Links between variables

Outline (1/2)

How to read the chapter

Activation: aggregated and disaggregated form

Fixed cost, capacity and continuous flow

Minimum lot size and semicontinuous variables

Integer counts and rounding up

The maximum auxiliary variable

Min-max, max-min and the range

The absolute value

Outline (2/2)

Big-M: conditional constraints and disjunctions

Precedences and sequencing

"If and only if"

Counting the different types

Alldiff and binary expansion

Soft constraints, deviations and penalties

Piecewise linear functions

The map of techniques

Learning objectives (1/2)

At the end of this chapter you will be able to:

- link binary, integer and continuous variables with the fourteen techniques of the chapter;
- prove which direction of a link is imposed by the constraints and which follows from optimality;
- choose the smallest valid big-M from the data;

Learning objectives (2/2)

At the end of this chapter you will be able to:

- compare the aggregated and disaggregated forms by the strength of the relaxation;
- use the map of techniques to model a new link.

Prerequisites

- From this module, Chapter 5 «Logic and binary variables»: linear constraints for implications between binary variables and their proofs
- From this module, Chapter 2 «Relaxations, duality and bounds»: the comparison of formulations through their LP relaxations
- From the module *Integer Linear Programming*, Chapter 3 «Ideal formulations»: stronger formulations and the convex hull of the integer points

How to read the chapter

Introduction: links between variables

- The previous chapter links variables that are *all binary*
- Here we link *different* families: binary with continuous, binary with integer, continuous with each other
- There are **fourteen techniques**: the real content of MIP modelling
- Almost every model in this course is a combination of them

How to read each section

Every technique is presented in five steps:

(a) the **link in words**: what one wants to say

(b) the **linear constraints** that say it, with their count

(c) the **proof**: in both directions, with an exchange argument, or with a *counterexample*

(d) the **strength of the relaxation**, on a minimal instance

(e) the `gurobipy` **line** that writes it in the solver

Step (c) tells a correct model from a plausible one.
Step (d) tells a model that can be solved from one that cannot.

Two properties never to be confused

Observation 1: Two properties never to be confused

- *A property **imposed by the constraints**: it holds for every feasible solution, optimal or not. It is proved by looking at the constraints alone*
- *A property **of optimality**: it holds only in optimal solutions. It is proved by an exchange argument*

The exchange argument

- Take a feasible solution that violates the property
- Build a modified solution
- Check that it stays feasible and compare the values

- If the value improves *strictly*: "in every optimum"
- If it merely does not worsen: "there exists an optimum", which is weaker

Writing "in every optimum" when the coefficient is only non-negative is the commonest mistake in this chapter.

Summary

- Every technique: words, constraints, proof, relaxation, `gurobipy`
- **Imposed by the constraints**: holds in every feasible solution
- **Of optimality**: proved by the exchange argument; strict improvement $\Rightarrow$ "in every optimum", otherwise only "there exists an optimum"

Activation: aggregated and disaggregated form

Introduction: activation

- A resource (a machine, a site, a plant) must be *activated* before being used
- The binaries x_{ij} say "object i uses resource j"
- The binaries y_j say "resource j is activated"
- We want: **if anyone uses j, then j is activated**

The constraints

Two ways, with n objects and m resources:

Disaggregated ($n\,m$ constraints):

$$x_{ij} \leq y_j, \qquad \forall i \in \{1, 2, \ldots, n\},\ \forall j \in \{1, 2, \ldots, m\}$$

Aggregated (m constraints):

$$\sum_{i=1}^{n} x_{ij} \leq n\, y_j, \qquad \forall j \in \{1, 2, \ldots, m\}$$

- With a capacity k_j (at most k_j objects): $\sum_i x_{ij} \leq k_j y_j$
- Then it imposes the capacity *as well*: two conditions in one constraint

The proof

On binary points the two forms are **equivalent**.

- *Disaggregated* $\Rightarrow$ *aggregated*: summing $x_{ij} \leq y_j$ over i gives $\sum_i x_{ij} \leq n y_j$
- *Aggregated* $\Rightarrow$ *disaggregated, on binary points*:
  - if $y_j = 0$: $\sum_i x_{ij} \leq 0$, hence $x_{ij} = 0 \leq y_j$ for every i
  - if $y_j = 1$: $x_{ij} \leq 1 = y_j$ because the x are binary

The second direction uses binarity: *outside* the binary points it does not hold, and that is what makes the two forms different in the relaxation.

The converse is not imposed

- "If j is activated then somebody uses it": $y_j = 1 \Rightarrow \sum_i x_{ij} \geq 1$
- It does **not** follow from the constraints: $y_j = 1$ with all $x_{ij} = 0$ is feasible in both forms
- It follows from optimality if the activation cost f_j is *strictly* positive:
  - an optimum with $y_j = 1$ and no $x_{ij} = 1$
  - setting $y_j = 0$ keeps all constraints satisfied and reduces the cost by $f_j > 0$: a contradiction
  - hence "in every optimum"
- If $f_j = 0$ is allowed: only "there exists an optimum with $y_j = 0$"

The strength of the relaxation

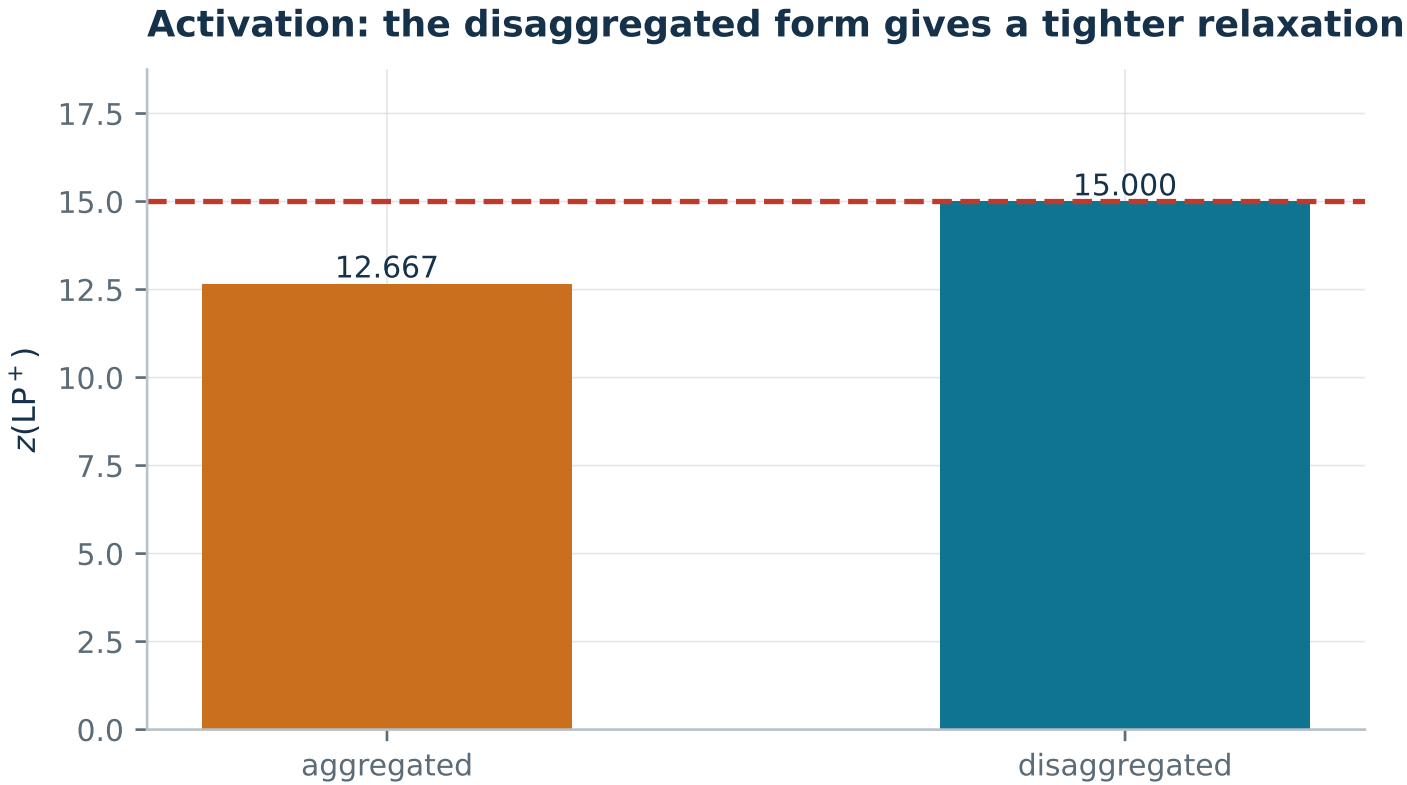
Instance: $n = 3$ customers, $m = 2$ sites, activation costs $f = (8, 6)$, service costs

$$c = \begin{pmatrix} 2 & 5 \\ 4 & 1 \\ 3 & 3 \end{pmatrix}$$

Formulation	link constraints	$z(LP^+)$
aggregated	$m = 2$	$38/3 \approx 12.67$
disaggregated	$n\,m = 6$	15

The integer optimum is $z(MILP) = 15$.

The relaxation of the two forms



The red line is the integer optimum.

More rows, tighter relaxation

- The disaggregated form already reaches the integer optimum in the relaxation
- The aggregated one stops at 38/3
- This is the typical trade-off of this technique
- On large instances the choice is not obvious ($n\,m$ rows can be many), but on hard instances the disaggregated form almost always pays off

In gurobipy

```
# disaggregated
m.addConstrs((x[i, j] <= y[j]
              for i in range(n) for j in range(mm)),
             name="link")
# aggregated
m.addConstrs((x.sum("*", j) <= n * y[j]
              for j in range(mm)), name="link")
```

Summary

- Activation: disaggregated $x_{ij} \leq y_j$ ($n\,m$ constraints) or aggregated $\sum_i x_{ij} \leq n y_j$ (m constraints)
- Equivalent on binary points, different in the relaxation: the disaggregated form is tighter
- "Activated $\Rightarrow$ used" is not imposed: it follows from optimality only if $f_j > 0$

Fixed cost, capacity and continuous flow

Introduction: fixed cost

- As in activation, but the quantity used is *continuous*
- $q_j \geq 0$ is how much is produced at plant j
- y_j says whether the plant is open
- A closed plant produces nothing; an open one produces at most its capacity c_j

The constraints and the proof

$$q_j \leq c_j\, y_j, \qquad \forall j \in \{1, 2, \ldots, m\} \qquad (m \text{ constraints})$$

One constraint per plant, with the *right* coefficient: the capacity, not some large number picked at random.

- $y_j = 0$: $q_j \leq 0$ and, with $q_j \geq 0$, $q_j = 0$. "Closed $\Rightarrow$ produces nothing" is imposed, and with it "produces $\Rightarrow$ open"
- $y_j = 1$: $q_j \leq c_j$, the capacity
- "Open $\Rightarrow$ produces" is not imposed: it follows from optimality only if $f_j > 0$

The strength of the relaxation

Instance: two plants, fixed costs $f = (10, 14)$, unit costs $c = (3, 2)$, capacities $C = (6, 7)$, demand $D = 9$.
The optimum is $z(MILP) = 44$ (both open, $q = (2, 7)$).

Coefficient in the constraint	$z(LP^+)$
the capacity, c_j	$112/3 \approx 37.33$
big-M $= 100$ (plus $q_j \leq c_j$ separately)	$1059/50 = 21.18$

Same integer set, same optimum, relaxations far apart.

The coefficient rule

The coefficient of an activation binary is the smallest value the continuous variable can be capped at when the activation equals 1, and it must be derived from the data.

In gurobipy

```
m.addConstrs((q[j] <= c[j] * y[j]
              for j in range(mm)), name="link")
```

Summary

- Fixed cost: $q_j \leq c_j y_j$, one constraint per plant
- "Closed $\Rightarrow$ produces nothing" is imposed; "open $\Rightarrow$ produces" only by optimality with $f_j > 0$
- The coefficient is the capacity, derived from the data: a large big-M gives the same optimum but a much weaker relaxation

Minimum lot size and semicontinuous variables

Introduction: minimum lot size

- "If you produce, produce at least ℓ"
- The quantity q_j is either zero or lies between a threshold ℓ and the capacity c_j
- It is not an interval: it is the union of a point and an interval
- A variable with this domain is called **semicontinuous**; when it is also integer (pieces, units, people) it is called **semi-integer**, and the constraints are the same

The constraints and the proof

With the same binary y_j as in the previous section:

$$\ell\, y_j \le q_j, \qquad \forall j \in \{1, 2, \ldots, m\} \qquad (m \text{ constraints})$$

$$q_j \le c_j\, y_j, \quad \forall j \in \{1, 2, \ldots, m\} \qquad (m \text{ constraints})$$

Both directions are imposed by the constraints:

- $y_j = 0$: $0 \le q_j \le 0$, that is $q_j = 0$
- $y_j = 1$: $\ell \le q_j \le c_j$

$$q_j \in \{0\} \cup [\ell,\ c_j]$$

One needs $\ell \leq c_j$

- If $\ell > c_j$, $y_j = 1$ is infeasible and the variable is forced to zero
- It is a data error
- The solver reports it as infeasibility only if y_j is forced to 1 by other constraints

The strength of the relaxation

- Same instance as in the previous section, with $\ell = 5$
- The optimum goes from 44 to $z(MILP) = 49$, with $q = (5, 5)$ instead of $(2, 7)$
- The threshold forces production of 5 at the second plant even though the first is more expensive
- But $z(LP^+)$ stays at $112/3$, identical to the case without the threshold

Why the minimum lot is invisible in the relaxation

Observation 2: Why the minimum lot is invisible in the relaxation

In the relaxation y_j is free in $[0, 1]$, and the constraint $q_j \geq \ell y_j$ is satisfied by lowering y_j: $y_j \leq q_j/\ell$ suffices. The threshold constraint never bites on the continuous problem. All of its effect is discharged onto integrality: this is why models with minimum lot sizes are typically harder than those without, at equal size.

In gurobipy

```
m.addConstrs((q[j] >= ell * y[j]
              for j in range(mm)), name="lot")
m.addConstrs((q[j] <= c[j] * y[j]
              for j in range(mm)), name="capacity")
```

- Gurobi also has the type `GRB.SEMICONT`, which declares the domain directly
- In this course the formulation is written by hand, because that is what one must be able to prove

Summary

- Minimum lot: $\ell y_j \leq q_j \leq c_j y_j$, that is $q_j \in \{0\} \cup [\ell, c_j]$
- Both directions imposed by the constraints; one needs $\ell \leq c_j$
- The threshold does not bite in the relaxation: all of its effect is on integrality

Integer counts, multiple capacity, rounding up

Introduction: integer counts

- "How many containers are needed?"
- The variable is not binary but *integer*: $w \in \mathbb{Z}_{\geq 0}$
- It counts indivisible objects (boxes, trucks, shifts, workers)
- Each one carries a capacity c

The constraints and the proof

$$\sum_i a_i x_i \leq c\, w \qquad \text{(1 constraint)}$$

$$w \in \mathbb{Z}_{\geq 0} \qquad \text{(1 integer variable)}$$

- Setting $q = \sum_i a_i x_i$: the constraint imposes $w \geq q/c$, integrality imposes $w \geq \lceil q/c \rceil$
- With an objective that minimises (or pays for) w, in every optimum w equals exactly that ceiling: if it were larger, lowering it by 1 would stay feasible and reduce the cost
- With a zero cost on w: only "there exists an optimum"

The ceiling is not written with $\lceil \cdot \rceil$

Observation 3: The ceiling is not written with $\lceil \cdot \rceil$

$\lceil t \rceil$ is not a linear function: it cannot be written inside a constraint. The pair "inequality $\leq$ cw + declaration that w is integer" realises it implicitly, and that is how it must be read when explaining the model.

The strength of the relaxation

- 17 unit items, capacity $c = 5$
- The relaxation gives $w \geq 17/5 = 3.4$
- The integer optimum is $z(MILP) = 4$
- The gap $4 - 17/5 = 3/5$ comes entirely from integrality: no linear cut on the x alone closes it, an inequality using w integer is needed

In gurobipy

```
w = m.addVar(vtype=GRB.INTEGER, name="w")
m.addConstr(gp.quicksum(a[i] * x[i]
                        for i in range(n)) <= c * w,
            name="capacity")
```

Summary

- Integer count: $\sum_i a_i x_i \leq c\, w$ with w integer
- Integrality implicitly realises the ceiling $\lceil q/c \rceil$; equality holds at the optimum if w has a cost
- The gap comes entirely from the integrality of w

The maximum auxiliary variable

Introduction: the maximum auxiliary variable

- A variable z must equal the *maximum* of certain quantities
- Examples: the duration of the longest activity, the highest connection cost, the load of the busiest machine
- We will see that half of the link is imposed by the constraints and half comes from optimality

The constraints

If the quantities are $t_j x_j$ with x_j binary:

$$z \geq t_j \, x_j, \quad \forall j \in \{1, 2, \ldots, n\} \qquad (n \text{ constraints})$$

$$z \geq 0$$

The proof, in three steps

1. *Imposed by the constraint*: for every j with $x_j = 1$ we get $z \geq t_j$, hence $z \geq \max_{j:x_j=1} t_j$, in every feasible solution
2. *From optimality*: if z is in the objective with a strictly positive coefficient in a minimisation (or strictly negative in a maximisation) and in no other constraint, lowering z to the maximum keeps the constraints satisfied and strictly improves the objective
3. *Conclusion*: in every optimum $z = \max_{j:x_j=1} t_j$ (the maximum over the empty set is 0, guaranteed by $z \geq 0$)

Step 2 fails if z is used elsewhere

Observation 4: Step 2 fails if z is used elsewhere

If z appears in another constraint that wants it large (for instance $z \geq w$, with w another variable) the exchange argument no longer works: lowering z may violate that constraint. In that case $z \geq \max$ still holds, but $z = \max$ does not. If the maximum of two different families is needed, two auxiliaries are needed.

The strength of the relaxation

- $\min z$ with $z \geq t_j x_j$, $t = (4, 7, 3)$ and $\sum_j x_j \geq 2$
- Integer optimum $z(MILP) = 4$: jobs 1 and 3 are chosen
- The relaxation is $168/61 \approx 2.75$: the fractional solution $x_j = z/t_j$ spreads the choice over all three jobs and lowers the maximum
- The maximum link gives *weak* relaxations: one of the reasons makespan problems are hard

In gurobipy

```
z = m.addVar(name="z")
m.addConstrs((z >= t[j] * x[j] for j in range(n)),
             name="maximum")
```

Summary

- Maximum auxiliary: $z \geq t_j x_j$, $z \geq 0$
- $z \geq \max$ is imposed; $z = \max$ holds only at the optimum, if z has the right sign in the objective and appears nowhere else
- Weak relaxations: makespan problems are hard

Min-max, max-min and the range

Introduction: fairness objectives

Three fairness objectives, often confused:

- **min-max**: minimise the largest load
- **max-min**: maximise the smallest load
- **range**: minimise the gap between the largest and the smallest

The constraints

With ℓ_k the load of resource $k \in \{1, 2, \ldots, m\}$:

min-max (m constraints):

$$\min z \quad \text{with} \quad z \geq \ell_k, \ \forall k \in \{1, 2, \ldots, m\}$$

max-min (m constraints):

$$\max u \quad \text{with} \quad u \leq \ell_k, \ \forall k \in \{1, 2, \ldots, m\}$$

range ($2m$ constraints):

$$\min (z - u) \quad \text{with both}$$

The proof

- Each is the maximum (or minimum) auxiliary variable of the previous section
- The exchange argument goes in the right direction:
  - in a $\min z$ the variable z falls to the maximum of the loads
  - in a $\max u$ the variable u rises to the minimum of the loads
- In the range both pressures are present at once and the two conclusions hold together

The three objectives are not comparable

Observation 5: The three objectives are not comparable

Five weights $p = (3, 5, 2, 4, 7)$ *split between two workers (total* 21*): the three versions choose the same split* $(11, 10)$*, the best possible because the total is odd, but the optimal values are* 11*,* 10 *and* 1*. Comparing "*$z = 11$*" of a min-max with "*$z = 1$*" of a range means nothing.*

- Before comparing two models one must make sure they optimise the same thing
- With more than two resources, min-max and max-min in general choose different splits

The strength of the relaxation and gurobipy

- The min-max on the five-weight instance gives $z(LP^+) = 21/2 = 10.5$ against $z(MILP) = 11$
- The relaxation splits the weights exactly in half, which integrality does not allow

```
z = m.addVar(name="z")
m.addConstrs((z >= load[k]
              for k in range(len(load))), name="max")
m.setObjective(z, GRB.MINIMIZE)
```

Summary

- Min-max: $\min z$, $z \geq \ell_k$; max-min: $\max u$, $u \leq \ell_k$; range: $\min(z - u)$ with both
- They are the maximum/minimum auxiliary with the exchange in the right direction
- The optimal values of the three objectives are not comparable

The absolute value

Introduction: the absolute value

- One wants $|u - v|$: a gap, an error, an imbalance
- The case in the objective and the case in a constraint behave radically differently
- We will see three cases: in the objective, as a $\leq$ constraint, as a $\geq$ constraint

The constraints (1/2): objective and $\leq$ constraint

In the objective, minimising: a variable $d \geq 0$ and two constraints

$$d \geq u - v \qquad \text{(1 constraint)}$$

$$d \geq v - u \qquad \text{(1 constraint)}$$

with d in the objective to be minimised. No binary.

As a $\leq$ constraint: $|u - v| \leq k$ is simply

$$u - v \leq k \quad \text{and} \quad v - u \leq k \qquad \text{(2 constraints, no binary)}$$

The constraints (2/2): $\geq$ constraint

- $|u - v| \geq k$ **cannot** be written without binaries
- It is the disjunction "$u - v \geq k$ *or* $v - u \geq k$"
- It needs a binary b and a big-M:

$$u - v \geq k - M(1 - b) \qquad \text{(1 constraint)}$$

$$v - u \geq k - M\,b \qquad \text{(1 constraint, 1 binary)}$$

The proof

- **In the objective**: the two constraints impose $d \geq |u - v|$ (one of the two right-hand sides *is* $|u - v|$)
- The objective minimises d, which appears nowhere else: the exchange argument of the maximum auxiliary drives it to equality
- **As a $\geq$ constraint**: $b = 1$ switches off the second constraint (provided $M \geq k + \max(v - u)$) and leaves the first, and vice versa
- It is a disjunction, not a conjunction: without the binary both would be imposed, that is $0 \geq 2k$, infeasible for $k > 0$

The $\geq$ case is not symmetric to the $\leq$ case

Observation 6: The $\geq$ case is not symmetric to the $\leq$ case

$|u - v| \leq k$ is the intersection of two half-planes: a convex set, written with two linear constraints. $|u - v| \geq k$ is the complement of a strip: it is not convex, and no system of linear constraints without integer variables can describe it. The binary is not a trick: it is necessary.

The strength of the relaxation and gurobipy

- Five-weight instance: min $|\ell_1 - \ell_2|$ has $z(MILP) = 1$ and $z(LP^+) = 0$
- The continuous problem splits 21 into two equal halves and zeroes the gap: the relaxation of an absolute-value objective is typically 0, that is, useless

```
d = m.addVar(name="d")
m.addConstr(d >= u - v, name="abs_plus")
m.addConstr(d >= v - u, name="abs_minus")
```

Summary

- In the objective (minimising): $d \geq u - v$, $d \geq v - u$, no binary
- $|u - v| \leq k$: two linear constraints; $|u - v| \geq k$: a disjunction, a binary is needed
- The relaxation of an absolute-value objective is typically 0

Big-M: conditional constraints and disjunctions

Introduction: big-M

- "If $y = 1$ then the constraint $a'x \leq b$ holds; if $y = 0$ the constraint is not there"
- A constraint that switches on and off
- The delicate point is the choice of M: both the correctness of the model and the strength of the relaxation depend on it

The constraint and the proof

$$a'x \leq b + M(1 - y) \qquad (1 \text{ constraint})$$

- With $y = 1$: $a'x \leq b$, switched on
- With $y = 0$: $a'x \leq b + M$, switched off *provided* M cuts nothing off:

  $M \geq \max\{a'x : x \text{ feasible for the other constraints}\} - b$

How M is chosen

Three things must be told apart:

- a **valid** M: it satisfies the condition, so the model has the right integer set
- an **improvable** M: valid but larger than necessary; same integer set, weaker relaxation
- the **smallest proved** M: the smallest value for which validity has been *proved*, typically by computing the maximum from the variable bounds alone. It is not necessarily the absolute minimum, and it is honest to say so

An invalid M is not "a little different"

Observation 7: An invalid M is not "a little different"

With $a = (3, 4, 5)$, $b = 6$ and x binary, the maximum of $a'x$ is 12, so $M = 12 - 6 = 6$ is valid. On max $x_1 + x_2 + x_3 + y$ *the optimum is $z(MILP) = 3$ (with $y = 0$ and all the x at 1). With $M = 5$ the constraint with $y = 0$ remains $3x_1 + 4x_2 + 5x_3 \leq 11$, which excludes $x = (1, 1, 1)$: the optimum drops to 2. The model no longer answers the question asked.*

The strength of the relaxation

On the same instance:

M	$z(LP^+)$
6 (the smallest proved)	3, coincides with $z(MILP)$
20	$37/10 = 3.7$
1000	$1997/500 \approx 3.994$, almost 4

- The degradation is monotone and fast
- Operational rule: **compute M from the data, always, and write it in the text right after the model**

In gurobipy

```
# computed from the data, not guessed
M = sum(max(a[j], 0) for j in range(n)) - b
m.addConstr(gp.quicksum(a[j] * x[j]
                        for j in range(n))
            <= b + M * (1 - y), name="cond")
```

The same idea comes back in precedences.

Summary

- Conditional constraint: $a'x \leq b + M(1 - y)$
- Valid $M \geq \max a'x - b$; too small changes the problem, too large weakens the relaxation
- M is computed from the data and declared right after the model

Precedences and sequencing

Introduction: precedences

- A machine runs one job at a time
- For every pair of jobs one precedes the other
- It is a **disjunction per pair**, not a precedence fixed by the data
- It is written with the big-M of the previous section

The constraints

$\kappa_j \geq 0$ completion time of job j, t_j its duration, $s_{ij} \in \{0, 1\}$ equal to 1 if j precedes i:

$$
\begin{aligned}
s_{ij} + s_{ji} &= 1, && \forall i < j && \left(\binom{n}{2} \text{ constraints}\right) \\
\kappa_i &\geq \kappa_j + t_i - M(1 - s_{ij}), && \forall i \neq j && (n(n-1) \text{ constraints}) \\
\kappa_j &\geq t_j, && \forall j && (n \text{ constraints})
\end{aligned}
$$

with $i, j \in \{1, 2, \ldots, n\}$.

The proof

- $s_{ij} + s_{ji} = 1$: exactly one of the two orders is chosen
- If $s_{ij} = 1$: $\kappa_i \geq \kappa_j + t_i$, job i finishes at least t_i after j finishes, so they do not overlap
- If $s_{ij} = 0$: $\kappa_i \geq \kappa_j + t_i - M$, which must always hold

The smallest M

Since $\kappa_i \geq t_i$ and $\kappa_j \leq \sum_h t_h$ in every sensible solution, it suffices to take

$$M = \sum_{h=1}^{n} t_h$$

because then

$$\kappa_j + t_i - M \leq \sum_h t_h + t_i - \sum_h t_h = t_i \leq \kappa_i$$

- The time horizon $\sum_h t_h$ must be *declared*
- Without an upper bound on the κ_j, no finite M is valid

The strength of the relaxation

- Three jobs of duration $(3, 2, 4)$ on one machine, makespan objective
- $z(MILP) = 9 = \sum_h t_h$ (obviously: a single machine), with completion times $(3, 5, 9)$
- The relaxation is 4: with $s_{ij} = 1/2$ all the precedence constraints are half switched off and the jobs may overlap
- It is the weakest relaxation in the whole chapter: it explains why big-M sequencing models scale badly

In gurobipy

```
M = sum(t)                    # the horizon, declared
for i in range(n):
    for j in range(i):
        m.addConstr(s[i, j] + s[j, i] == 1,
                    name=f"disj{i}{j}")
        m.addConstr(kappa[i] >= kappa[j] + t[i]
                    - M * (1 - s[i, j]),
                    name=f"prec{i}{j}")
        m.addConstr(kappa[j] >= kappa[i] + t[j]
                    - M * (1 - s[j, i]),
                    name=f"prec{j}{i}")
```

Summary

- Sequencing: $s_{ij} + s_{ji} = 1$ and $\kappa_i \geq \kappa_j + t_i - M(1 - s_{ij})$
- $M = \sum_h t_h$, the horizon, which must be declared
- Very weak relaxation: with $s_{ij} = 1/2$ the jobs overlap

“If and only if”

Introduction: "if and only if"

- A binary y must equal 1 *exactly when* a condition holds
- For instance: "all the jobs of the class are done"
- One implication is not enough: two are needed

The constraints and the proof

With $x_1, \ldots, x_p$ binary and the condition "all equal 1":

$$y \le x_j, \qquad \forall j \in \{1, 2, \ldots, p\} \qquad (p \text{ constraints})$$

$$y \ge \sum_{j=1}^{p} x_j - (p-1) \qquad (1 \text{ constraint})$$

- The first gives $y = 1 \Rightarrow x_j = 1$ for every j: if even one x_j is zero, $y = 0$
- The second gives the converse: if all $x_j = 1$ the right-hand side equals 1, so $y = 1$; otherwise it is ≤ 0 and says nothing
- Together: $y = 1 \iff \sum_j x_j = p$ in *every* feasible solution

When one direction is enough

- If y appears in the objective of a maximisation with a bonus $v > 0$ and in no other constraint, the second constraint may be omitted
- In every optimum: if all $x_j = 1$ and $y = 0$, raising y to 1 stays feasible for $y \leq x_j$ and increases the objective by $v > 0$
- This is an optimality argument, and it holds only at the optimum

With $v = 0$ the argument fails (1/2)

Observation 8: With $v = 0$ the argument fails, and y stops meaning anything

Three-job instance, revenues $(2, 2, 2)$*, capacity 3. With bonus* $v = 0$ *and only* $y \leq x_j$*, the optimum may have* $y = 0$ *with* $x = (1, 1, 1)$*: y is no longer a faithful indicator.*

With $v = 0$ the argument fails (2/2)

v	constraints	$z(MILP)$	optimum	faithful?
9	only $y \leq x_j$	15	$y = 1$, $x = (1, 1, 1)$	yes
9	both	15	$y = 1$, $x = (1, 1, 1)$	yes
0	only $y \leq x_j$	6	$y = 0$, $x = (1, 1, 1)$	**no**
0	both	6	$y = 1$, $x = (1, 1, 1)$	yes

- With $v = 0$ the optimal value is the same in the two models
- If y only collects a bonus, the missing direction is useless; if y appears in *other* constraints (e.g. on the number of complete classes), it must be written

The strength of the relaxation and gurobipy

- $z(LP^+) = 15 = z(MILP)$ on this instance: with both directions the relaxation is exact
- The second constraint is weak in the relaxation (with $x_j = 1/2$ its right-hand side is negative), but here that does not matter

```
m.addConstrs((y <= x[j] for j in range(p)),
             name="iff_up")
m.addConstr(y >= gp.quicksum(x[j] for j in range(p))
            - (p - 1), name="iff_down")
```

Summary

- "If and only if": $y \leq x_j$ and $y \geq \sum_j x_j - (p-1)$
- Together they impose $y = 1 \iff$ all $x_j = 1$ in every feasible solution
- One direction may be omitted only if it follows from optimality ($v > 0$) and y appears nowhere else

Counting the different types

Introduction: counting types

- "At least two different types must be produced"
- "At most three foods in the diet"
- "No more than four configurations"
- One counts *how many* elements are active, not how much of them is produced

The constraints

With $q_j \geq 0$ the quantity and y_j the indicator:

$$q_j \leq c_j\, y_j, \quad \forall j \in \{1, 2, \ldots, n\} \qquad (n \text{ constraints})$$

$$q_j \geq \ell\, y_j, \quad \forall j \in \{1, 2, \ldots, n\} \qquad (n \text{ constraints})$$

$$\sum_{j=1}^{n} y_j \geq p \qquad (1 \text{ constraint})$$

The proof, and why both thresholds are needed

- $q_j \leq c_j y_j$ gives "$y_j = 0 \Rightarrow q_j = 0$"
- $q_j \geq \ell y_j$ gives the missing direction "$y_j = 1 \Rightarrow q_j \geq \ell$"
- Without the threshold, $y_j = 1$ with $q_j = 0$ is feasible: the count is satisfied by switching on empty indicators, and "at least p different types" says nothing any more

Counting types works only if every switched-on type actually produces something, and the threshold ℓ is what guarantees it.

The strength of the relaxation and gurobipy

- Three types, unit revenues $(4, 3, 5)$, resource 12, capacity 10 each, threshold $\ell = 3$, at least two types
- $z(MILP) = 57$, with $q = (3, 0, 9)$; the relaxation is also 57: no gap

```
m.addConstrs((q[j] <= c[j] * y[j] for j in range(n)),
             name="activate")
m.addConstrs((q[j] >= ell * y[j] for j in range(n)),
             name="lot")
m.addConstr(y.sum() >= p, name="at_least_p_types")
```

Summary

- Counting types: $\ell y_j \leq q_j \leq c_j y_j$ and $\sum_j y_j \geq p$
- Without the threshold ℓ the count is empty: indicators are switched on without producing
- On the example instance the relaxation is exact

Alldiff and binary expansion

Introduction: alldiff and binary expansion

Two different but related needs:

- giving n objects n *all distinct* values (**alldiff**)
- representing a bounded integer variable with binary variables (**binary expansion**)

The constraints

Alldiff, with $p_{iv} = 1$ if object i gets value v:

$$\sum_{v=1}^{n} p_{iv} = 1, \quad \forall i \in \{1, 2, \ldots, n\} \qquad (n \text{ constraints})$$

$$\sum_{i=1}^{n} p_{iv} = 1, \quad \forall v \in \{1, 2, \ldots, n\} \qquad (n \text{ constraints})$$

Binary expansion of $v \in \{0, 1, \ldots, 2^p - 1\}$:

$$v = \sum_{k=0}^{p-1} 2^k \, b_k \qquad (1 \text{ constraint})$$

$$b_k \in \{0, 1\}, \qquad \forall k \in \{0, \ldots, p-1\} \qquad (p \text{ binaries})$$

The proof

- Alldiff is a double *set partitioning*:
  - the first group gives "every object one value"
  - the second "every value to one object"
- Together they impose a bijection, that is, all distinct values
- Binary expansion is the base-2 representation, unique for every integer in that range: the correspondence between v and $(b_0, \ldots, b_{p-1})$ is one-to-one

Alldiff has an exact relaxation, expansion does not help

- The matrix of the double partitioning is the one of the assignment problem: it is **totally unimodular**
- Every vertex of the relaxation is integer and $z(LP^+) = z(MILP)$ (on the 3×3 instance both are 7): integrality is free
- Binary expansion adds no strength: $\sum_k 2^k b_k$ with $b_k \in [0, 1]$ covers all of $[0, 2^p - 1]$, exactly like $0 \leq v \leq 2^p - 1$
- It serves to *reformulate*, not to tighten: for instance when another part of the model needs binary indicators

Example and gurobipy

- $v \in \{0, \ldots, 7\}$ with $v \geq 5$, $\min v$: the model gives $v = 5 = 1 + 4$, that is $(b_0, b_1, b_2) = (1, 0, 1)$

```
m.addConstrs((p.sum(i, "*") == 1 for i in range(n)),
             name="one_value")
m.addConstrs((p.sum("*", v) == 1 for v in range(n)),
             name="alldiff")
m.addConstr(v == gp.quicksum(2 ** k * b[k]
            for k in range(p)), name="expansion")
```

Summary

- Alldiff: double partitioning, $\sum_v p_{iv} = 1$ and $\sum_i p_{iv} = 1$
- Totally unimodular matrix: exact relaxation
- Binary expansion $v = \sum_k 2^k b_k$: it reformulates, it does not tighten

Soft constraints, deviations and penalties

Introduction: soft constraints

- A constraint one would *prefer* to satisfy, but which may be violated at a price
- Examples: the demand of a period, a timetable preference, a service target
- Turning a hard constraint into a soft one is often the difference between an infeasible model and a useful one

The constraint

In place of $a'x = \beta$ one writes

$$a'x + s^- - s^+ = \beta \qquad \text{(1 constraint)}$$

$$s^- \geq 0 \qquad \text{(1 continuous variable)}$$

$$s^+ \geq 0 \qquad \text{(1 continuous variable)}$$

- The objective gets $\pi^- s^- + \pi^+ s^+$, with penalties $\pi^-, \pi^+ > 0$
- s^- measures how far *below* the target one is, s^+ how far *above*

The proof

- The constraint alone is always satisfiable: any $a'x$ can be compensated with one of the two deviations
- **In every optimum at least one of the two deviations is zero**:
  - if $s^- = \sigma > 0$ and $s^+ = \tau > 0$, subtracting $\min(\sigma, \tau)$ from both keeps the constraint satisfied
  - and reduces the objective by $(\pi^- + \pi^+)\min(\sigma, \tau) > 0$
- So $s^- + s^+ = |a'x - \beta|$ in every optimum: the absolute value written with an equality instead of two inequalities

With a zero penalty the deviation costs nothing

Observation 9: With a zero penalty the deviation costs nothing

The exchange argument only needs $\pi^- + \pi^+ > 0$. With $\pi^- = 0$ and $\pi^+ > 0$ every optimum still has at least one deviation equal to zero, but falling short of β costs nothing: the soft constraint no longer pushes $a'x$ up towards the target. Only if $\pi^- + \pi^+ \leq 0$ may an optimal solution have both deviations positive (with a negative sum the model is even unbounded), and the reading "s^- is the shortfall" is lost. The strength of the conclusion depends on the sign of the coefficients in the objective.

Example: uncovered demand

- Demand 6 in each of three periods, total availability $15 < 18$
- Penalties $\pi^+ = 3$ and $\pi^- = 2$
- The hard model would be infeasible
- The soft one gives $z(MILP) = 6$, with $q = (3, 6, 6)$ and a shortfall of 3 concentrated in the first period
- Nothing in the data says it should be concentrated: any split of the total shortfall 3 has the same cost, and the solver returns one of them

In gurobipy

```
sm = m.addVars(T, name="s_minus")
sp = m.addVars(T, name="s_plus")
m.addConstrs((q[t] + sm[t] - sp[t] == demand[t]
              for t in range(T)), name="target")
m.setObjective(cost + gp.quicksum(
    pen_down * sm[t] + pen_up * sp[t]
    for t in range(T)), GRB.MINIMIZE)
```

Summary

- Soft constraint: $a'x + s^- - s^+ = \beta$ with penalties $\pi^- s^- + \pi^+ s^+$
- With strictly positive penalties, at the optimum at least one deviation is zero and $s^- + s^+ = |a'x - \beta|$
- With a zero penalty the deviation costs nothing

Piecewise linear functions

Introduction: piecewise linear functions

- The cost is not proportional to the quantity: it changes in brackets
- Discounts above a threshold, surcharges beyond a capacity, banded tariffs
- The function $g(q)$ is continuous and piecewise linear, with breakpoints $q_0 < q_1 < \cdots < q_K$ and values $g_0, \ldots, g_K$

The constraints

$\lambda_k \geq 0$ weights of the convex combination, $w_t \in \{0, 1\}$ indicator of piece t (between q_{t-1} and q_t):

$$\sum_{k=0}^{K} \lambda_k = 1, \qquad q = \sum_{k=0}^{K} q_k \lambda_k, \qquad g(q) = \sum_{k=0}^{K} g_k \lambda_k$$

$$\sum_{t=1}^{K} w_t = 1$$

$$\lambda_k \leq \sum_{t\,:\,k \in \{t-1, t\}} w_t, \qquad \forall k \in \{0, \ldots, K\} \quad (K+1 \text{ constraints})$$

The proof

- The last constraint is **adjacency**: λ_k can be positive only if the chosen piece has q_k as an endpoint
- With $\sum_t w_t = 1$ the piece is unique: at most two consecutive λ are positive
- Then $(q, g(q))$ is a point of the segment between (q_{t-1}, g_{t-1}) and (q_t, g_t): a point *of the graph*
- Without adjacency non-consecutive nodes can be mixed and (q, g) lands in the **lower convex envelope**, below the function

If the function is convex adjacency is superfluous

Observation 10: If the function is convex adjacency is superfluous, otherwise it is not

When g is convex and one minimises, the lower convex envelope coincides with the function and the optimal λ turn out adjacent automatically: the model without binaries is already correct. When g is not convex (quantity discounts, marginal cost that falls) the difference shows immediately.

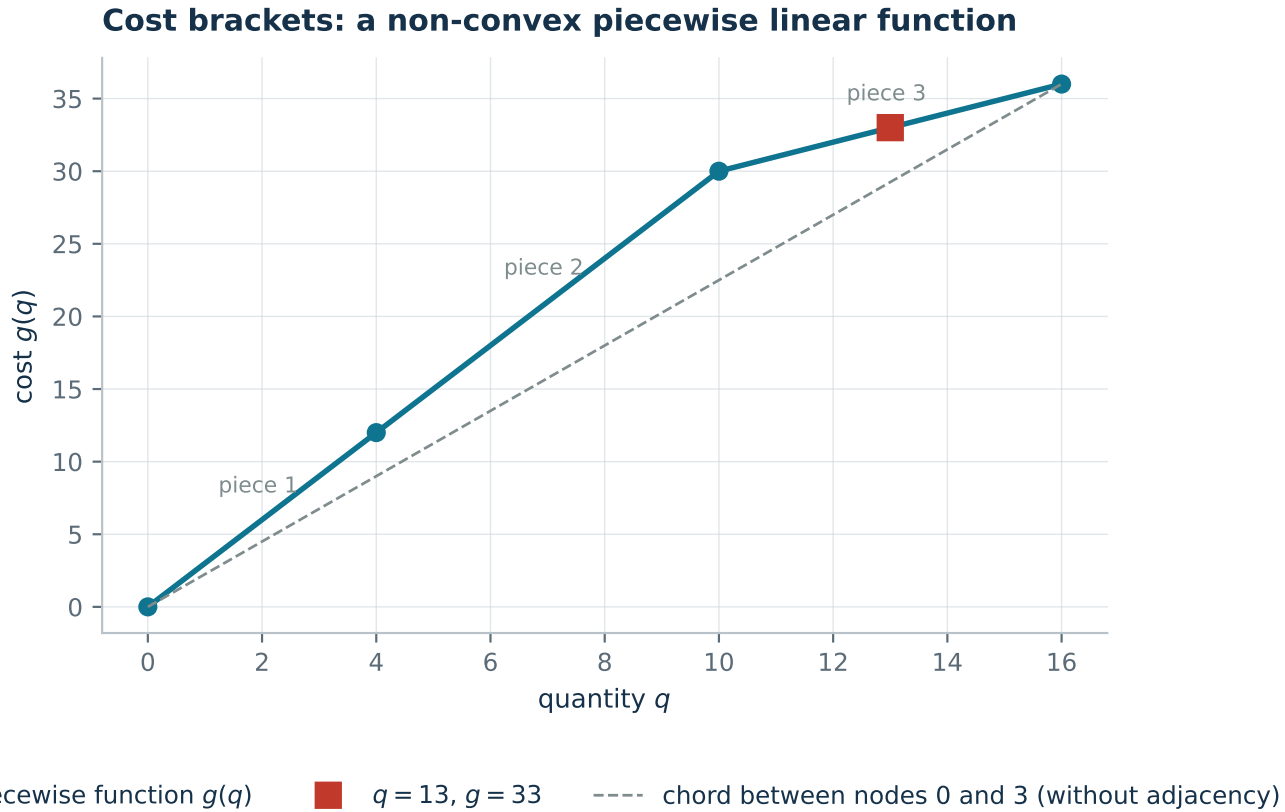
Example: quantity discount

Breakpoints $(0, 4, 10, 16)$, values $(0, 12, 30, 36)$, demand $q \geq 13$:

Formulation	optimal value
without adjacency (free combination)	$117/4 = 29.25$
with adjacency	33

- The first mixes nodes 0 and 3: a cost the function *attains nowhere*, the value of the envelope
- The second mixes nodes 2 and 3, adjacent: $g(13) = 30 + 6 \cdot \frac{3}{6} = 33$

The piecewise function with a quantity discount



The dashed chord between the extreme nodes is what the formulation without adjacency can reach: it lies *below* the graph.

The strength of the relaxation

- Two things must not be confused
- Adjacency changes the *integer set*: the two models have different optima, 29.25 and 33
- But not the strength of the relaxation: with w_t fractional the adjacency constraint does not bite and the two models share the same $z(LP^+) = 117/4$

In gurobipy

```
m.addConstr(lam.sum() == 1, name="convex")
m.addConstr(q == gp.quicksum(nodes[k] * lam[k]
            for k in range(K + 1)), name="abscissa")
m.addConstr(w.sum() == 1, name="one_piece")
# adjacency: lambda_k only on the pieces touching k
for k in range(K + 1):
    m.addConstr(lam[k] <= gp.quicksum(w[t]
                for t in (k - 1, k) if 0 <= t < K),
                name=f"adjacency{k}")
```

Gurobi and the manual formulation

- Gurobi also offers `addGenConstrPWL` and SOS2 types, which do the same job internally
- Here the manual formulation remains the main material, because it is the one whose correctness must be provable

Summary

- Piecewise function: convex combination of the nodes + a single piece + adjacency
- With g convex and minimised adjacency is superfluous; otherwise it is mandatory
- Adjacency changes the integer set, not the strength of the relaxation

The map of techniques

Introduction: the map of techniques

- A table meant for consultation
- Read from the first column ("what I want to say") to the second ("how it is written")
- The third recalls what must be declared every time the technique is used

The map of techniques (1/4)

Technique	Formulation	To be declared
activation	$x_{ij} \leq y_j$ or $\sum_i x_{ij} \leq k_j y_j$	which form, and why; sign of f_j
fixed cost	$q_j \leq c_j y_j$	that c_j is the capacity, not a big-M
minimum lot	$\ell y_j \leq q_j \leq c_j y_j$	that $\ell \leq c_j$; that the relaxation is unaffected

The map of techniques (2/4)

Technique	Formulation	To be declared
integer count	$\sum_i a_i x_i \leq Kw$, w integer	that integrality realises the ceiling
maximum auxiliary	$z \geq t_j x_j$, $z \geq 0$	that z appears nowhere else; its sign in the objective
min-max / max-min	$T \geq L_k$ and min T; $U \leq L_k$ and max U	which of the three objectives, and that they are not comparable
absolute value	$d \geq \pm(u - v)$ in the objective; disjunction if $\geq k$	objective or constraint, and in which direction

The map of techniques (3/4)

Technique	Formulation	To be declared
big-M	$a'x \leq b + M(1-y)$	the value of M computed from the data
precedences	$s_{ij} + s_{ji} = 1$, $\kappa_i \geq \kappa_j + t_i - M(1-s_{ij})$	the horizon and $M = \sum_h t_h$
if and only if	$y \leq x_j$ and $y \geq \sum_j x_j - (p-1)$	whether the second direction is needed or follows from optimality

The map of techniques (4/4)

Technique	Formulation	To be declared
counting types	$\ell y_j \leq q_j \leq c_j y_j$, $\sum_j y_j \geq p$	that without the threshold ℓ the count is empty
alldiff / expansion	double partitioning; $v = \sum_k 2^k b_k$	that alldiff has an exact relaxation
soft constraints	$a'x + s^- - s^+ = \beta$ with penalties	the two signs of the penalties
piecewise function	convex combination + adjacency	whether g is convex; otherwise adjacency is mandatory

The three questions to ask about a new link

1. **Which direction is imposed by the constraints and which is not?** Look at the constraints alone, one case per value of the binary
2. **Does the missing direction follow from optimality?** Exchange argument: it depends on the *sign* of the coefficient in the objective; strictly nonzero gives "in every optimum", zero gives at most "there exists an optimum"
3. **How strong is the relaxation?** Compare $z(LP^+)$ with $z(MILP)$ on a small instance and, with two formulations, compare them with each other, after proving them equivalent on integer points

Summary

- Fourteen techniques, each with its formulation and what must be declared
- For every new link: imposed direction, optimality direction, strength of the relaxation

Final summary (1/2)

Correctness:

- Always distinguish what is imposed by the constraints from what holds only at the optimum
- The exchange argument gives "in every optimum" only with a strictly nonzero coefficient

Coefficients:

- Capacities and big-M are derived from the data and declared
- An invalid M changes the problem; one that is too large weakens the relaxation

Final summary (2/2)

Relaxation:

- The disaggregated form is tighter than the aggregated one
- Minimum lot, maximum, absolute value and big-M precedences give weak relaxations
- Alldiff: totally unimodular matrix, exact relaxation

Disjunctions:

- Non-convex sets ($|u - v| \geq k$, precedences, non-convex piecewise functions) require binaries