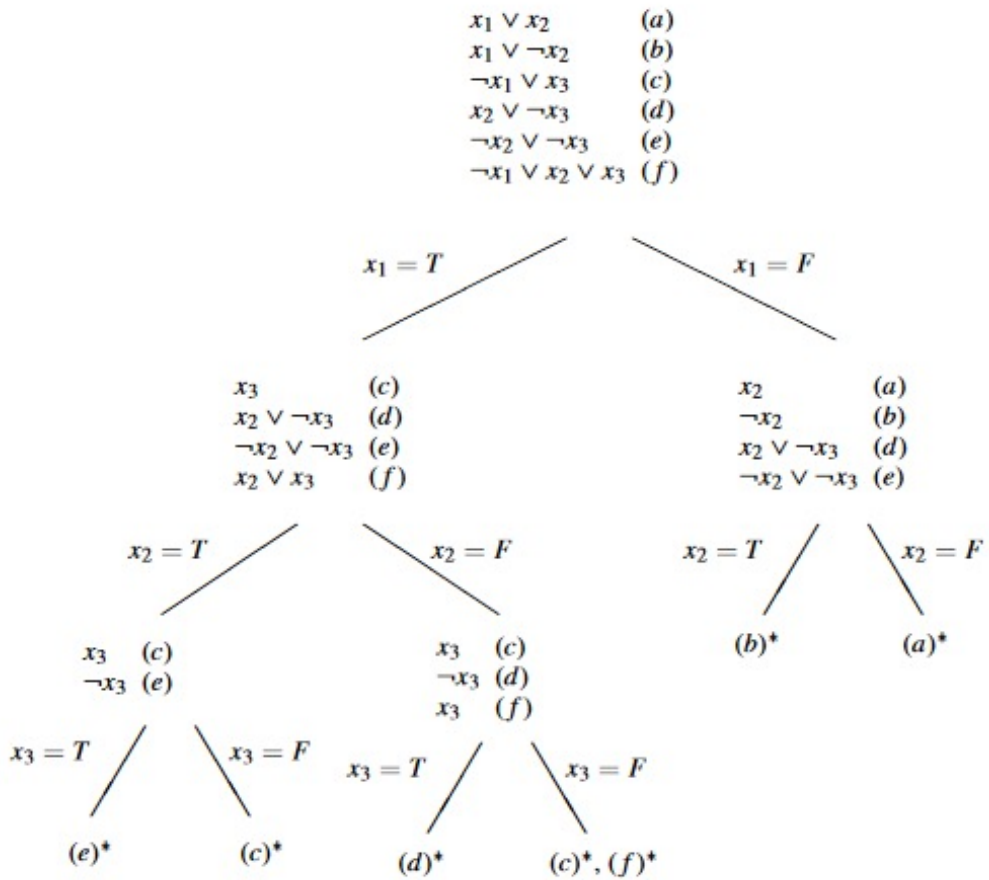
Benders decomposition

Ioannis Avgerinos
iavgerinos@aueb.gr



Overview

$$
\begin{aligned}
&\textbf{(MILP)}: \quad C_{max} \\
&\text{s.t.}: \\
&\sum_{i,j\in J, i\neq j} X_{i,j,m,t} \leq 1, && \forall t \in \mathcal{T}, m \in M \quad (1)\\
&\sum_{t\in J} X'_{i,m,t} \leq 1, && \forall t \in \mathcal{T}, m \in M \quad (2)\\
&\sum_{t\in \mathcal{T}} X_{i,j,m,t} \leq 1, && \forall i,j \in J, i \neq j, m \in M \quad (3)\\
&\sum_{j\in J, t\in \mathcal{T}} X_{0,j,m,t} \leq 1, && \forall m \in M \quad (4)\\
&\sum_{m\in M} Q_{i,m} = v_i, && \forall i \in J \quad (5)\\
&L_i \cdot Y_{i,m} \leq Q_{i,m} \leq v_i \cdot Y_{i,m}, && \forall i \in J, m \in M \quad (6)\\
&Y_{i,m} = \sum_{t\in \mathcal{T}, j\in J, j\neq i} X_{i,j,m,t}, && \forall i \in J, m \in M \quad (7)\\
&Y_{j,m} = \sum_{t\in \mathcal{T}, i\in J, i\neq j} X_{i,j,m,t}, && \forall j \in J, m \in M \quad (8)\\
&\sum_{t\in \mathcal{T}} X_{i,j,m,t} + \sum_{t\in \mathcal{T}} X_{j,i,m,t} \leq 1 && \forall i,j \in J, i,j \neq 0, i \neq j, m \in M \quad (9)\\
&\sum_{i,j\in J, i\neq j, t\in \mathcal{T}} X_{i,j,m,t} = \sum_{i\in J} Y_{i,m} - 1, && \forall m \in M \quad (10)\\
&X_{i,j,m,t} + \sum_{\substack{i'\in J, i'\neq i,\\ t'\in \mathcal{T}, t'\leq t}} X_{j,i',m,t'} \leq 1, && \forall i \in J, j \in J, i \neq j, m \in M, t \in \mathcal{T} \quad (11)\\
&X_{i,j,m,t} \cdot l_{i,j,m} \leq \sum_{t'=t}^{t+l_{i,j,m}-1} X'_{j,m,t'} \\
& && \forall i,j \in J, i \neq j, m \in M, t \in \mathcal{T} \setminus \{T - r | 1 \leq r \leq l_{i,j,m}\} \quad (12)\\
&\sum_{t\in \mathcal{T}} X'_{j,m,t} \leq \sum_{i\in J, i\neq j, t\in \mathcal{T}} l_{i,j,m} \cdot X_{i,j,m,t}, && \forall j \in J, m \in M \quad (13)\\
&\sum_{i\in J, m\in M} X'_{i,m,t} \leq R && \forall t \in \mathcal{T} \quad (14)\\
&\sum_{i\in J, m\in M, t\in q} l_\tau \cdot X'_{i,m,t} \leq u_q && \forall q \in \mathcal{D} \quad (15)\\
&C_{j,m} - C_{i,m} + V(1 - \sum_{t\in \mathcal{T}} X_{i,j,m,t}) \geq Q_{j,m} \cdot \frac{u_j}{s_m} + S_{i,j,m} \cdot \sum_{t\in \mathcal{T}} X_{i,j,m,t}, \\
& && \forall i,j \in J, j \neq i, m \in M \quad (16)\\
&C_{j,m} \geq \sum_{i\in J, i\neq j, t\in \mathcal{T}} X_{i,j,m,t}(\tau_{t-1} + S_{i,j,m}) + Q_{j,m} \cdot \frac{u_j}{s_m}, && \forall j \in J, m \in M \quad (17)\\
&\bar{S}_i \cdot Y_{i,m} + Q_{i,m} \cdot \frac{u_i}{s_m} \leq C_{i,m} \leq C_{\max} && \forall i \in J, m \in M \quad (18)\\
&\sum_{j\in J, t\in \mathcal{T}, t>\lceil \frac{R}{M} \rceil} X_{0,j,m,t} = 0 && \forall m \in M \quad (19)\\
&Y_{i,m}, X'_{i,m,t}, X_{i,j,m,t} \in \{0,1\}, C_{i,m}, Q_{i,m} \in \mathbb{R}^+, \quad && \forall i,j \in J, m \in M, t \in \mathcal{T}
\end{aligned}
$$

- Real operations consist of multiple problems to be jointly optimised.
- Such problems incur an intractable number of variables/constraints – regular MILPs are not efficient.
- Solving large optimisation problems is time-consuming – decision-making requires agile actions, facilitated by fast computational tools.

Overview

$\mathcal{M}_{k-1}$:

min z

subject to:

Assignment of jobs to machines

$$\sum_{m \in M} W_{i,m}^{k-1} = 100 \qquad \forall i \in J^\star$$

$$100 \cdot y_{i,m}^{k-1} \geq W_{i,m}^{k-1} \qquad \forall i \in J^\star, m \in M$$

Sequencing of jobs

$$y_{i,m}^{k-1} = \sum_{j \in J, j \neq i} x_{i,j,m}^{k-1} \qquad \forall i \in J^\star, m \in M$$

$$y_{i,m}^{k-1} = \sum_{j \in J, j \neq i} x_{j,i,m}^{k-1} \qquad \forall i \in J^\star, m \in M$$

$$\sum_{j \in J} x_{0,j,m}^{k-1} \leq 1 \qquad \forall m \in M$$

$$n_{i,m}^{k-1} - n_{j,m}^{k-1} + |J| \cdot x_{i,j,m}^{k-1} \leq |J| - 1 \qquad \forall i, j \in J^\star, m \in M$$

$$n_{i,m}^{k-1} \leq |J| - 1 \qquad \forall i \in J^\star, m \in M$$

Objective function

$$z \geq \sum_{j \in J^\star} \frac{p_{j,m}}{100} \cdot W_{j,m}^{k-1} + \sum_{i \in J} \sum_{j \in J^\star} s_{i,j,m} \cdot x_{i,j,m}^{k-1} \qquad \forall m \in M$$

Variables

$$x_{i,j,m}^{k-1} \in \{0,1\} \qquad \forall j \in J, i \in J \setminus \{j\}, m \in M$$

$$y_{i,m}^{k-1} \in \{0,1\} \qquad \forall i \in J, m \in M$$

$$n_{i,m}^{k-1}, W_{i,m}^{k-1} \in \mathbb{Z}_+ \qquad \forall i \in J, m \in M$$

$\mathcal{S}_k$:

min ζ^k

subject to:

$$\texttt{Cumulative}((\texttt{start\_of}(\bar{\mu}_{i,m}^k)), (\bar{s}_{i,m}^{k-1}), 1, R)$$

$$\texttt{start\_of}(\bar{\mu}_{i,m}^k) \geq \texttt{end\_of}(\bar{\mu}_{i-1,m}^k) + \bar{p}_{i-1,m}^{k-1} \qquad \forall m \in \bar{M}_{k-1}, i = 2, \ldots, |\bar{\nu}_m^{k-1}|$$

$$\zeta^k \geq \texttt{end\_of}(\bar{\mu}_{i,m}^k) + \bar{p}_{i,m}^{k-1} \qquad \forall m \in \bar{M}, i = 1, \ldots, |\bar{\nu}_m^{k-1}|$$

$$\bar{\mu}_{i,m}^k : \texttt{interval}(\text{size} = \bar{s}_{i,m}^{k-1}) \qquad \forall m \in \bar{M}, i = 1, \ldots, |\bar{\nu}_m^{k-1}|$$

- Partition to smaller subproblems: each one of them is more easily and quickly solved.
- Combining partial solutions of the subproblems construct holistic solutions of the original problem.
- Exploitation of the structure of problems: each stage of the problem is a distinct subproblem.

Overview

Decomposition methods:

A large and complicated optimisation problem:

$$\begin{aligned} min \quad & f(x) + g(y) \\ & C(x), C(y), C(x,y) \\ & x \in D_x, y \in D_y \end{aligned}$$

is partitioned into:

- the **master problem**
- a (family of) **subproblem(s)**

Each counterpart is solved more easily, hence constructing a feasible solution of the original problem.

Overview

Decomposition methods:

The **master problem**:

$$\begin{aligned} min \quad & f(x) + g(y) \\ & C(x), C(y), C(x,y) \\ & x \in D_x, y \in D_y \end{aligned}$$

will only use less variables, constraints, costs.

It is easily seen that the master problem is a **relaxation** of the original one. The optimal objective value is a **lower bound** of the optimal value.

Overview

Decomposition methods:

The **master problem**:

$$\begin{aligned} min \quad & f(x) + g(y) \\ & C(x), C(y), C(x,y) \\ & x \in D_x, y \in D_y \end{aligned}$$

Let $\bar{x}$ be the optimal solution and z be the optimal value of the master problem.

Overview

Decomposition methods:

The **subproblem**:

$$\begin{aligned} min \quad & f(\bar{\boldsymbol{x}}) + g(y) \\ & C(\bar{\boldsymbol{x}}), C(y), C(\bar{x}, y) \\ & y \in D_y \end{aligned}$$

will restore the remaining variables, constraints and costs, given the partial solution of the master problem.

It is easily seen that the solution of the subproblem is **feasible** for the original one. The optimal objective value is an **upper bound** of the optimal value.

Overview

Decomposition methods:

The **subproblem**:

$$\begin{aligned} min \quad & f(\bar{\boldsymbol{x}}) + g(y) \\ & C(\bar{\boldsymbol{x}}), C(y), C(\bar{x}, y) \\ & y \in D_y \end{aligned}$$

Let ζ be the optimal value of the subproblem.

If $\bar{\boldsymbol{z}}$ is the optimal value of the original problem, then for all feasible solutions $x \in D_x, y \in D_y$:

$$\boldsymbol{z \leq \bar{z} \leq \zeta}$$

Overview

Decomposition methods:

The **subproblem**:

$$\begin{aligned} min \quad & f(\bar{x}) + g(y) \\ & C(\bar{x}), C(y), C(\bar{x}, y) \\ & y \in D_y \end{aligned}$$

The partitioning scheme will provide a feasible solution of the problem and an optimality gap:

$$\boldsymbol{Gap\% = 100 \cdot \frac{\zeta - z}{z}}$$

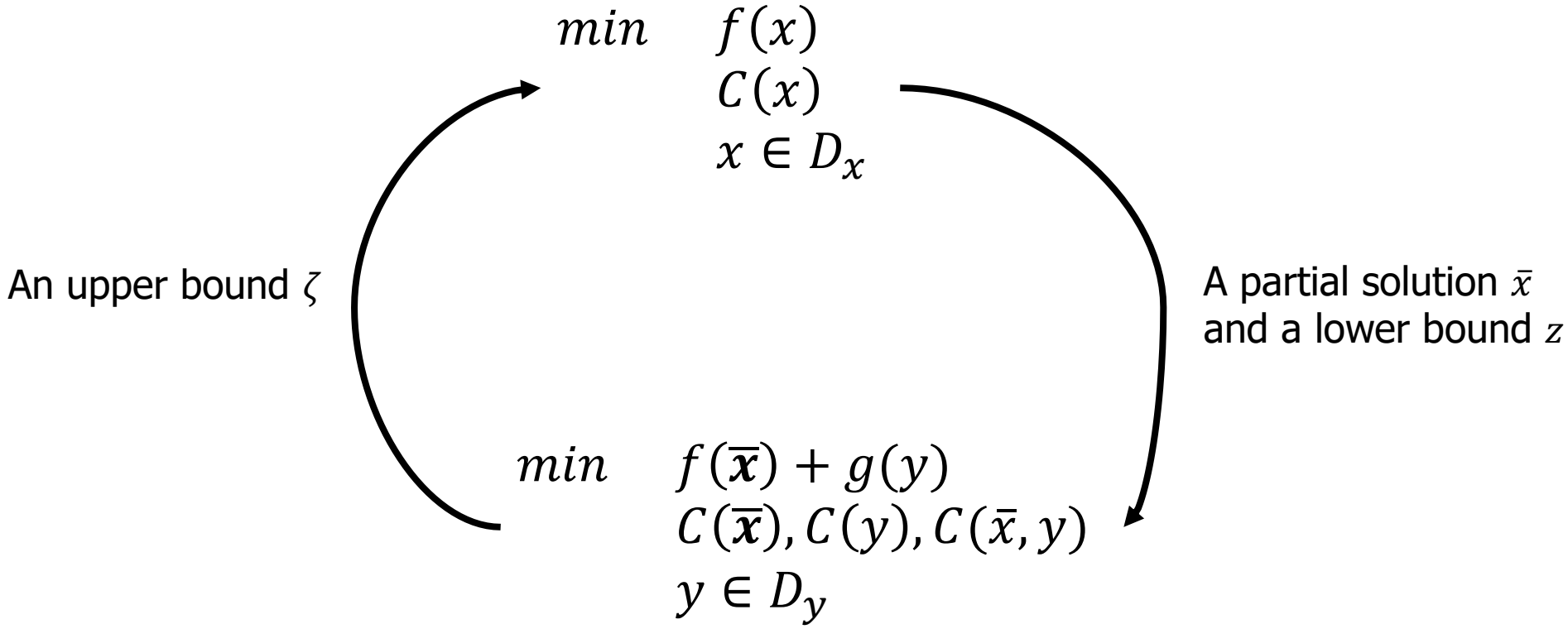
Let ζ be the optimal value of the subproblem.

If $\bar{z}$ is the optimal value of the original problem, then for all feasible solutions $x \in D_x, y \in D_y$:

$$\boldsymbol{z \leq \bar{z} \leq \zeta}$$

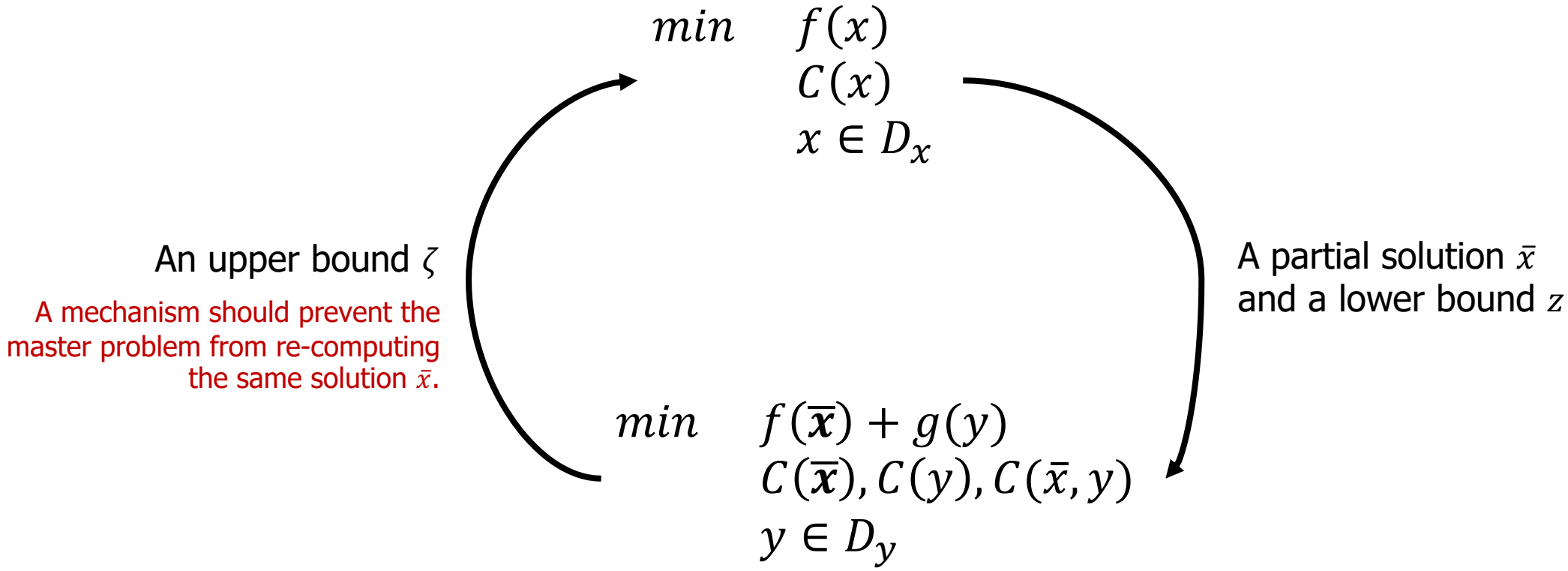
Logic-Based Benders Decomposition

An iterative exchange of knowledge between the counterparts of a partitioning scheme:



Logic-Based Benders Decomposition

An iterative exchange of knowledge between the counterparts of a partitioning scheme:



Logic-Based Benders Decomposition

Cuts: A set of linear inequalities which restrict the computation of $\bar{x}$.

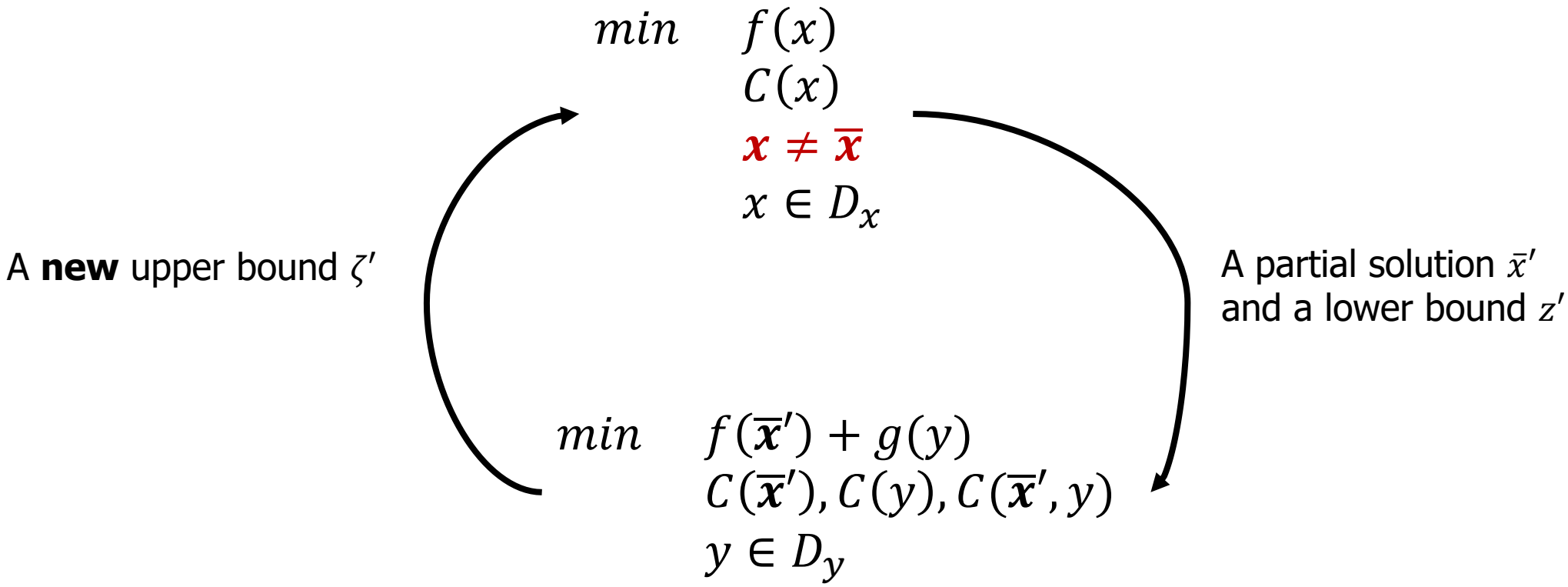
The master problem is reformulated as:

$$
\begin{aligned}
min \quad & f(x) \\
& C(x) \\
& \boldsymbol{x \neq \bar{x}} \\
& x \in D_x
\end{aligned}
$$

Solving this problem to optimality will imply a **different partial solution** $\bar{x}' \neq \bar{x}$ and a new **lower bound** $z' \geq z$.

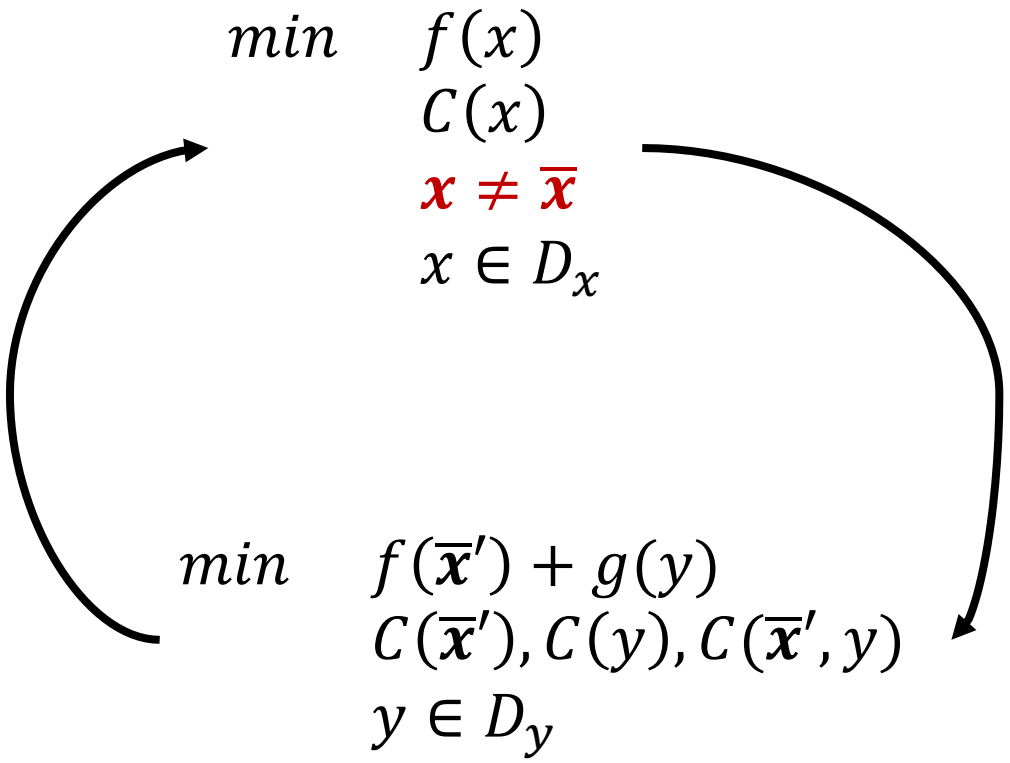
Logic-Based Benders Decomposition

The next iteration will be:



Logic-Based Benders Decomposition

The next iteration will be:



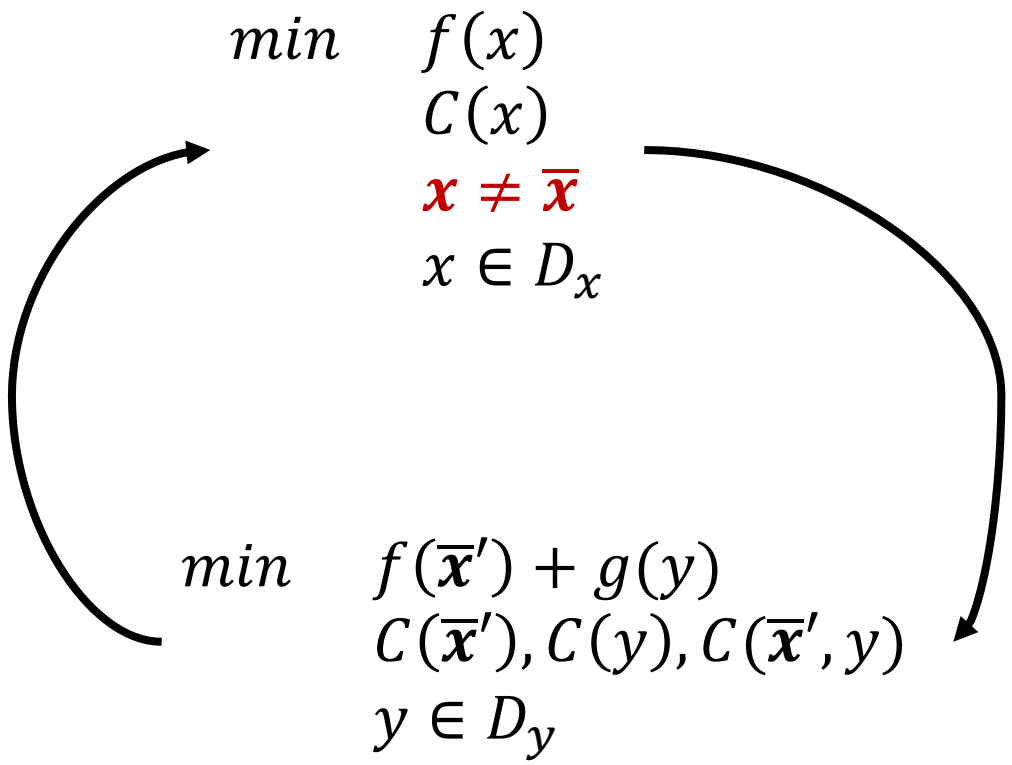
A **new** upper bound ζ'

The new upper bound ζ' is not necessarily improved. But eventually, as long as new partial solutions are provided, improved upper bounds will be explored.

A partial solution $\bar{x}'$ and a lower bound z'

Logic-Based Benders Decomposition

The next iteration will be:



A **new** upper bound ζ'

The new upper bound ζ' is not necessarily improved. But eventually, as long as new partial solutions are provided, improved upper bounds will be explored.

A partial solution $\bar{x}'$ and a lower bound z'

On the contrary, the new lower bounds will **always** be improved at each iteration, gradually converging to optimality.

Logic-Based Benders Decomposition

Cuts: A set of linear inequalities which restrict the computation of $\bar{x}$.

The master problem is reformulated as:

$$
\begin{aligned}
min \quad & f(x) \\
& C(x) \\
& \boldsymbol{x \neq \bar{x}} \\
& x \in D_x
\end{aligned}
$$

The mathematical expression $x \neq \bar{x}$ is **non-linear** – the master problem is usually a Mixed-Integer Linear Program.

A cuts-generation scheme should construct linear expressions which restrict previous solutions.

Logic-Based Benders Decomposition

Theorem of Hooker:

Let z be the objective function of the master problem, $\bar{x}^t$ be the optimal solution of the master problem, and ζ^t be the upper bound at any iteration t.

If a **bounding function** $B_t(x)$ has the following properties:

1. $\mathrm{f(x)} + \mathrm{g(y)} \geq \mathrm{B_t(x)}$ for all feasible solutions x, y
2. $\mathrm{B_t}(\bar{\mathrm{x}}) = \zeta$

then the **cuts** $z \geq B_t(x)$ will converge to the **optimal solution** of the original problem after finitely many steps.

In practice, we should construct a linear equality, for which:

- If $x = \bar{x}^t$, then $z \geq \zeta$.
- Otherwise, $z \geq 0$.

Logic-Based Benders Decomposition

The Classical Benders Decomposition:

In 1962, J.F.Benders designed a partitioning method, consisted of:

- a **master problem**, which is a Mixed-Integer Linear Program
- a **subproblem**, which is a Linear Program (strictly using continuous variables).

Assuming that the following LP is the subproblem:

$$\begin{aligned} min \quad & cy \\ & Ay \geq b \\ & y \geq 0 \end{aligned}$$

then, the **dual** would be:

$$\begin{aligned} max \quad & ub \\ & c = uA \\ & u \geq 0 \end{aligned}$$

Logic-Based Benders Decomposition

The Classical Benders Decomposition:

In 1962, J.F.Benders designed a partitioning method, consisted of:

- a **master problem**, which is a Mixed-Integer Linear Program
- a **subproblem**, which is a Linear Program (strictly using continuous variables).

Assuming that the following LP is the subproblem:

$$\begin{aligned} min \quad & cy \\ & Ay \geq b \\ & y \geq 0 \end{aligned}$$

then, the **dual** would be:

$$\begin{aligned} max \quad & ub \\ & c = uA \\ & u \geq 0 \end{aligned}$$

Exploiting the property of **strong duality**, we can generate linear inequalities which always satisfy the properties of the Theorem.

Logic-Based Benders Decomposition

The Classical Benders Decomposition:

The presented method, namely the **Benders Decomposition** is one the most popular partitioning methods for complicated Mixed-Integer Linear Programs.

However, its inapplicability for schemes of **integer subproblems** is restrictive for real industrial problems, which usually require programs of integer variables.

Logic-Based Benders Decomposition

Logic-based extension:

In 2000, J.N.Hooker extended the framework, presenting the **Logic-Based Benders Decomposition**.
LBBD provides flexibility:

- Subproblems of integer variables are allowed.
- Subproblems could be solved by any optimizing method (as long as this will compute the optimal solution of the subproblem).

For LBBD, the bounding functions are **problem-specific**. However, a globally valid function is:

$$B_t(x) = \begin{cases} \zeta^t, \text{if } x = \bar{x}^t \\ 0, \text{otherwise} \end{cases}$$

Logic-Based Benders Decomposition

A minimal example of a cut:

Let ζ^t be the upper bound at any iteration t, given a partial solution:

$$\bar{x}^t = \{x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0\}$$

If z is the objective function of the master problem, then the following inequality:

$$z \geq \zeta^t - \zeta^t \cdot (x_1 + x_2 - x_3 - x_4 - 2)$$

will ensure that:

- if $x = \bar{x}^t \rightarrow x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$, then $z \geq \zeta^t$.
- otherwise, $x_1 + x_2 - x_3 - x_4 < 2 \rightarrow z \geq 0$.

Manufacturing – a case in Textile industry

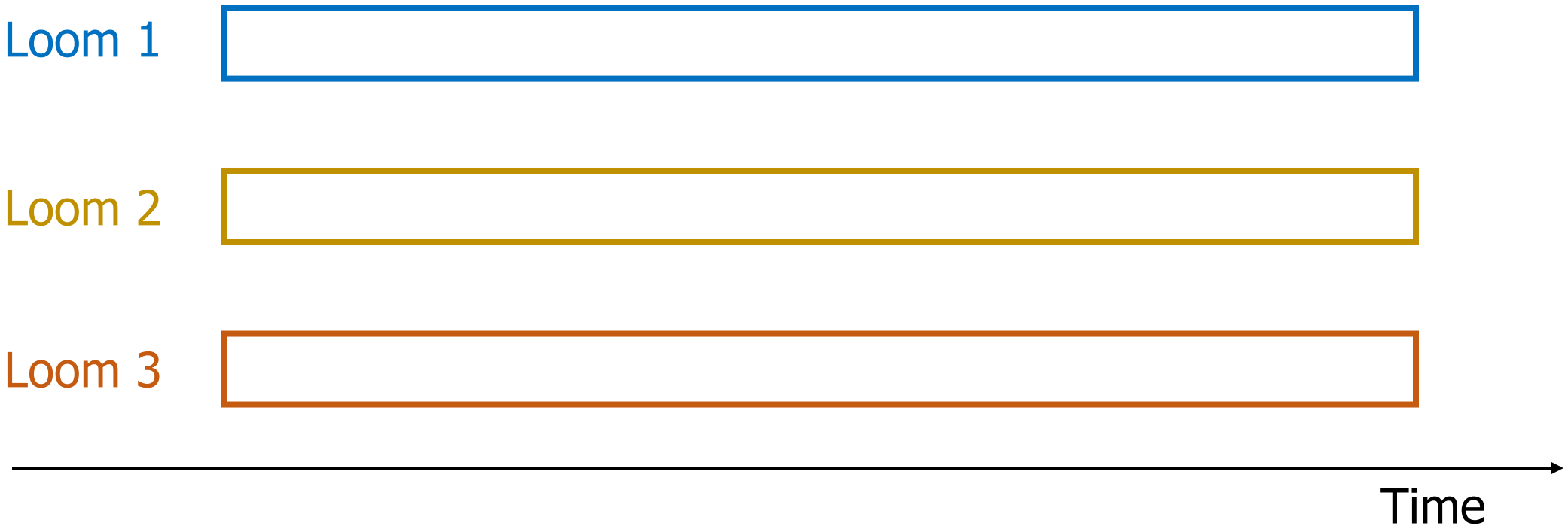
PIACENZA
1733

A textile industry in Italy manufactures woolen fabrics.

The scheduling of the manufacturing line is an optimisation problem

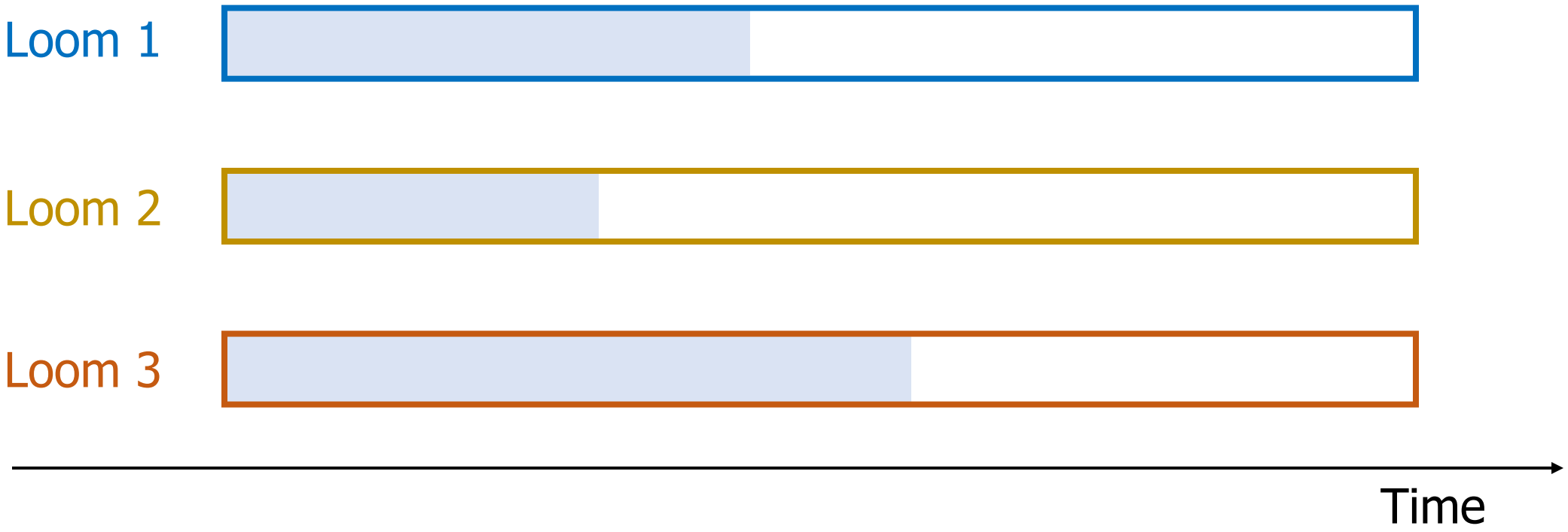
Manufacturing – a case in Textile industry

The industry has a set of weaving looms, which can operate in parallel.



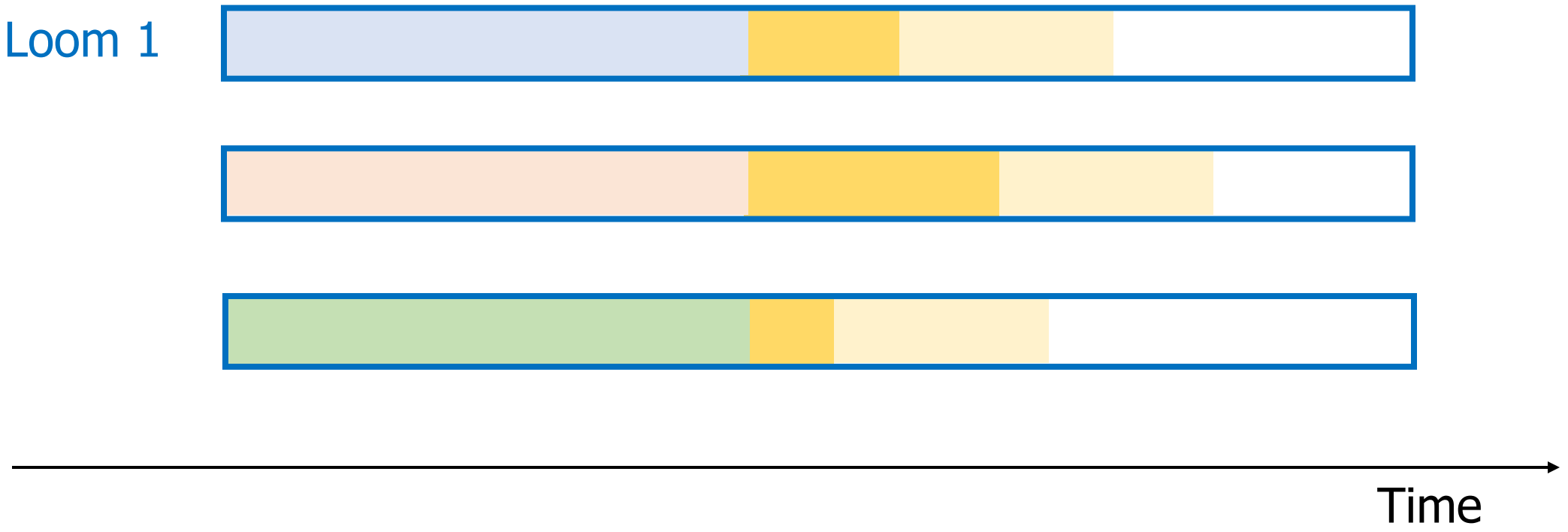
Manufacturing – a case in Textile industry

Textiles should be processed to the weaving looms. The duration of each process depends on the speed of the loom and the size of the product to be manufactured. E.g., for the same textile, the processing time to each loom is different:



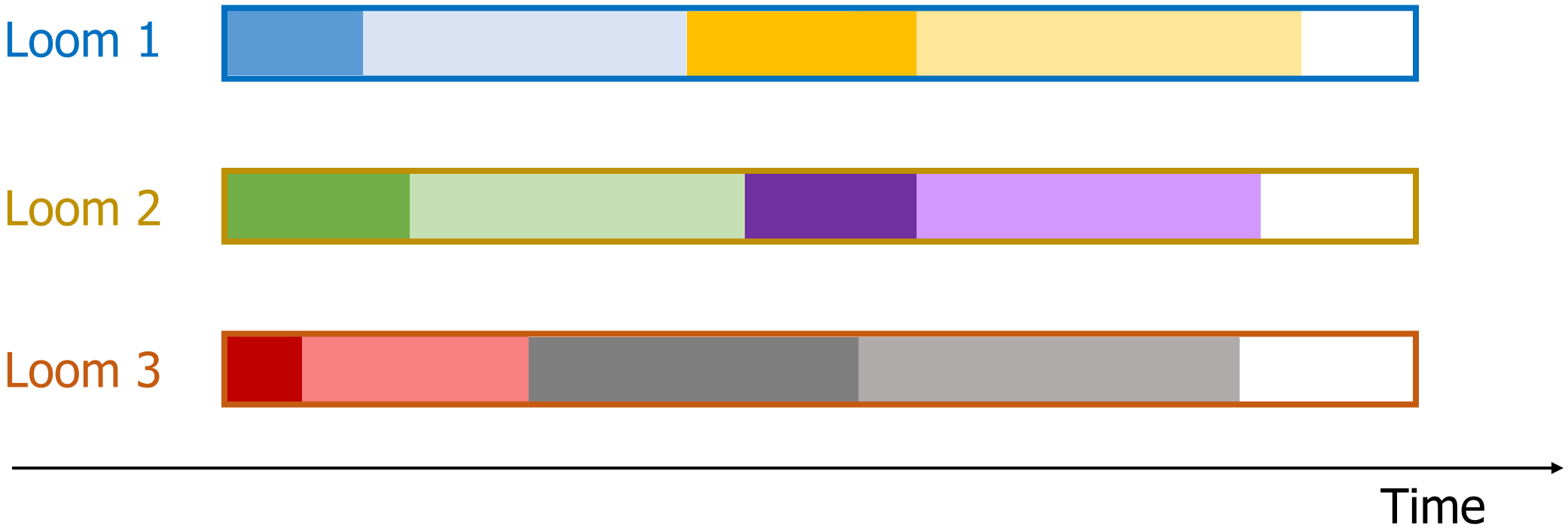
Manufacturing – a case in Textile industry

Each textile should be set up to the loom, before the processing begins. The setup procedure is sequence-dependent:



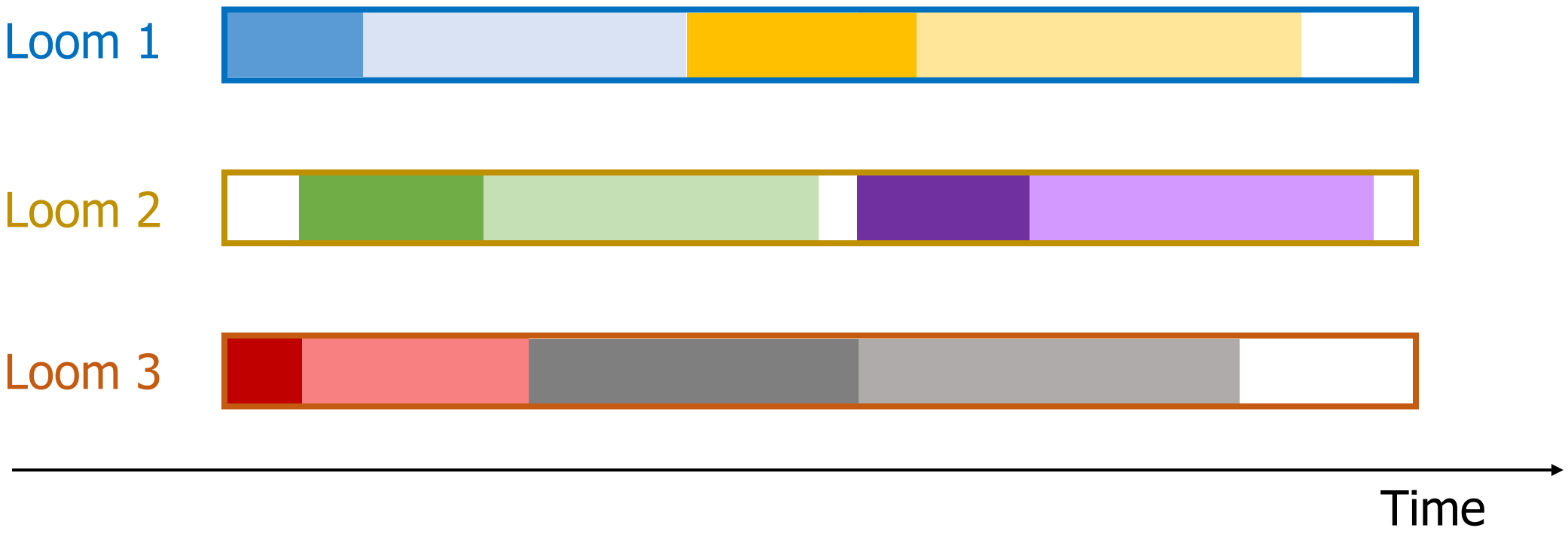
Manufacturing – a case in Textile industry

The daily schedule of the industry includes the **assignment** of textiles to looms and their **sequencing** to minimise critical metrics:



Manufacturing – a case in Textile industry

Limited resources: The setup is conducted by a set of R working groups. During a setup, one working group is occupied – no more than R setup operations can be executed simultaneously. E.g., if $R = 2$**:**



Manufacturing – a case in Textile industry

The described problem resembles with the **Parallel Machines Scheduling Problem**:

J: Jobs – Orders of the textile industry
M: Machines – Weaving looms, which operate in parallel

- All jobs must be assigned to one machine.
- Each machine can be occupied by no more than one job at the same time.

Manufacturing – a case in Textile industry

Position-based MILP

For a minimal example of 5 jobs to 2 machines:

- Each machine has a number of empty slots (positions) – as many as the number of jobs.
- Each job should be assigned to one slot – each slot can be either remain vacant or be occupied by one job.

	1	2	3	4	5
m_1	-	-	j_1	j_2	j_3
m_2	-	-	-	j_4	j_5

Manufacturing – a case in Textile industry

$$\min z$$

$$\sum_{i \in J} \sum_{m \in M} x_{ijm} = 1 \qquad \forall j \in J$$

$$\sum_{j \in J} x_{ijm} \leq 1 \qquad \forall i \in J, m \in M$$

$$P_{im} = \sum_{j \in J} p_{jm} \cdot x_{ijm} \qquad \forall i \in J, m \in M$$

$$\sum_{k \in J \setminus \{j\}} s_{kjm} \cdot x_{i-1km} - S_{im} \leq V \cdot (1 - x_{ijm}) \qquad \forall i \in J^*, m \in M, j \in J$$

$$C_{im} = C_{i-1m} + P_{im} + S_{im} \qquad \forall i \in J^*, m \in M$$

$$C_{0m} = P_{0m} + S_{0m} \qquad \forall m \in M$$

$$\sum_{j \in J} x_{ijm} \geq \sum_{j \in J} x_{i-1jm} \qquad \forall i \in J^*, m \in M$$

$$x_{ijm} \in \{0, 1\} \qquad \forall i \in J, j \in J, m \in M$$

$$P_{im}, S_{im}, C_{im} \geq 0 \qquad \forall i \in J, m \in M$$

Manufacturing – a case in Textile industry

$$\min z$$

$$\sum_{i \in J} \sum_{m \in M} x_{ijm} = 1 \qquad \forall j \in J$$

$$\sum_{j \in J} x_{ijm} \leq 1 \qquad \forall i \in J, m \in M$$

- Any scheduling objective could be set to z (e.g., makespan, total completion times, total tardiness etc.)
- Each job must be assigned to one slot of any machine.
- Each slot can be occupied by one job at most.

Manufacturing – a case in Textile industry

$$P_{im} = \sum_{j \in J} p_{jm} \cdot x_{ijm} \qquad \forall i \in J, m \in M$$

$$\sum_{k \in J \setminus \{j\}} s_{kjm} \cdot x_{i-1km} - S_{im} \leq V \cdot (1 - x_{ijm}) \qquad \forall i \in J^*, m \in M, j \in J$$

- The processing time of each slot is determined by the processing time of the assigned job. If no jobs are assigned to the slot, then the processing time is set to 0.
- Big-M constraint: If a variable x_{ijm} is set to 1 (i.e., job j is assigned to slot i of machine m), then the setup of the slot is equal/greater than the sequence-dependent setup time of job j. Otherwise, the left-hand side is always valid.

Manufacturing – a case in Textile industry

$$C_{im} = C_{i-1m} + P_{im} + S_{im} \qquad \forall i \in J^*, m \in M$$

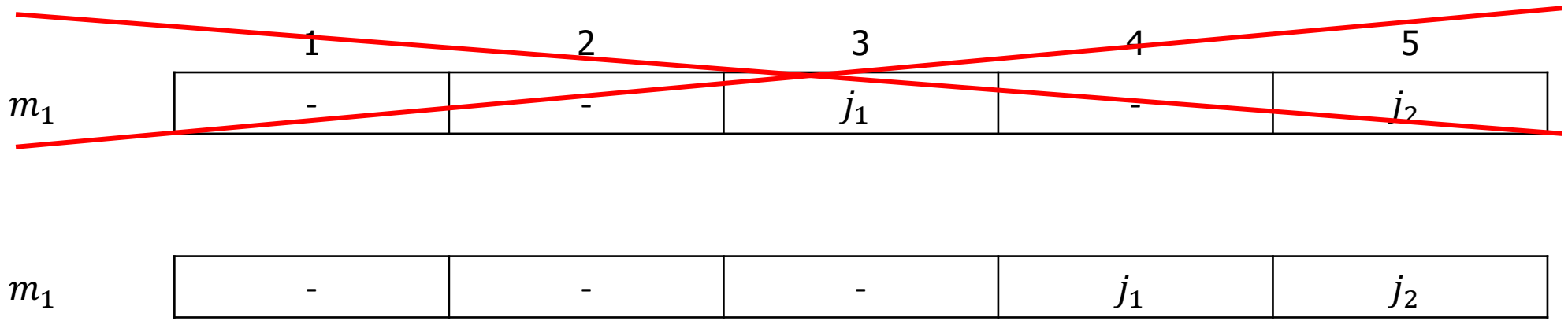
$$C_{0m} = P_{0m} + S_{0m} \qquad \forall m \in M$$

- The completion time of each slot is equal with the completion time of the preceding one, added by the processing and setup times.
- For the first slot of a machine, the completion time is equal with the sum of the processing and the setup time.

Manufacturing – a case in Textile industry

$$\sum_{j \in J} x_{ijm} \geq \sum_{j \in J} x_{i-1jm} \qquad \forall i \in J^*, m \in M$$

- No intermediate slots remain vacant.



Manufacturing – a case in Textile industry

Limited resources:
The linearisation of resource constraints requires the construction of t-index variables:

$$x_{ijmt} = \begin{cases} 1, \text{if job } j \text{ is processed at slot } i \text{ of machine } m \text{ at time instance } t \\ 0, \text{otherwise} \end{cases}$$

meaning that the number of variables is multiplied by t times – the number of distinct time instances.

For a dataset of medium scale (e.g., 50 jobs on 5 machines of 50 slots), a working shift of 8 hours requires 480 minutes - t indices.

$50 \times 50 \times 5 \times 480 = 6.000.000$ x_{ijmt} variables

Manufacturing – a case in Textile industry

Decomposition:

Master problem: The position-based MILP, assuming that an infinite number of resources is available.

This is a relaxation of the original problem:

- All constraints of the master problem are valid for the original problem.
- A part of the constraints of the original problem are not considered for the master problem.

Therefore, the solution of the master problem is a **lower bound** of the global optimal solution.

Manufacturing – a case in Textile industry

Decomposition:

Subproblem: Given a set of sequences of jobs, as acquired by the master problem, the actual number of available resources is imposed.

- The sequences of jobs will remain intact.
- The completion times of jobs will be extended, since a limited number of resources is now considered.
- The solution will be feasible – an **upper bound** is computed.

Manufacturing – a case in Textile industry

Decomposition:

Subproblem: Given a set of sequences of jobs, as acquired by the master problem, the actual number of available resources is imposed.

$$\min \zeta$$

$$\texttt{noOverlap}(\texttt{sequence}_m) \qquad \forall m \in \bar{M}$$

$$\texttt{startAtEnd}(\texttt{process}_{m_i}, \texttt{setup}_{m_i}) \qquad \forall m \in \bar{M}, i \in \{1, ..., |m|\}$$

$$\texttt{previous}(\texttt{sequence}_m, \texttt{process}_{m_{i-1}}, \texttt{setup}_{m_i}) \qquad \forall m \in \bar{M}, i \in \{2, ..., |m|\}$$

$$\texttt{Cumulative}(\texttt{start}(\texttt{setup}_{m_i}), (\texttt{size}(\texttt{setup}_{m_i}), 1, R)$$

$$\texttt{sequence}_m : \{\texttt{setup}_{m_1}, ..., \texttt{setup}_{m_{|m|}}, \texttt{process}_{m_1}, ..., \texttt{process}_{m_{|m|}}\} \qquad \forall m \in \bar{M}$$

$$\texttt{setup}_{m_i}, \texttt{process}_{m_i} : Interval \qquad \forall m \in \bar{M}, i \in \{1, ..., |m|\}$$

Constraint Programming

An **exact** solving method for maximisation/minimisation problems, using global functions as constraints:

Interval variables: Variables which receive two – start time, end time

`intervalVar:  startOf, endOf, sizeOf`

Sequence variables: Variables of multiple intervals

$\texttt{sequenceVar: [intervalVar}_i\texttt{|i}\in\texttt{I]}$

A sequence of interval variables for processes I.

Constraint Programming

An **exact** solving method for maximisation/minimisation problems, using global functions as constraints:

Interval variables: Variables which receive two – start time, end time

`intervalVar:  startOf, endOf, sizeOf`

Global functions: reserved expressions which indicate specific constraints

`Cumulative(intervalVar`$_i$`|i∈I, d`$_i$`|i∈I, c`$_i$`|i∈I, C )`

If each process i of duration d_i consumes c_i resources, and C resources are available, then the Cumulative constraints restricts the violation of resource availability.

Constraint Programming

An **exact** solving method for maximisation/minimisation problems, using global functions as constraints:

Interval variables: Variables which receive two – start time, end time

`intervalVar:  startOf, endOf, sizeOf`

Scheduling constraints:

`startAtEnd(intervalVar`$_i$`, intervalVar`$_j$`)`

Process i will start at the end of process j.

Constraint Programming

An **exact** solving method for maximisation/minimisation problems, using global functions as constraints:

Interval variables: Variables which receive two – start time, end time

```
intervalVar:   startOf, endOf, sizeOf
```

Scheduling constraints:

```
noOverlap(sequenceVar)
```

No interval variables of the sequence will overlap.

Constraint Programming

An **exact** solving method for maximisation/minimisation problems, using global functions as constraints:

Interval variables: Variables which receive two – start time, end time

`intervalVar:  startOf, endOf, sizeOf`

Scheduling constraints:

`previous(sequenceVar,intervalVar`$_i$`, intervalVar`$_j$`)`

Process i and process j will be successive in a sequence.

Manufacturing – a case in Textile industry

Decomposition:

Subproblem: Given a set of sequences of jobs, as acquired by the master problem, the actual number of available resources is imposed.

$$\min \zeta$$

$$\texttt{noOverlap}(\texttt{sequence}_m) \qquad \forall m \in \bar{M}$$

$$\texttt{startAtEnd}(\texttt{process}_{m_i}, \texttt{setup}_{m_i}) \qquad \forall m \in \bar{M}, i \in \{1, ..., |m|\}$$

$$\texttt{previous}(\texttt{sequence}_m, \texttt{process}_{m_{i-1}}, \texttt{setup}_{m_i}) \qquad \forall m \in \bar{M}, i \in \{2, ..., |m|\}$$

$$\texttt{Cumulative}(\texttt{start}(\texttt{setup}_{m_i}), (\texttt{size}(\texttt{setup}_{m_i}), 1, R)$$

$$\texttt{sequence}_m : \{\texttt{setup}_{m_1}, ..., \texttt{setup}_{m_{|m|}}, \texttt{process}_{m_1}, ..., \texttt{process}_{m_{|m|}}\} \qquad \forall m \in \bar{M}$$

$$\texttt{setup}_{m_i}, \texttt{process}_{m_i} : \mathit{Interval} \qquad \forall m \in \bar{M}, i \in \{1, ..., |m|\}$$

Given that the sequences of jobs are defined by the solution of the master problem, using a CP formulation implies a more easily solved problem than using an equivalent MILP.

Manufacturing – a case in Textile industry

Decomposition:

Benders cuts: After the computation of the upper bound, a set of constraints is added to the master problem.

If $\bar{x}_{ijm}$ are the solutions of variables x_{ijm}, then:

$$z \geq \zeta - \zeta \cdot (|J| - \sum_{x_{ijm}:\bar{x}_{ijm}=1} (1 - x_{ijm}))$$

- If $\sum_{x_{ijm}:\bar{x}_{ijm}=1}(1 - x_{ijm}) = |J|$ (i.e., all jobs are assigned to the same slots), then the objective of the master problem z will receive the value of the upper bound ζ.
- If $\sum_{x_{ijm}:\bar{x}_{ijm}=1}(1 - x_{ijm}) < |J|$ (i.e., at least one job is assigned to a different slot), then a new solution is obtained, and $z \geq 0$.

Programming tools

- Open-source Python package to **formulate** optimisation problems (Linear Programs, Mixed-Integer Linear Programs)
- Syntax which is close to the natural language

Programming tools

$$\min z$$

$$\sum_{i \in J} \sum_{m \in M} x_{ijm} = 1 \qquad \forall j \in J$$

$$\sum_{j \in J} x_{ijm} \leq 1 \qquad \forall i \in J, m \in M$$

$$P_{im} = \sum_{j \in J} p_{jm} \cdot x_{ijm} \qquad \forall i \in J, m \in M$$

$$\sum_{k \in J \setminus \{j\}} s_{kjm} \cdot x_{i-1km} - S_{im} \leq V \cdot (1 - x_{ijm}) \qquad \forall i \in J^*, m \in M, j \in J$$

$$C_{im} = C_{i-1m} + P_{im} + S_{im} \qquad \forall i \in J^*, m \in M$$

$$C_{0m} = P_{0m} + S_{0m} \qquad \forall m \in M$$

$$\sum_{j \in J} x_{ijm} \geq \sum_{j \in J} x_{i-1jm} \qquad \forall i \in J^*, m \in M$$

$$x_{ijm} \in \{0, 1\} \qquad \forall i \in J, j \in J, m \in M$$

$$P_{im}, S_{im}, C_{im} \geq 0 \qquad \forall i \in J, m \in M$$

Programming tools

$$x_{ijm} \in \{0,1\} \qquad \forall i \in J, j \in J, m \in M$$

$$P_{im}, S_{im}, C_{im} \geq 0 \qquad \forall i \in J, m \in M$$

```
# Sets
model.Slots = range(nSlots)
model.Jobs = range(nJobs)
model.Machines = range(nMachines)

# Variables
model.z = Var(within = NonNegativeReals)
model.x = Var(model.Slots, model.Jobs, model.Machines, within = Binary)
model.Completion = Var(model.Slots, model.Machines, within = NonNegativeReals)
model.Processing = Var(model.Slots, model.Machines, within = NonNegativeReals)
model.Setup = Var(model.Slots, model.Machines, within = NonNegativeReals)
```

Programming tools

$$\min z$$

$$\sum_{i \in J} \sum_{m \in M} x_{ijm} = 1 \qquad \forall j \in J$$

$$\sum_{j \in J} x_{ijm} \leq 1 \qquad \forall i \in J, m \in M$$

$$P_{im} = \sum_{j \in J} p_{jm} \cdot x_{ijm} \qquad \forall i \in J, m \in M$$

```
# If job 'j' is assigned to machine 'm', then it will occupy exactly one slot of 'm'.
for j in model.Jobs:
    model.constraints.add(sum(model.x[i, j, m] for i in model.Slots for m in model.Machines) == 1.0)
# Each slot can be occupied by one job at most.
for i in model.Slots:
    for m in model.Machines:
        model.constraints.add(sum(model.x[i, j, m] for j in model.Jobs) <= 1.0)
# Processing time of slot 'i' in machine 'm' is determined by the assigned job 'j'.
for i in model.Slots:
    for m in model.Machines:
        model.constraints.add(model.Processing[i, m] == sum(processing_times[j, m]*model.x[i, j, m] for j in model.Jobs))
```

Programming tools

$$\sum_{k \in J \setminus \{j\}} s_{kjm} \cdot x_{i-1km} - S_{im} \leq V \cdot (1 - x_{ijm}) \qquad \forall i \in J^*, m \in M, j \in J$$

$$C_{im} = C_{i-1m} + P_{im} + S_{im} \qquad \forall i \in J^*, m \in M$$

$$C_{0m} = P_{0m} + S_{0m} \qquad \forall m \in M$$

$$\sum_{j \in J} x_{ijm} \geq \sum_{j \in J} x_{i-1jm} \qquad \forall i \in J^*, m \in M$$

```
# The completion time of slot 'i' in machine 'm' is determined by the completion time of slot 'i-1', added by the processing and setup times of 'i'.
for i in model.Slots:
    for m in model.Machines:
        if i > 0:
            model.constraints.add(model.Completion[i, m] == model.Completion[i-1, m] + model.Processing[i, m] + model.Setup[i, m])
        else:
            model.constraints.add(model.Completion[i, m] == model.Processing[i, m] + model.Setup[i, m])
# If job 'j' is assigned to slot 'i' of machine 'm', then the setup time of this slot is determined by the succession of jobs in slots 'i-1' --> 'i'.
for i in model.Slots:
    if i > 0:
        for j in model.Jobs:
            for m in model.Machines:
                model.constraints.add(sum(setup_times[k, j, m]*model.x[i-1, k, m] for k in model.Jobs) - model.Setup[i, m] <= big_M*(1 - model.x[i, j, m]))
# A slot 'i' can be occupied only if slot 'i-1' is also occupied.
for i in model.Slots:
    if i > 0:
        for m in model.Machines:
            model.constraints.add(sum(model.x[i, j, m] for j in model.Jobs) - sum(model.x[i-1, j, m] for j in model.Jobs) >= 0.0)
```

Programming tools

- Commercial solver for (Mixed-Integer) Linear Programs
- Python API

The master problem is formulated using Pyomo and solved using Gurobi.

Programming tools

- Commercial solver for (Mixed-Integer) Linear Programs, Constraint Programming models
- Python API

The DOCplex component provides a Python API for Constraint Programming models.

Programming tools

$$\min \zeta$$

$$\texttt{noOverlap}(\texttt{sequence}_m) \qquad \forall m \in \bar{M}$$

$$\texttt{startAtEnd}(\texttt{process}_{m_i}, \texttt{setup}_{m_i}) \qquad \forall m \in \bar{M}, i \in \{1, ..., |m|\}$$

$$\texttt{previous}(\texttt{sequence}_m, \texttt{process}_{m_{i-1}}, \texttt{setup}_{m_i}) \qquad \forall m \in \bar{M}, i \in \{2, ..., |m|\}$$

$$\texttt{Cumulative}(\texttt{start}(\texttt{setup}_{m_i}), (\texttt{size}(\texttt{setup}_{m_i}), 1, R)$$

$$\texttt{sequence}_m : \{\texttt{setup}_{m_1}, ..., \texttt{setup}_{m_{|m|}}, \texttt{process}_{m_1}, ..., \texttt{process}_{m_{|m|}}\} \qquad \forall m \in \bar{M}$$

$$\texttt{setup}_{m_i}, \texttt{process}_{m_i} : Interval \qquad \forall m \in \bar{M}, i \in \{1, ..., |m|\}$$

Programming tools

$$\mathbf{sequence}_m : \{\mathbf{setup}_{m_1}, \ldots, \mathbf{setup}_{m_{|m|}}, \mathbf{process}_{m_1}, \ldots, \mathbf{process}_{m_{|m|}}\} \qquad \forall m \in \bar{M}$$

$$\mathbf{setup}_{m_i}, \mathbf{process}_{m_i} : Interval \qquad \forall m \in \bar{M}, i \in \{1, \ldots, |m|\}$$

```python
# Variables
setupVar = {}
for m in range(len(solution)):
    for j in range(len(solution[m])):
        if j > 0:
            start, end, size = (0, big_M), (0, big_M), math.ceil(setup_times[solution[m][j-1]][solution[m][j]][m])
        else:
            start, end, size = (0, big_M), (0, big_M), 0
        setupVar[(m, j)] = subproblem.interval_var(start, end, size, name = f"setup_{m},{j}")
processVar = {}
for m in range(len(solution)):
    for j in range(len(solution[m])):
        start, end, size = (0, big_M), (0, big_M), math.ceil(processing_times[solution[m][j]][m])
        processVar[(m, j)] = subproblem.interval_var(start, end, size, name = f"process_{m},{j}")
sequence = {} # Sequence variables 'sequence[m]': Sequence of setup and processing interval variables on machine 'm'
for m in range(len(solution)):
    intervals = []
    for j in range(len(solution[m])):
        intervals.append(setupVar[(m, j)])
        intervals.append(processVar[(m, j)])
    sequence[m] = subproblem.sequence_var(intervals, name = f"sequence_{m}")
```

Programming tools

$$\texttt{noOverlap}(\texttt{sequence}_m) \qquad \forall m \in \bar{M}$$

$$\texttt{startAtEnd}(\texttt{process}_{m_i}, \texttt{setup}_{m_i}) \qquad \forall m \in \bar{M}, i \in \{1, ..., |m|\}$$

$$\texttt{previous}(\texttt{sequence}_m, \texttt{process}_{m_{i-1}}, \texttt{setup}_{m_i}) \qquad \forall m \in \bar{M}, i \in \{2, ..., |m|\}$$

```python
# Constraints
for m in range(len(solution)):
    subproblem.add(subproblem.no_overlap(sequence[m]))
for m in range(len(solution)):
    for i in range(len(solution[m])):
        subproblem.add(subproblem.start_at_end(processVar[(m, i)], setupVar[(m, i)]))
for m in range(len(solution)):
    for i in range(len(solution[m])):
        if i > 0:
            subproblem.add(subproblem.previous(sequence[m], processVar[(m, i-1)], setupVar[(m, i)]))
```