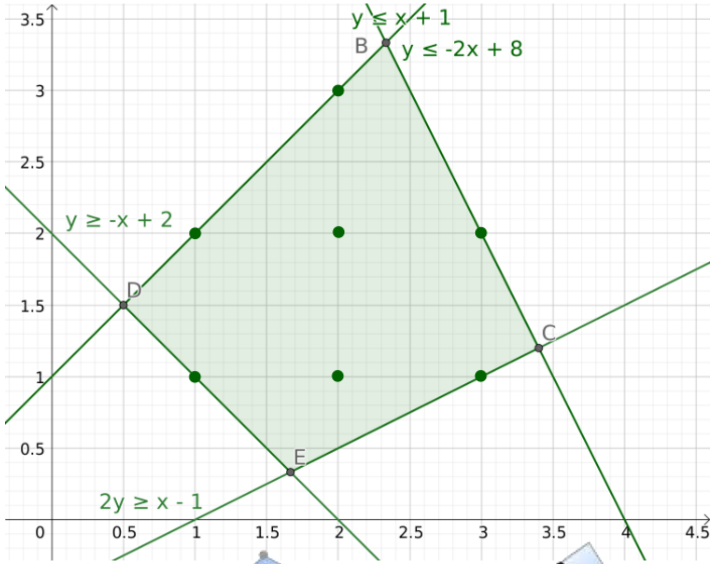
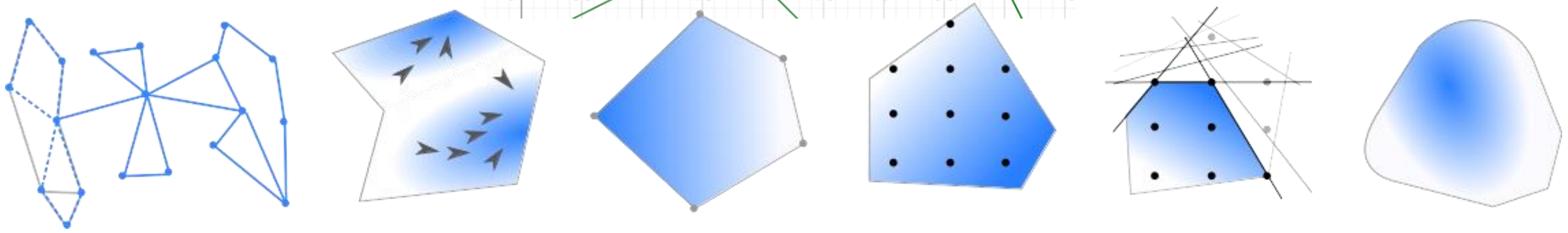
Mixed-Integer Programming 3

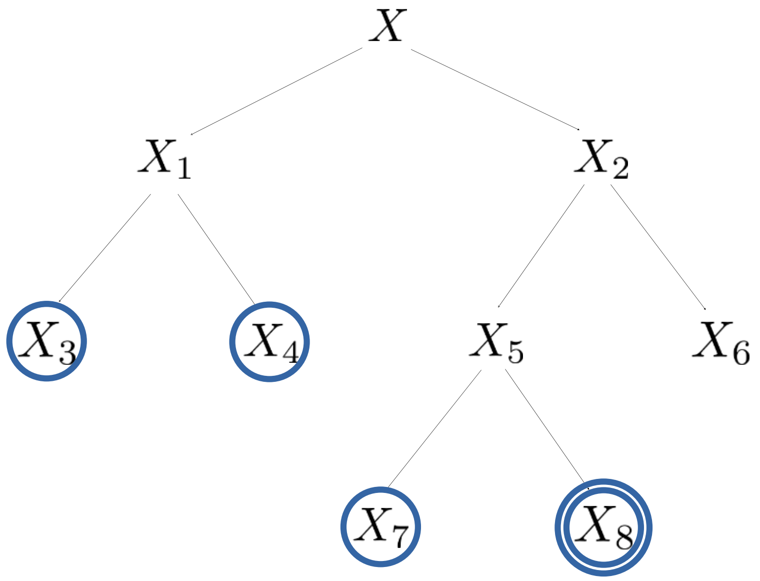
COMP4691 / 8691





MILP Topic Outline

- Linear relaxation
- Modelling and solving
- **Branch and bound**
  - Optimality gap
  - Feasibility heuristics
  - Scheduling problems
- Cutting plane methods
- Duality



Branch and Bound

“Divide and conquer” the solution space.

We calculate upper and lower bounds on the optimal solution to prune the solution space, avoiding the need to enumerate all feasible solutions.

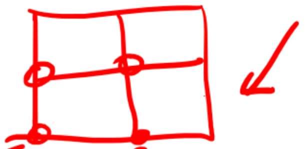
$\rightarrow \alpha$-β PRUNMING

$\leftarrow$ U. MELZ.

1960 **Alison Harcourt** (Doig) and **Ailsa Land**

An automatic method of solving discrete programming problems

Branch and Bound



Feasible region: $\boxed{X}$

Partition the feasible region: $\rightarrow X = \boxed{\cup_t X_t}$ ← NEVER REMOVE FEAS SOL (to simplify notation we allow overlap)

$$o(X_t) := \boxed{\min}\ c^\intercal x \quad \text{s.t. } x \in \boxed{X_t}$$

Partition optimal solution. Generally still expensive to calculate.

$$o(X) = \min_t \ o(X_t)$$

Best partition optimal solution is optimal for overall.

We want some means to find feasible solutions and bound partitions:

$\rightarrow u(X_t)$ A feasible solution found in partition. Can be optimal for partition, or any feasible point found by heuristics (e.g., rounding LP solution).

$x \in X_t$

$x \in X$

$\rightarrow$ An **upper bound** on **partition and overall problem.**

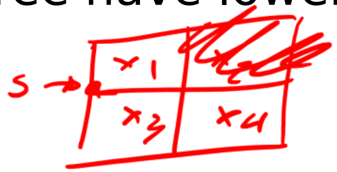
$\rightarrow l(X_t)$ A lower bound on partition optimal solution.
Should be efficient to calculate (e.g., solve linear relaxation).
A **lower bound** on **partition,** *not overall problem.*

Branch and Bound

Say we split a minimisation problem into **4 partitions**. One we have solved exactly to get an upper bound, and the other three have lower bounds:

$$\rightarrow u(X_1) = \boxed{5} \quad (= o(X_1))$$

$$\boxed{l(X_2) = 6,} \quad \boxed{l(X_3) = 3,} \quad l(X_4) = 4$$



What should we do next?

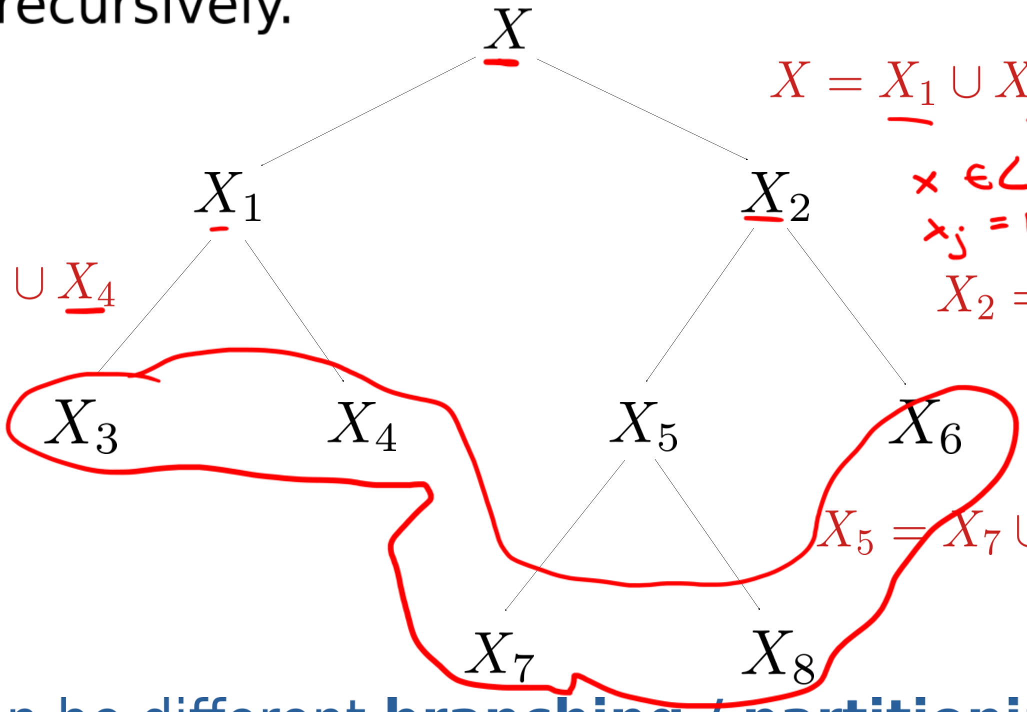
Drop 2, explore 3 or 4.

What are our next possible actions then?

- Solving partitions 3 or/and 4 exactly ←
- Using heuristics to get a feasible solution with a better upper bound $\rightarrow u(X_3) = 4$
- Further partitioning 3 or 4 in order to get a better lower bound

Branch and Bound

In practice the partitions are not predetermined but rather generated online recursively.



$X = X_1 \cup X_2$

$X_1 = X_3 \cup X_4$

$X_2 = X_5 \cup X_6$

$X_5 = X_7 \cup X_8$

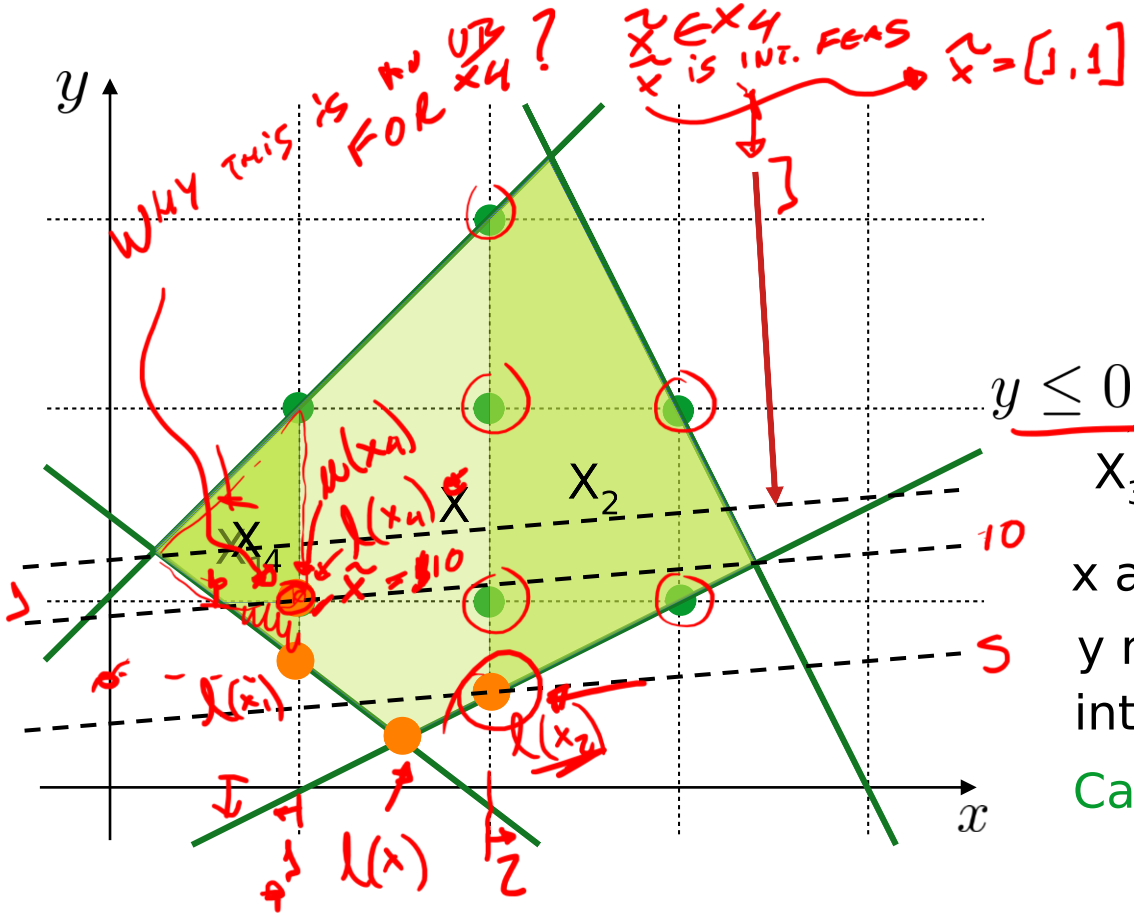
$x \in LINREX$
$x_j = 1.4$

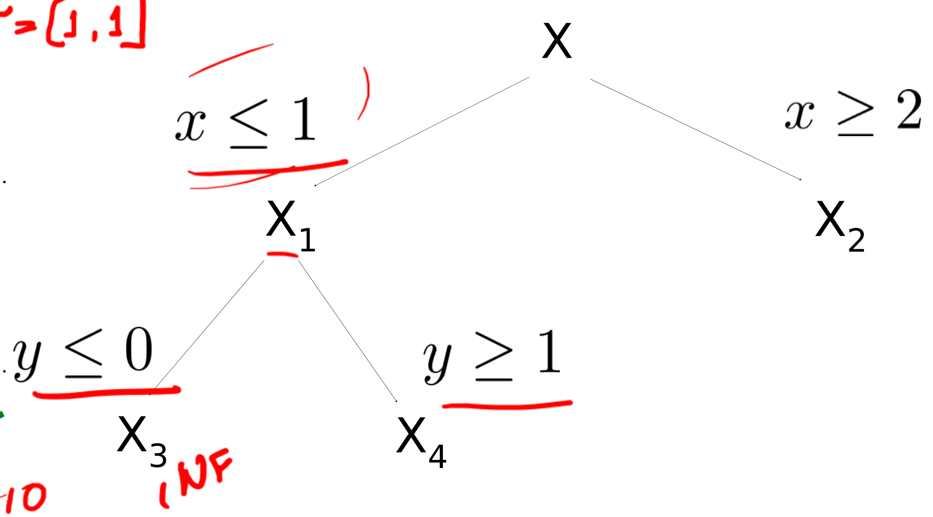
$X_5 = \{x \in X_2 | x_j \leq 1\}$ x_j

$X_6 = \{x \in X_2 | x_j \geq 2\}$

There can be different **branching / partitioning strategies**.
One example: if a variable $x_j = 1.4$ in the linear relaxation to X_2.

Branching Example





x and y not integral, branch on x

y not integral in X_1, branch on it

integer solution in X_4, X_3 infeasible

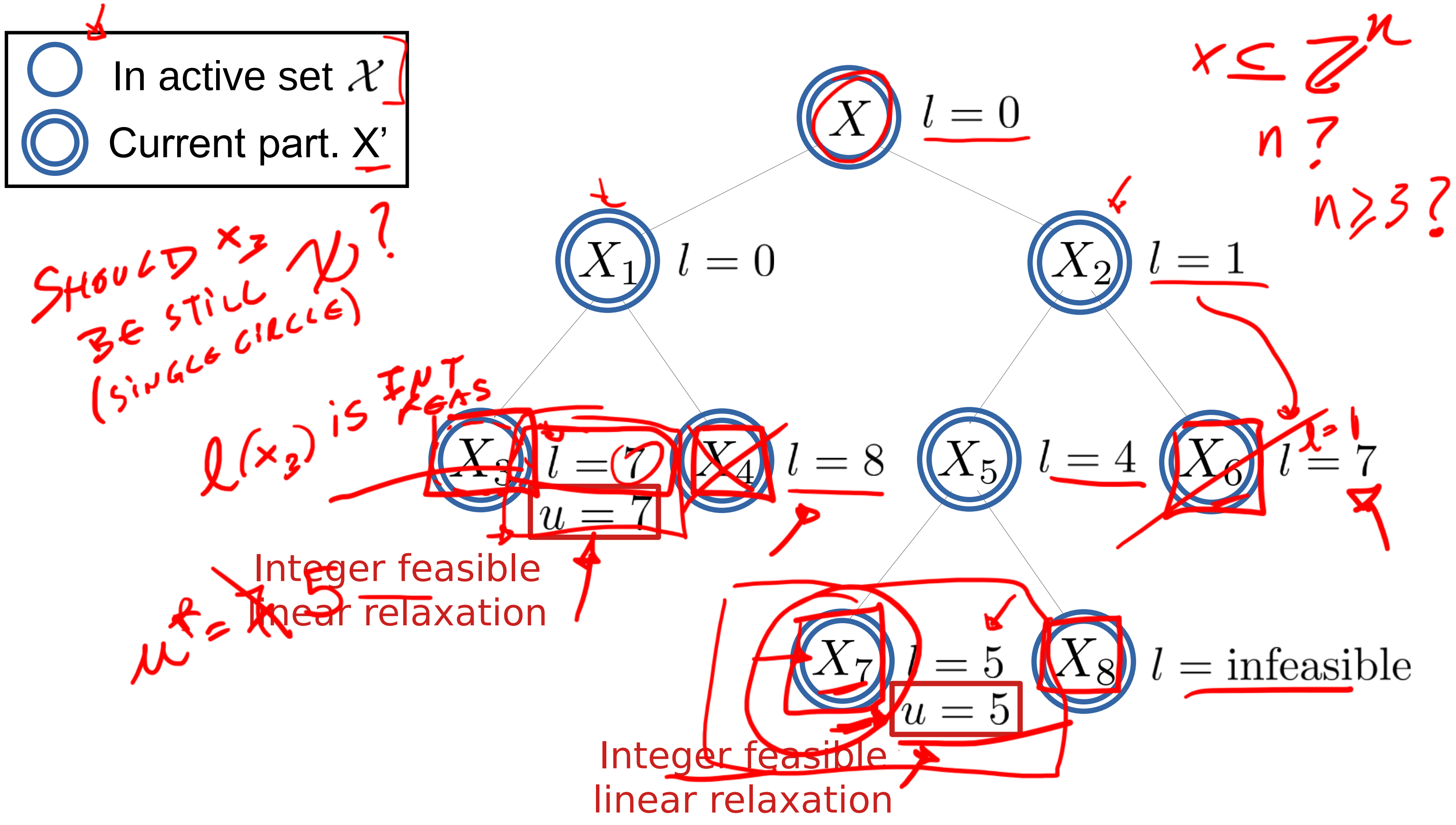
Can we stop here, what about X_2 ?

Branch and Bound Algorithm

```
$\mathcal{X} \leftarrow \{X\}$;                          A set of active nodes (active partitions).
$u^* \leftarrow \infty$;                             Best (lowest) feasible solution found so far.
while $\mathcal{X} \neq \{\}$ do   ← NON-DET
    $X' \leftarrow$ choose a partition to pop from $\mathcal{X}$; ←
    if $l(X') < u^*$ then                             If not then do nothing (prune this partition).
        either
            $u^* \leftarrow \min(u^*, u(X'))$;         Try to find tighter upper bound.
            Add $X'$ back into $\mathcal{X}$;            Partition may contain a better solution.
        or
            Branch at node, adding new partitions to $\mathcal{X}$; ←
    end
    IMPLICITLY PRUNE X' IF $l(X') \geq u^*$ ←
end
```

$l(x') == u(x')$

Branch and Bound Example



Optimality Gap

optimality gap $= u^* - \boxed{l^*}$

$$\frac{u^* - l^*}{u^*}$$

Best feasible solution found so far

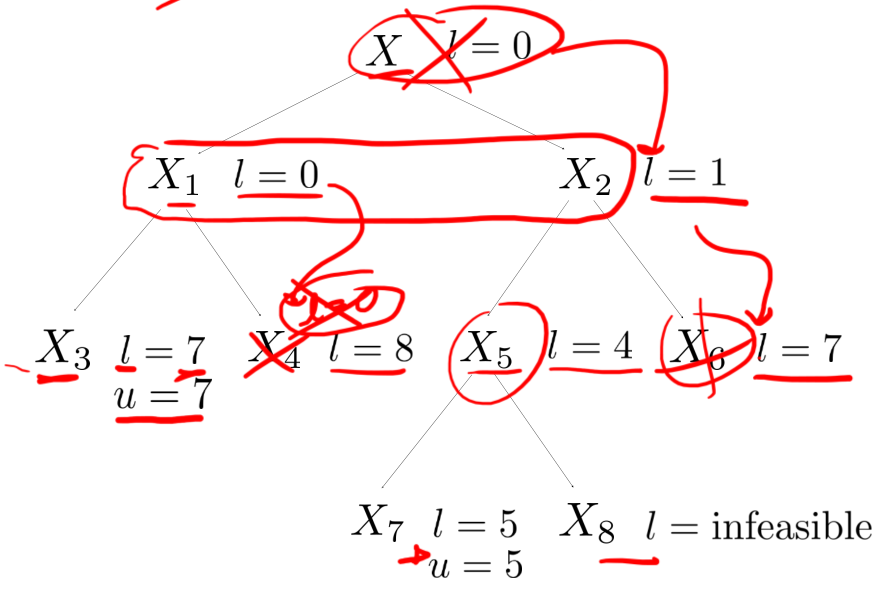
Weakest lower bound over open nodes (unexplored node takes parent LB), so long as there are open nodes:

$$l^* = \begin{cases} u^* & \text{if } \boxed{\mathcal{X} = \{\}} \\ \min_{X' \in \mathcal{X}} l(X') & \text{otherwise} \end{cases}$$

Let's calculate them for our example.

We might decide to stop early if we are happy with being at most a certain distance from optimal objective

$$\begin{array}{l} \boxed{u^*} - l^* \\ \infty - 0 = \infty \\ 7 - 0 = 7 \\ 7 - \boxed{1} = 6 \\ 5 - \boxed{1} = 4 \\ 5 - 5 = 0 \end{array}$$



Strategies

In order for this to work well in practice we need to come up with different strategies / heuristics for:

- **Choosing which open node** to explore (e.g., DFS vs BFS)
- Deciding when to focus on finding **upper vs lower bounds**
- How much time to devote to getting **stronger upper and lower bounds** vs branching
- How to **partition the feasible region** given a relaxation (e.g., what if multiple variables are fractional?)

Then there are the algorithms / heuristics for calculating stronger upper or lower bounds themselves.

Feasibility Heuristics

We don't need to find the optimal for a partition to get an upper bound, we **can use any feasible point.** Often will want to spend time using heuristics to find a good upper bound as it might reduce the branching we have to do.

These heuristics can be applied at the root node, and / or at intermediate nodes.

CBC AND ADD PART TO FEAS PUMP

BRANCH

The Feasibility Pump

x_i

Not guaranteed to work, so only try a fixed number of times

1. Solve the linear relaxation to get $\boxed{x^*}$
2. Round the elements of x^* to their nearest integer to get $\boxed{\tilde{x}}$
3. Resolve the LP but with objective: $\boxed{\sum_j |x_j - \tilde{x}_j|}$ CHEAP s.t "SAME CONSTR" $x \in X_i$ AS ORIG. PROG
4. If solution is integer return, else try again from 2 with updated x^*

$-z_j \le x_j - \tilde{x}_j \le z_j \; \forall j$ min $\sum z_j$

B&B Closing Remarks

- Need good upper and lower bounds to limit branching / space we need to search
- We get intermediate feasible points, and a measure of the optimality gap – we might decide to stop early

Scheduling

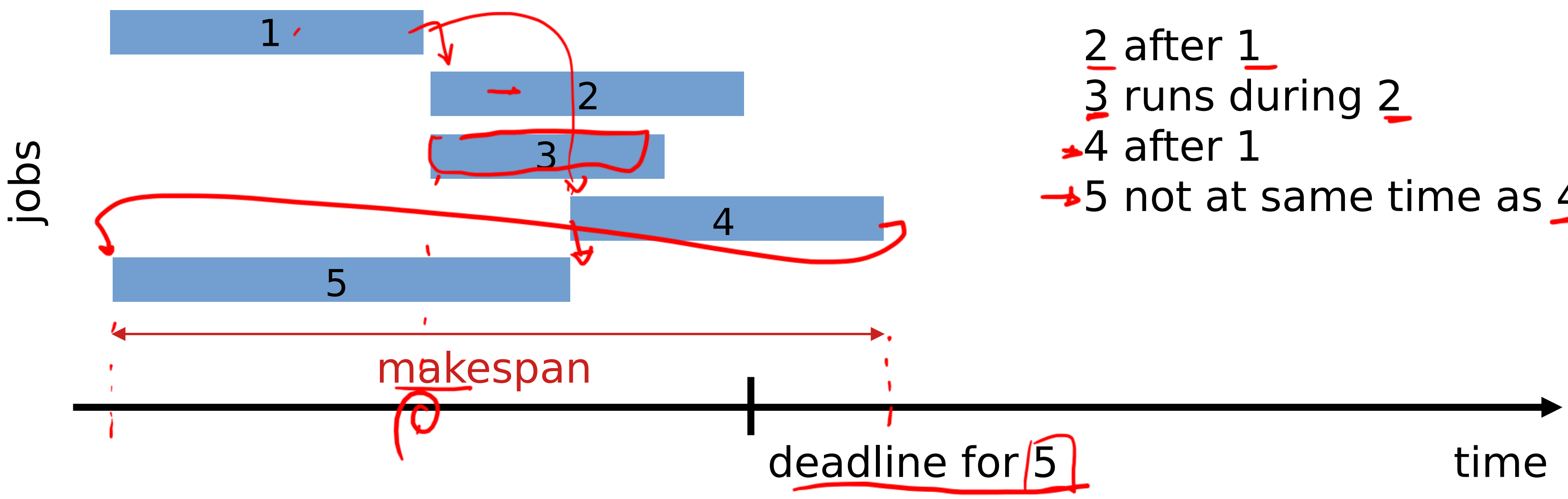
An import class of optimisation problems.

Ordering / arranging tasks, and allocating resources to them, e.g.:

- Job shop scheduling (we looked at a simple version)
- Project planning
- Staff scheduling
- Timetabling
- Jobs on compute cluster
- Processes across CPUs (very faster heuristics)

Could probably do a whole course on scheduling problems! Here we will explore some of the basic modelling decisions for scheduling problems.

Scheduling Constraints

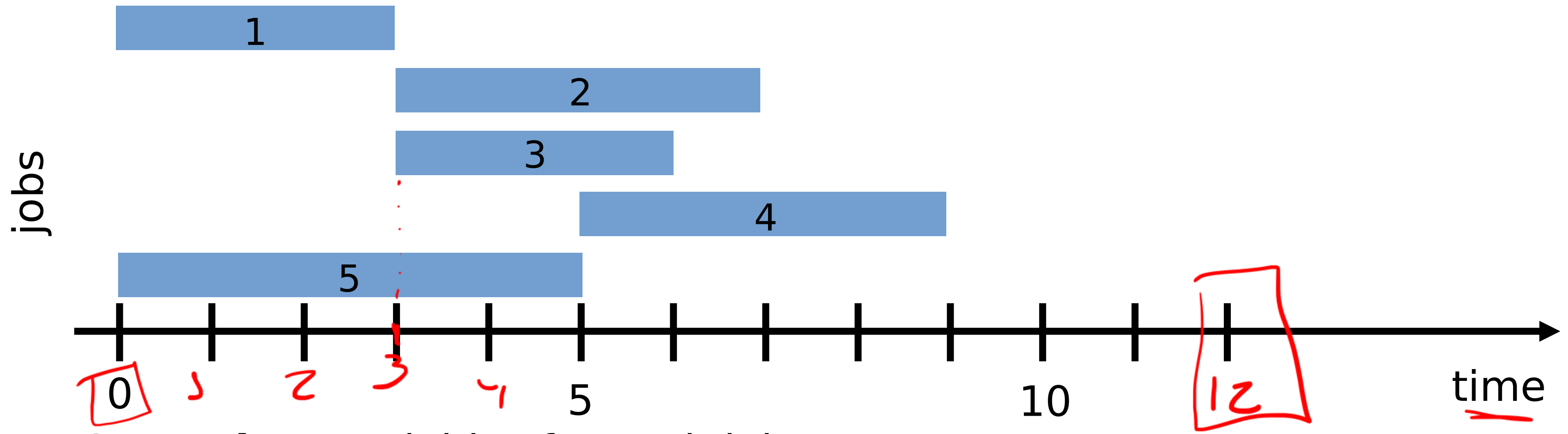


2 after 1
3 runs during 2
4 after 1
5 not at same time as 4

Some modelling choices:

- Variables that represent the **start time of each job,** or $x_3 =$
- **discretise time** and have **indicator variables** for each job and each time period, or $y_{i,t}$ .. $y_{3,5} = 1$
- some combination of both.

Scheduling Constraints



Start time variables for each job:

$\rightarrow (x_1, x_2, x_3, x_4, x_5) = (0, 3, 3, 5, 0)$ $\qquad \rightarrow x_j \in \mathbb{Z}$ or possibly $\mathbb{R}$

Discretise time, start **indicator variables** for each job:

JOB 1 $\rightarrow (y_{1,0}, y_{1,1}, \dots, y_{1,12}) = (1, 0, \dots, 0)$

3 $\qquad y_{3,3} = 1$

$\vdots$

$(y_{5,0}, y_{5,1}, \dots, y_{5,12}) = (1, 0, \dots, 0)$

$$\rightarrow y_{j,t} \in \{0, 1\}$$

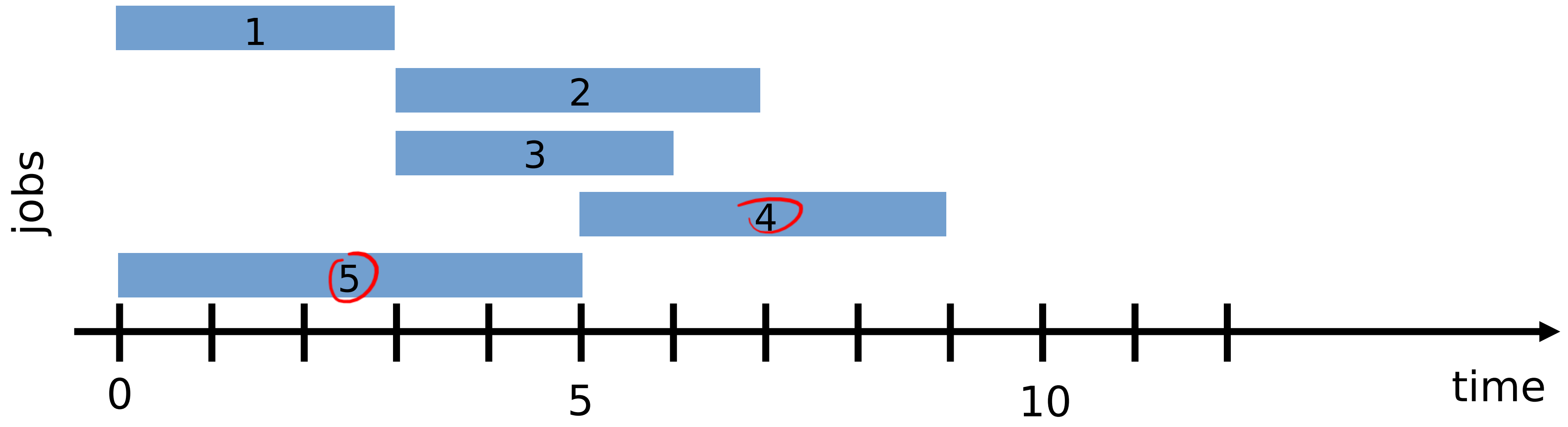
$j \to$ job, $t \to$ time

$$\boxed{\sum_t y_{j,t} = 1}$$

$\forall j$ ← JOB

TIME

Scheduling Constraints



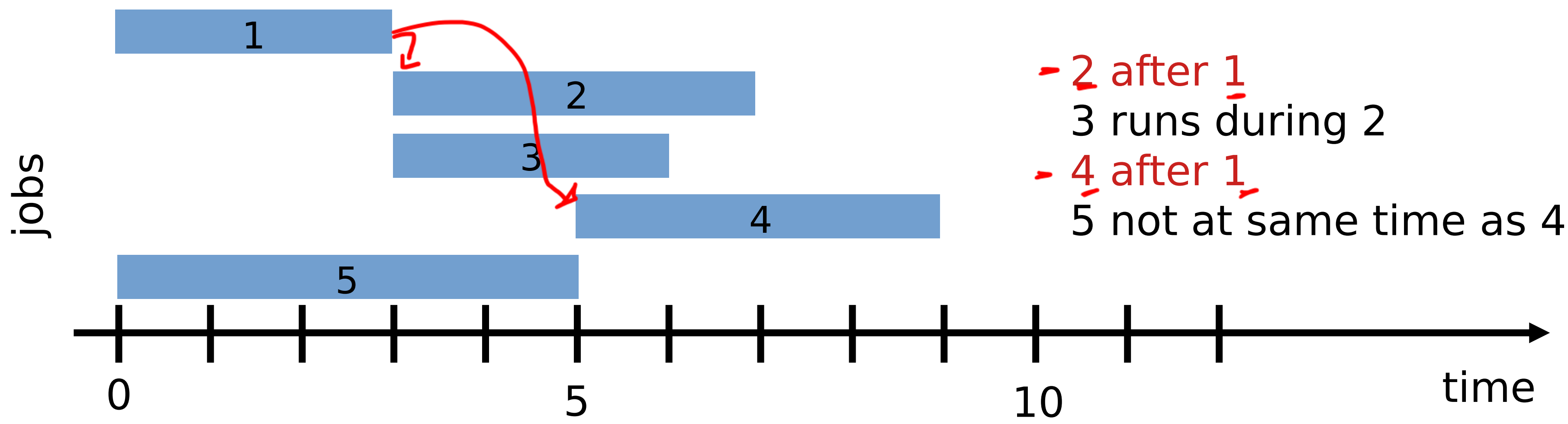
Time variables x_j

- Natural to model ordering constraints
- Time can be continuous
- Fewer variables
- Difficult to add resource or exclusivity constraints

Indicator variables $y_{j,t}$

- Natural to reason about what is happening at each point in time
- Need to choose discretisation
- Might not be obvious what the longest time we need to capture is

Scheduling Constraints



- 2 after 1
3 runs during 2
- 4 after 1
5 not at same time as 4

Time variables

DURATION

$$\underline{x_2} \geq \underline{x_1} + \underline{d_1}$$

$$\underline{x_4 \geq x_1 + d_1}$$

Indicator variables

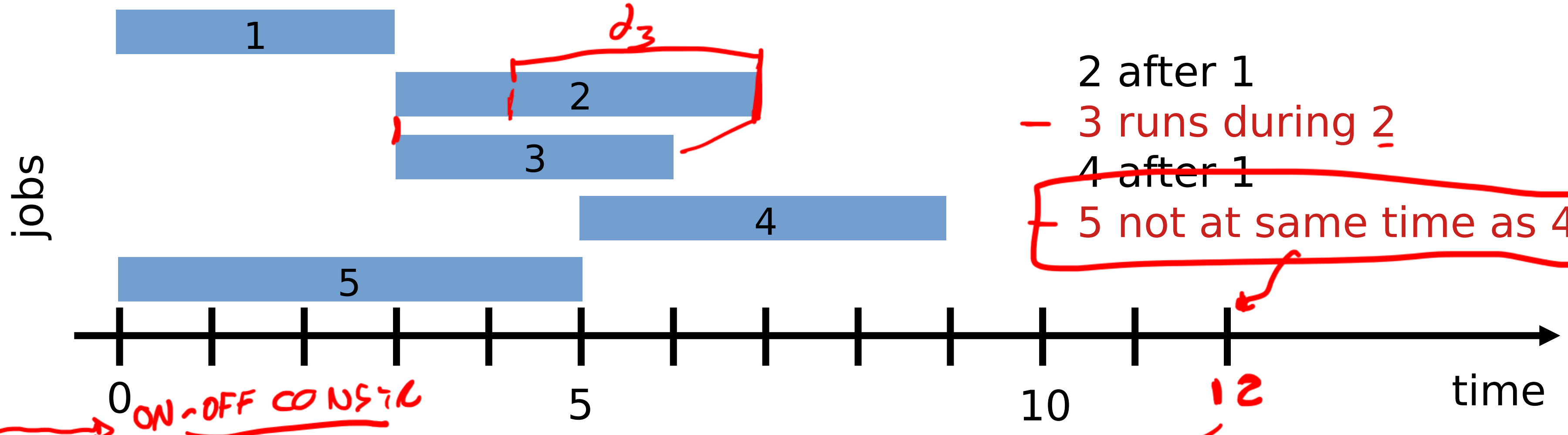
WHEN JOB 2 STARTS

$$\boxed{y_{2,t}} \leq \sum_{\tau=0}^{\boxed{t-d_1}} y_{1,\tau}, \quad \forall t \in \{0, \ldots, 12\}$$

OR EARLIER

$$y_{4,t} \leq \sum_{\tau=0}^{t-d_1} y_{1,\tau}, \quad \forall t \in \{0, \ldots, 12\}$$

Scheduling Constraints



2 after 1

3 runs during 2

4 after 1

5 not at same time as 4

ON-OFF CONSTR

Time variables

Deadline to start

Now we need bounds on time

$x_2 \le x_3 \le \boxed{x_2 + d_2 - d_3}$

$x_4 \le x_5 - d_4 \vee x_4 \ge x_5 + d_5$

$u = 0$ Before

$u = 1$ After

BIG-M

IF $u=1$

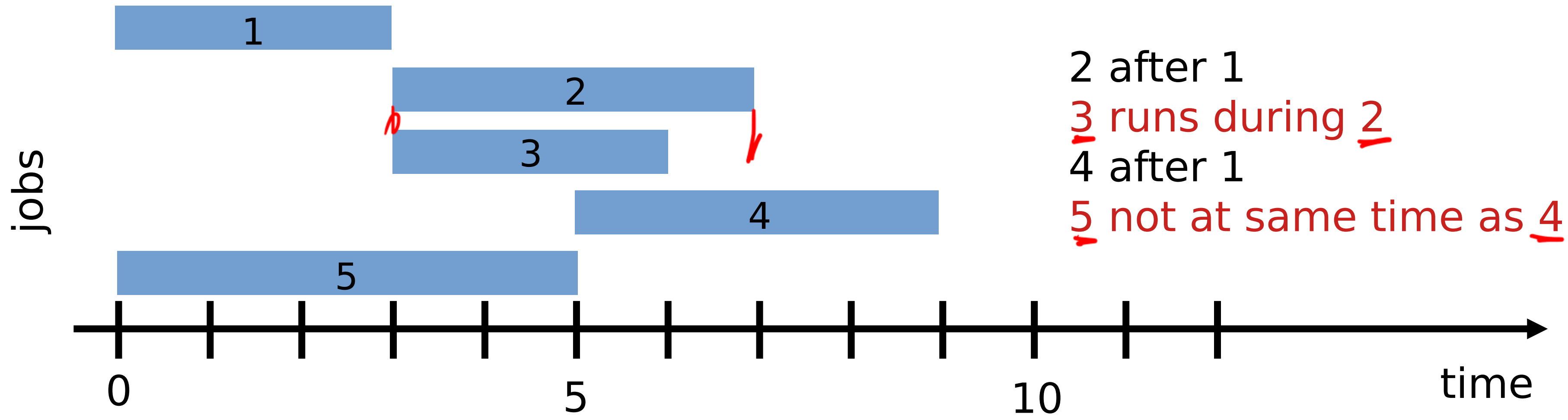
Constr active when:

$x_4 \le x_5 - d_4 + 12u$ x_4 before x_5

$x_4 \ge x_5 + d_5 - 12(1-u)$ x_4 after x_5

$\rightarrow x_4 + d_4 < x_5 + 12$

Scheduling Constraints



2 after 1
3 runs during 2
4 after 1
5 not at same time as 4

Indicator variables (directly with start indicator variables)

$$\underline{y_{3,t}} \le \sum_{\underline{\tau=t-d_2+d_3}}^{t} \boxed{y_{2,\tau}} \quad \forall t \in \{0,\ldots,12-d_3\}$$

← ENOUGH TIME TO FINISH

$$\boxed{\underline{y_{4,t}} \le 1 - \boxed{y_{5,\tau}}} \quad \forall \tau \in \{t-d_5+1,\ldots,t+d_4-1\}$$

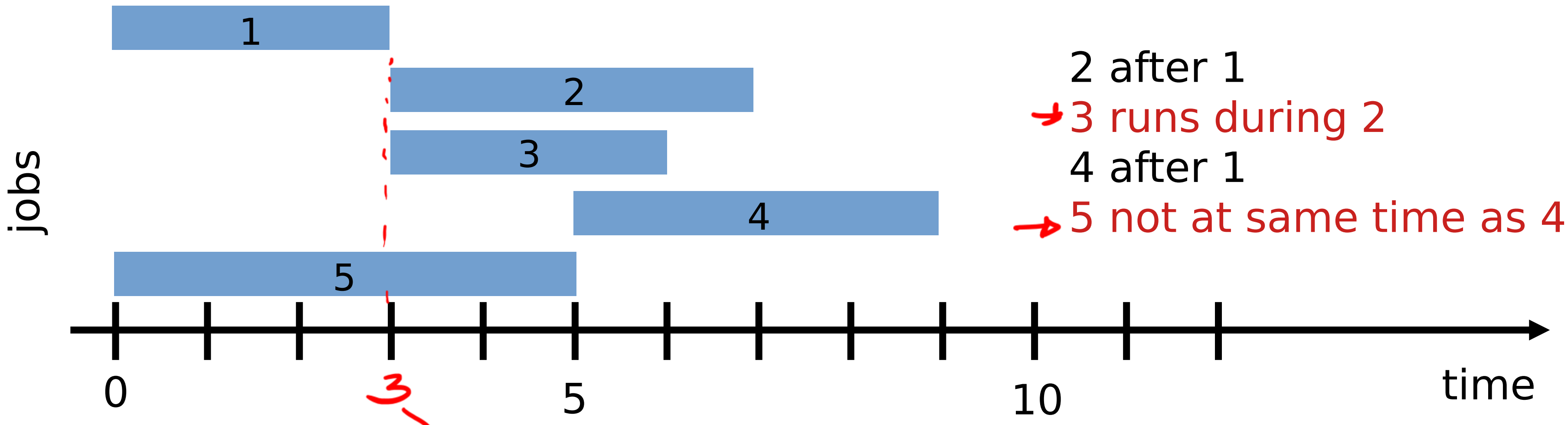
NEG

$$\rightarrow \forall t \in \{\underline{0},\ldots,\underline{12}-\underline{d_4}\}$$

CAN GROW QUICKLY

There is a better way!

Scheduling Constraints



2 after 1
3 runs during 2
4 after 1
5 not at same time as 4

Indicator variables (with new "running" indicator variables)

JOB 1 IS FINISH

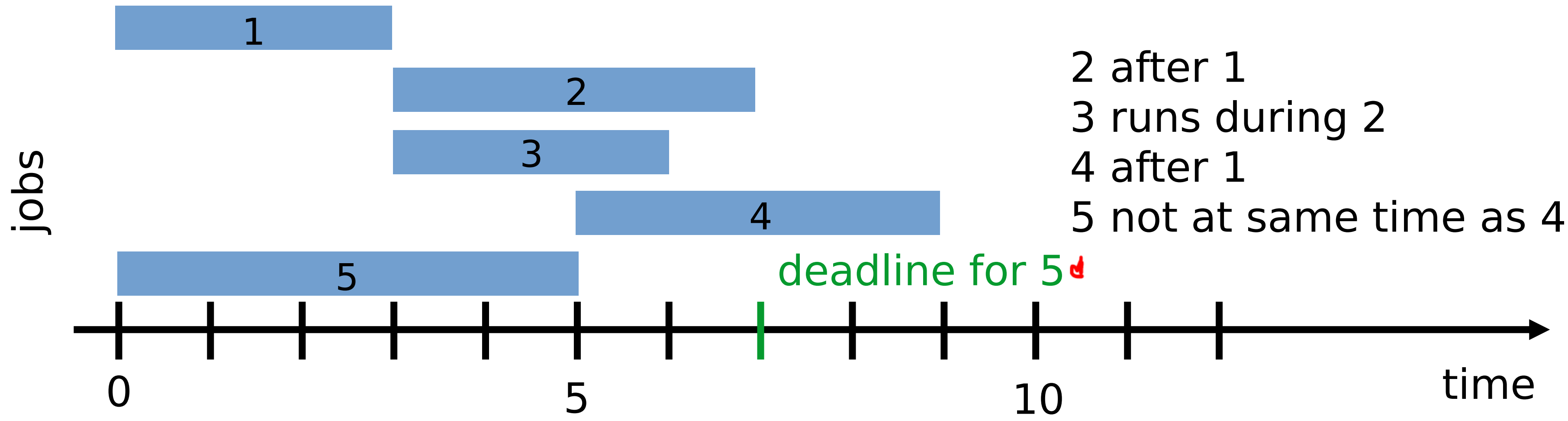
$$r_{j,t} \in \{0,1\}$$

$$r_{j,t} = \sum_{\tau=\boxed{t-d_j+1}}^{t} y_{j,\tau}$$

$$r_{3,t} \leq r_{2,t}, \quad \forall t$$

$$\boxed{r_{4,t} + r_{5,t}} \leq 1, \quad \forall t$$

Scheduling Constraints



2 after 1
3 runs during 2
4 after 1
5 not at same time as 4

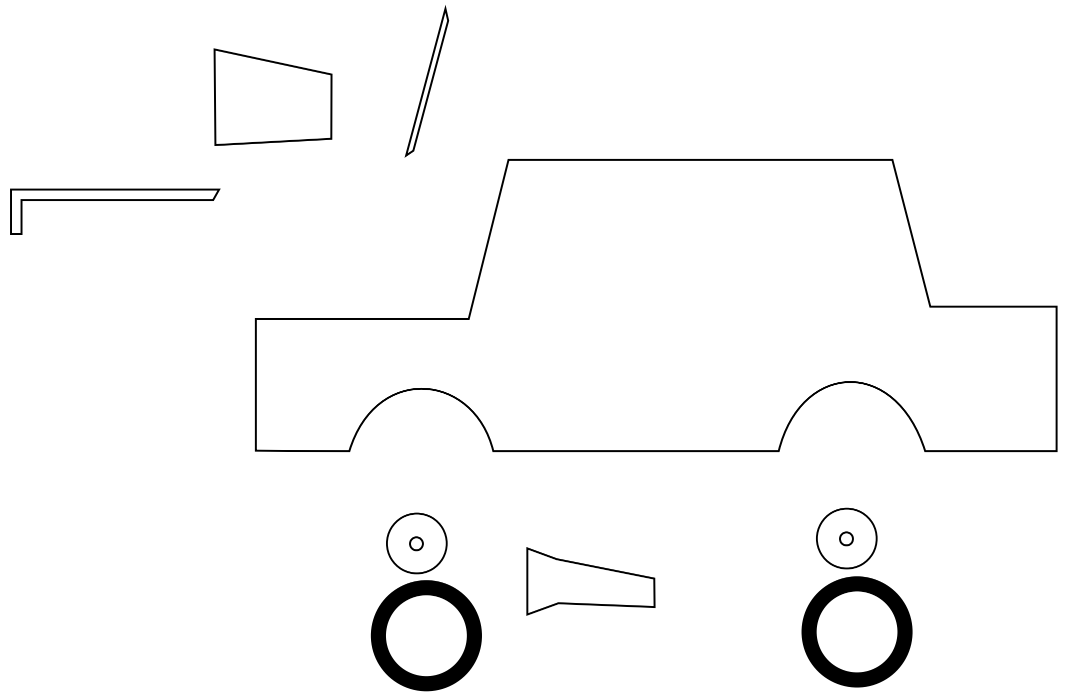
Time variables

$$\underline{x_5 + d_5} \leq 7$$

Indicator variables

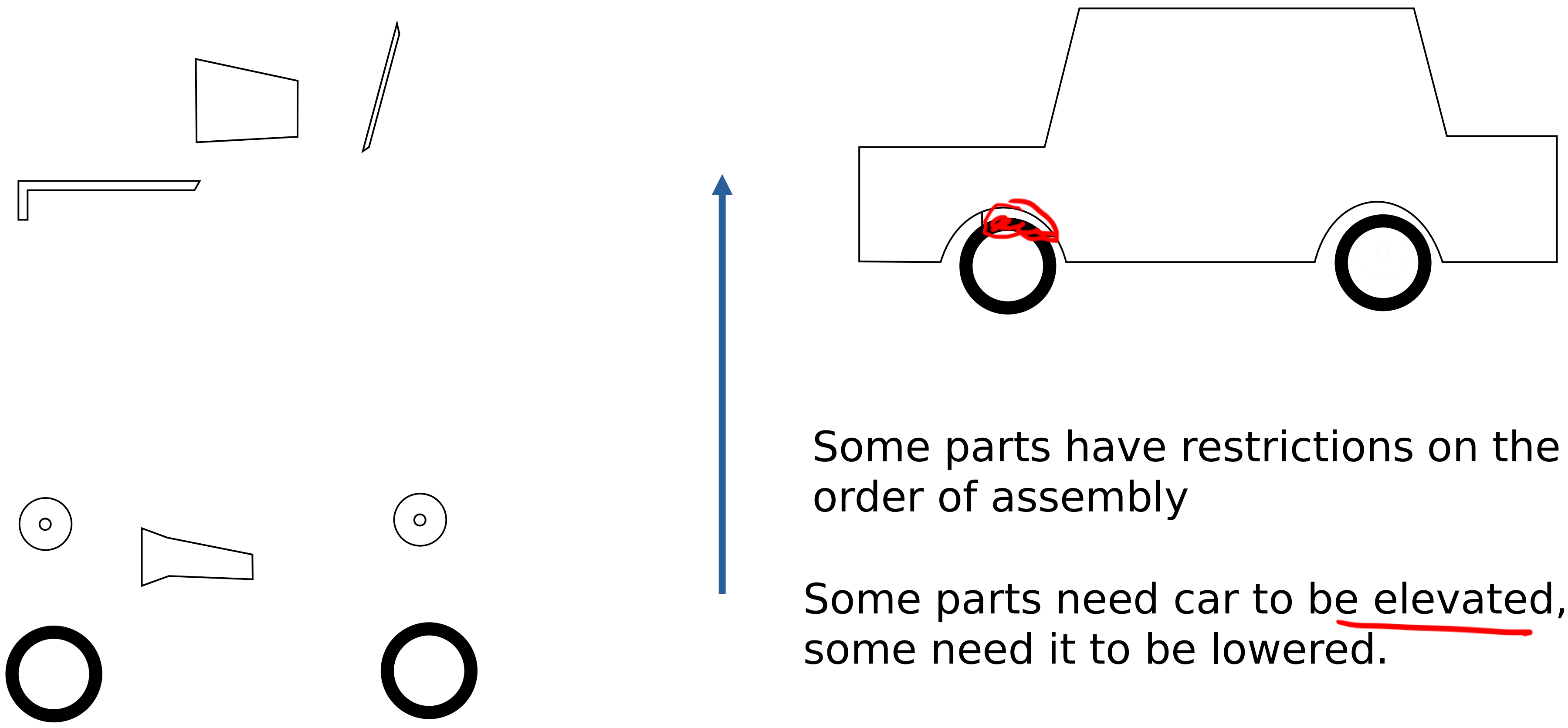
$$\underline{y_{5,t} = 0}, \quad \forall t \in \{\underline{7 - d_5 + 1, \ldots, 12}\}$$

Car Assembly Example



- Limited number of robotic arms
- Each part requires one or more arms for assembly
- Each part takes a different number of minutes to assemble

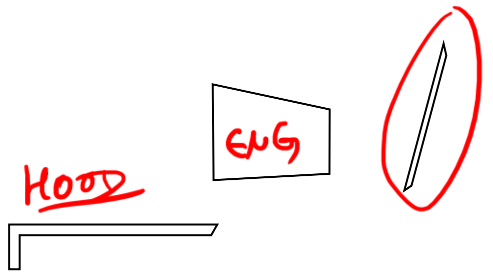
Car Assembly Example



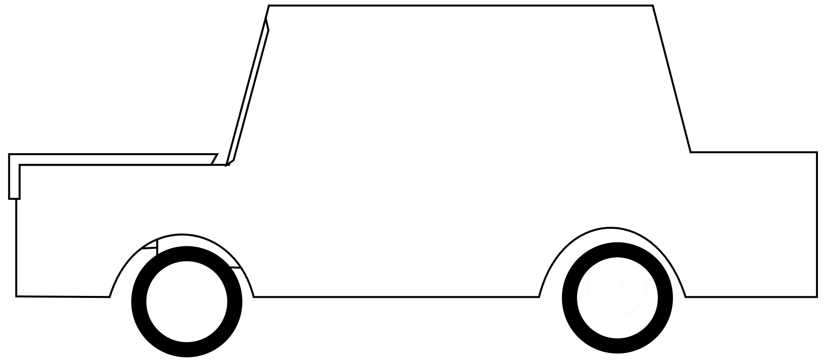
Some parts have restrictions on the order of assembly

Some parts need car to be elevated, some need it to be lowered.

Car Assembly Example



Some parts need access to the same space so cannot be assembled in parallel



Car Assembly

Discretise time into minutes.

For job j, time t:

$x_{j,t} \in \{0,1\}$ start indicator variables $\qquad \displaystyle\sum_t x_{j,t} = 1$

$r_{j,t} \in \{0,1\}$ running indicator variables $\qquad \displaystyle r_{j,t} = \sum_{\tau=t-d_j+1}^{t} x_{j,t}$ $\leftarrow$

$s_j \in \mathbb{Z}$ start time variables $\qquad \displaystyle s_j = \sum_t t x_{j,t}$

Car Assembly

You will work on the problem in the computer lab, so we are not going to provide much more detail here...

One aspect you will explore is the impact of a weaker and stronger formulation of the makespan of the problem:

$$m \in \mathbb{Z}$$

min m

If we only had start indicator variables:

$$m \geq t x_{j,t} + d_j, \quad \forall j, t$$

IF JOB 5 STARTS @ 5 AND $d_5 = 2$ $m \geq 7$

$\hookrightarrow 5 \boxed{x_{5,5}} \rightarrow J$

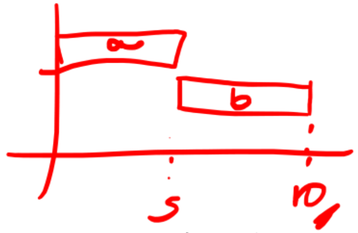
Which is stronger and why?

If we have start time variables:

$$m \geq \boxed{s_j} + d_j, \quad \forall j$$

$$s_j = \sum_t t x_{j,t} \longleftarrow \text{from previous slide}$$

Makespan Formulation



$$\rightarrow \quad \underline{m \geq t x_{j,t} + d_j}, \quad \forall j, t$$

vs

$$\left[\begin{array}{ll} m \geq s_j + d_j, \quad \forall j & s_j = \sum_t t x_{j,t} \\ m \geq \sum_t t x_{j,t} + \boxed{d_j}, \quad \forall j & \end{array}\right]$$

Consider a case where we have two jobs, a and b, each with duration 5, and the first job must complete first. What would be the optimal solution?

$\underline{x_{a,0} = 1}$ $\qquad x_{b,5} = 1$

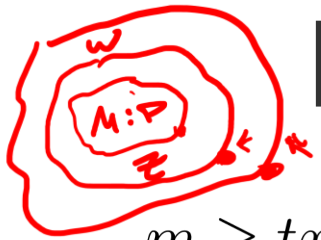
Now considering the makespan constraints for b:

$$\begin{array}{rl} & \vdots \\ & m \geq 4 \cdot \overset{0}{\cancel{x_{b,4}}} + 5 = 5 \\ t=5 \rightarrow & m \geq 5 \cdot \boxed{x_{b,5}} + \underline{5} = \underline{10} \\ & m \geq 6 \cdot x_{b,6} + 5 = 5 \\ & \vdots \end{array}$$

$$m \geq \ldots + 4 \cdot \overset{6}{\cancel{x_{b,4}}} + \overset{5 \times 1}{\boxed{5 \cdot x_{b,5}}} + 6 \cdot \underset{0}{\cancel{x_{b,6}}} + \ldots + \underline{5} = \boxed{10}$$

The same makespan, so far so good

Makespan Formulation



∃ INT FEAS SOL WITH M=10 (PREV SLIDE)

$$\underline{m \ge t x_{j,t} + d_j}, \quad \forall j,t \qquad \text{vs} \qquad m \ge \sum_t t x_{j,t} + d_j, \quad \forall j$$

W ←

Z ← FEAS REG OF LIN RELAX

What about when we take the linear relaxation? How can we check if one is stronger than the other?

If the feasible region for the linear relaxation is larger in one formulation (and contains the other), that formulation is weaker.

Let's see about $\boxed{x_{b,5} = 0.5, \; x_{b,6} = 0.5}$ ← LIN. RELAX

t=5
t=6

$$m \ge \overbrace{5 \cdot x_{b,5}}^{5\times 0.5} + 5 = 7.5 \leftarrow$$

$$m \ge \underbrace{6 \cdot x_{b,6}}_{6\times 0.5} + 5 = 8 \leftarrow \quad M \ge 8$$

$$m \ge \overset{0}{\cancel{\ldots + 4 \cdot x_{b,4}}} + \overbrace{5 \cdot x_{b,5}}^{5\times 0.5 = \boxed{2.5}}$$

$$\underset{\textcircled{3}}{6\times 0.5} \quad + \boxed{6 \cdot x_{b,6}} + \overset{6}{\cancel{\ldots}} + \boxed{5} = \underline{10.5}$$

It looks like the variable m can take on more values for the first formulation… we also see this directly leads to a weaker lower bound!

Image Attributions

- By The Sun in 1965 - http://www.abc.net.au/news/2018-10-08/meet-alison-harcourt-the-grandmother-of-australian-mathematics/10350170, Public Domain, https://commons.wikimedia.org/w/index.php?curid=73487846
- By LSE Library - https://www.flickr.com/photos/lselibrary/4437665957/, No restrictions, https://commons.wikimedia.org/w/index.php?curid=51989313
- By Jo Teichmann, Augsburg, Germany - KUKA Roboter GmbH, Public Domain, https://commons.wikimedia.org/w/index.php?curid=2076942