MIP heuristics for the classical problems

Outline

- Why MIP heuristics for the classical problems?
- MIP-based heuristics
- The problems and their MIP models
- Rounding
- Diving
- The feasibility pump
- RINS, local branching and proximity search
  - RINS
  - Local branching
  - Proximity search
- Relax-and-fix and fix-and-optimize on lot sizing
- Heuristics inside the solver
- What the MIP-based heuristics give
- Typical mistakes
- `Python` implementation
- In summary

Learning objectives

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

- write the MIP models of the classical problems in the form required by the MIP-based heuristics, and compute their LP relaxations;
- execute by hand rounding, diving, the feasibility pump, RINS, local branching and proximity search on the 0/1 knapsack problem, on bin packing and on $P \,||\, C_{\max}$;
- execute relax-and-fix and fix-and-optimize on a lot sizing instance;
- explain why each method succeeds or fails on each problem;
- compute the gap of a heuristic solution when a dual bound is available.

Prerequisites (1/2)

- From the module *Integer Linear Programming*, the chapter «MIP heuristics»: the methods of this chapter (rounding, diving, the feasibility pump, RINS, local branching, proximity search, relax-and-fix and fix-and-optimize), their pseudo-codes and the conditions under which they work.

- From the module *Combinatorial Optimization Problems*, the formulations of the classical problems: the models of the 0/1 knapsack problem, of the bin packing problem, of the travelling salesman problem, of $P \,||\, C_{\max}$ and of lot sizing.

Prerequisites (2/2)

- From this module, the chapters on heuristic algorithms, from «Heuristic algorithms: definitions» to «Metaheuristics» («Constructive heuristics», «Local search»): the constructive heuristics (*GR*, *FF*, *NN*, *LPT*), local search and metaheuristics, whose solutions are the starting points of this chapter.

Why MIP heuristics for the classical problems?

Introduction: Why MIP heuristics for the classical problems?

- We have already met two very different ways of looking for good solutions
- In this chapter we bring them together: the methods based on a MIP model are applied to the classical problems of the module *Combinatorial Optimization Problems*
- We compare what the two approaches can say about the quality of their solutions

From the methods to the classical problems (1/2)

- The heuristic algorithms of the previous chapters — constructive heuristics, local search and metaheuristics — work directly on a representation of the solutions:
  - a binary vector,
  - a vector *bin*,
  - a tour,
  - a vector *mach*.
- They never look at a mathematical model of the problem, and they never use a solver.

From the methods to the classical problems (2/2)

- The MIP-based heuristics of the chapter «MIP heuristics» of the module *Integer Linear Programming* start instead from the model of the problem.
- They use its LP relaxation and a MIP solver.
- In this chapter they are applied to the classical problems, with the reference instances of the previous chapters.

Summary

Two families of heuristics:

- constructive heuristics, local search, metaheuristics: they work on a representation of the solutions, without a model and without a solver
- MIP-based heuristics: they work on the model, with its LP relaxation and a MIP solver

This chapter:

- the MIP-based heuristics on the classical problems and their reference instances

MIP-based heuristics

Introduction: MIP-based heuristics

- We define what a MIP-based heuristic is
- We recall the notation of the chapter «MIP heuristics» of the module *Integer Linear Programming*, which we use throughout
- We say how the examples of this chapter are computed

MIP-based heuristics

Definition 1: MIP-based heuristic

*A heuristic algorithm for a problem with a MIP model is **MIP-based** (a **matheuristic**) if it uses the LP relaxation of the model and a MIP solver to search for a feasible solution.*

- The MIP-based heuristics are defined in the module *Integer Linear Programming*, chapter «MIP heuristics».
- In this chapter they are applied to the classical problems, with the reference instances of the previous chapters.

Notation

The notation is that of the chapter «MIP heuristics»:

- a MIP with integer variables $\mathscr{D}$;
- $\tilde{\boldsymbol{x}}$ is an optimal solution of the LP relaxation;
- $\bar{\boldsymbol{x}}$ is a feasible solution (the incumbent);
- $[\,v\,] = \lfloor v + \frac{1}{2} \rfloor$ is the rounding to the nearest integer;
- $\Delta(\boldsymbol{x}, \bar{\boldsymbol{x}}) = \sum_{j \in \mathscr{D}} |x_j - \bar{x}_j|$ is the distance from an integer point.

A maximization problem, such as the KP, is handled by exchanging min and max, $\leq$ and $\geq$ in the comparisons of objective values.

How the examples are computed

- The examples are solved with the MIP solver Gurobi, used from Python through `gurobipy`.
- When an LP has several optimal solutions, the text reports the one returned by the solver.
- The codes are in the last section.

Representation or model

Constructive heuristics, local search and metaheuristics work on a representation of the solutions;

a MIP-based heuristic works on the model of the problem, through its LP relaxation and a MIP solver.

Summary

MIP-based heuristics:

- a matheuristic uses the LP relaxation of the model and a MIP solver
- they are defined in the module *Integer Linear Programming*

Notation:

- $\tilde{\boldsymbol{x}}$ LP solution, $\bar{\boldsymbol{x}}$ incumbent, $[\, v \,]$ nearest integer, $\Delta(\boldsymbol{x}, \bar{\boldsymbol{x}})$ distance
- the examples are solved with Gurobi from Python

The problems and their MIP models

Introduction: The problems and their MIP models

- We recall the four classical problems, their reference instances and the values found by the heuristics of the previous chapters
- We recall the MIP models on which the methods of this chapter work
- We look at the LP relaxations on the reference instances: they are the starting point of every method

The four problems

- The heuristics are illustrated on the four problems of the previous chapters:
  - the 0/1 knapsack problem (KP),
  - the bin packing problem (BPP),
  - the travelling salesman problem (TSP),
  - $P \,||\, C_{\max}$.

  Their definitions and assumptions are on their pages.

- The reference instances are recalled in the following table, with the values computed in the previous chapters on heuristic algorithms.

The reference instances

problem	reference instance	values
KP	$p = (10, 7, 6, 4)$, $w = (5, 4, 3, 3)$, $c = 9$	GR: $\{1, 3\}$, profit 16; $z(I) = 17$, $\{1, 2\}$
BPP	$w = (4, 4, 5, 3, 2, 3)$, $c = 7$	FF: 4 bins; $z(I) = 3$
TSP	5 cities, the reference distances	NN from city 1: 25; $z(I) = 18$
$P \mid\mid C_{\max}$	$t = (5, 5, 4, 4, 3, 3, 3)$, $m = 3$	LPT: $J_1 = \{1, 5, 7\}$, $J_2 = \{2, 6\}$, $J_3 = \{3, 4\}$, 11; $z(I) = 9$

The MIP models

- The MIP models are those of the module *Combinatorial Optimization Problems*:
  - the $\mathrm{ILP}_{\mathrm{KP}}$ model of the chapter «Formulations of knapsack problems»,
  - the $\mathrm{ILP}_{\mathrm{BPP}}$ model of the chapter «Formulations of packing problems»,
  - the $\mathrm{ILP}_{\mathrm{TSP}}^{\mathrm{DFJ}}$ model of the chapter «Formulations of routing problems»,
  - the $\mathrm{ILP}_{P\|C_{\max}}$ model of the chapter «Formulations of scheduling problems».

- We recall the three models used in the examples, with the variables and the notation of those chapters.

The 0/1 knapsack problem

- The binary variable x_j is equal to 1 if item j is selected in the knapsack, and to 0 otherwise, for every $j \in \mathscr{I}$.

Using these variables, an ILP model for the 0/1 knapsack problem (KP) reads as follows:

$$\max \quad \sum_{j \in \mathscr{I}} p_j \, x_j$$

$$\text{subject to} \quad \sum_{j \in \mathscr{I}} w_j \, x_j \leq c,$$

$$x_j \in \{0, 1\}, \quad \forall j \in \mathscr{I}.$$

This ILP model is referred to as the $\mathrm{ILP}_{\mathrm{KP}}$ model.

The bin packing problem: variables

- Given an upper bound m on the number of bins used in an optimal solution, and the set $\mathscr{B} = \{1, 2, \ldots, m\}$ of the available bins:
  - the binary variable x_{jb} is equal to 1 if item j is packed in bin b,
  - the binary variable y_b is equal to 1 if bin b is used,

  for every $j \in \mathscr{I}$ and $b \in \mathscr{B}$.

- The content of bin b is the packing pattern $P_b = \{j \in \mathscr{I} : x_{jb} = 1\}$.

- The number n of items is always a valid upper bound m; on the reference instance we use $m = n = 6$.

The bin packing problem: model

Using these variables, an ILP model for the bin packing problem (BPP) reads as follows:

$$\min \quad \sum_{b \in \mathscr{B}} y_b$$

$$\text{subject to} \quad \sum_{b \in \mathscr{B}} x_{jb} = 1, \qquad \forall j \in \mathscr{I},$$

$$-\sum_{j \in \mathscr{I}} w_j \, x_{jb} + c \, y_b \geq 0, \qquad \forall b \in \mathscr{B},$$

$$x_{jb} \in \{0,1\}, \quad \forall j \in \mathscr{I}, \ \forall b \in \mathscr{B},$$

$$y_b \in \{0,1\}, \qquad \forall b \in \mathscr{B}.$$

This ILP model is referred to as the $\mathrm{ILP}_{\mathrm{BPP}}$ model.

The travelling salesman problem

- The $\mathrm{ILP}_{\mathrm{TSP}}^{\mathrm{DFJ}}$ model has:
  - a binary variable x_{ij} for every arc $(i, j) \in A$,
  - the constraints on the arcs leaving and entering every city,
  - the subtour elimination constraints on the subsets $S \subseteq \mathcal{V} \setminus \{1\}$ with $|S| \geq 2$.

- On the reference instance, with $n = 5$ cities, it has 20 variables and $5 + 5 + 11 = 21$ constraints.

The scheduling problem $P \,||\, C_{\max}$: variables

- Given the set $\mathscr{H} = \{1, 2, \ldots, m\}$ of the machines:
  - the binary variable x_{jh} is equal to 1 if job j is assigned to machine h, for every $j \in \mathscr{J}$ and $h \in \mathscr{H}$,
  - the non-negative continuous variable $C_{\max}$ is the makespan.
- $C_{\max}$ is the only continuous variable of the four models: given the assignment, the completion sets it to the largest load $L_h = \sum_{j \in \mathscr{J}} t_j \, x_{jh}$.

The scheduling problem $P \,||\, C_{\max}$: model

Using these variables, an ILP model for $P||\,C_{\max}$ reads as follows:

$$\begin{aligned}
&\min \quad C_{\max} \\
&\text{subject to} \quad \sum_{h \in \mathscr{H}} x_{jh} = 1, && \forall j \in \mathscr{J}, \\
&-\sum_{j \in \mathscr{J}} t_j \, x_{jh} + C_{\max} \geq 0, && \forall h \in \mathscr{H}, \\
&x_{jh} \in \{0,1\}, \quad \forall j \in \mathscr{J}, \; \forall h \in \mathscr{H}, \\
&C_{\max} \geq 0.
\end{aligned}$$

This ILP model is referred to as the $\mathrm{ILP}_{P||\,C_{\max}}$ model.

With the models in hand,
we look at their LP relaxations
on the reference instances.

The LP relaxations on the reference instances

- The following table reports the size of the four models on the reference instances, the value $z(\mathrm{LP})$ of their LP relaxations, computed with Gurobi, and the optimal value $z(I)$.

problem	variables	constraints	$z(\mathrm{LP})$	$z(I)$	LP solution integer?
KP	4	1	17.75	17	no
BPP	42	12	3	3	no
TSP	20	21	18	18	yes
$P \mid\mid C_{\max}$	22	10	9	9	no

Strong relaxations

- On all four instances the LP value, rounded in the direction of the bound (down for the KP, up for the three minimization problems), is already the optimal value.
- The relaxations are *strong* on these instances: the difficulty is to find a solution that attains the bound, not to prove it.
- This is the typical situation in which MIP-based heuristics are useful.

The TSP reference instance

- On the TSP reference instance the LP solution of the $\mathrm{ILP}_{\mathrm{TSP}}^{\mathrm{DFJ}}$ model is itself the optimal tour $(1, 3, 5, 2, 4)$, with

$$\tilde{x}_{13} = \tilde{x}_{35} = \tilde{x}_{52} = \tilde{x}_{24} = \tilde{x}_{41} = 1 :$$

  every method of this chapter finds it at once.

- The LP relaxation of the $\mathrm{ILP}_{\mathrm{TSP}}^{\mathrm{MTZ}}$ model is weaker: its value is 17.

- The TSP is therefore not used in the other examples of this chapter.

Example: the LP solutions of the reference instances (1/3)

KP.

- The LP solution is
$$\tilde{\boldsymbol{x}} = \left(1, \tfrac{1}{4}, 1, 0\right).$$
- The items are taken by non-increasing efficiency, and the split item 2 is taken for the fraction $\frac{9-8}{4} = \frac{1}{4}$ of its weight, so
$$z(\mathrm{LP}) = 10 + \frac{7}{4} + 6 = 17.75.$$

Example: the LP solutions of the reference instances (2/3)

$P \,||\, C_{\max}$. The LP solution returned by the solver is (zeros omitted)

	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$
$\tilde{x}_{j1}$		$\frac{3}{5}$				1	1
$\tilde{x}_{j2}$			$\frac{1}{2}$	1	1		
$\tilde{x}_{j3}$	1	$\frac{2}{5}$	$\frac{1}{2}$				

with loads

$$L_1 = 3+5\cdot\tfrac{3}{5}+3 = 9, \quad L_2 = 4\cdot\tfrac{1}{2}+4+3 = 9, \quad L_3 = 5+5\cdot\tfrac{2}{5}+4\cdot\tfrac{1}{2} = 9$$

and $\tilde{C}_{\max} = 9 = \frac{27}{3}$. The LP has infinitely many optimal solutions: every fractional assignment with three loads equal to 9 is optimal.

Example: the LP solutions of the reference instances (3/3)

BPP. The LP solution returned by the solver:

- puts item 1 into bin 6, item 2 into bin 4,
- item 3 for $\frac{4}{5}$ into bin 3 and for $\frac{1}{5}$ into bin 4,
- item 4 into bin 6, item 5 into bin 1, item 6 into bin 3,
- and opens the bins by the fractions

$$\tilde{\boldsymbol{y}} = \left(\frac{2}{7}, 0, 1, \frac{5}{7}, 0, 1\right),$$

just enough to contain their loads: 2, 0, 7, 5, 0, 7.

Its value is $\frac{21}{7} = 3$.

Summary

The models:

- $\mathrm{ILP}_{\mathrm{KP}}$, $\mathrm{ILP}_{\mathrm{BPP}}$, $\mathrm{ILP}_{\mathrm{TSP}}^{\mathrm{DFJ}}$ and $\mathrm{ILP}_{P\|C_{\max}}$, from the part «Formulations and their properties»
- $C_{\max}$ is the only continuous variable

The LP relaxations on the reference instances:

- strong: the rounded LP value is already the optimal value
- the difficulty is to find a solution that attains the bound
- on the TSP the LP solution is already the optimal tour, so the TSP is not used later

Rounding

Introduction: Rounding

- Rounding is the simplest way to turn the LP solution into an integer point
- We apply it to the KP, to $P \,||\, C_{\max}$ and to the BPP
- We see when it works and why it fails on problems with assignment constraints

Rounding

- *Rounding* rounds the LP solution to the nearest integer point and completes the continuous variables; it fails if the rounded point violates a constraint.
- Rounding down is always feasible when all the coefficients of A are non-negative and $b \geq 0$ (the observation on rounding down of the chapter «MIP heuristics» of the module *Integer Linear Programming*).
- An equality constraint, such as the assignment constraints of the BPP and of $P \,||\, C_{\max}$, is almost always violated by independent roundings.

Example: rounding on the reference instances (1/3)

KP.

- The LP solution $\tilde{\boldsymbol{x}} = \left(1, \frac{1}{4}, 1, 0\right)$ is rounded to

$$\bar{\boldsymbol{x}} = (1, 0, 1, 0): \quad \text{weight } 8 \leq 9, \quad \text{profit } 16.$$

- It is the solution of *GR* of the chapter «Constructive heuristics».

- Rounding $\tilde{x}_2 = \frac{1}{4}$ up instead would give $(1, 1, 1, 0)$, of weight $12 > 9$: infeasible.

Example: rounding on the reference instances (2/3)

$P \,||\, C_{\max}$.

- In the LP solution of the reference instance, $\left[\frac{3}{5} \right] = 1$ and $\left[\frac{2}{5} \right] = 0$ send job 2 to machine 1.
- But $\left[\frac{1}{2} \right] = 1$ assigns job 3 to *both* machines 2 and 3:

$$\bar{x}_{31} + \bar{x}_{32} + \bar{x}_{33} = 0 + 1 + 1 = 2 \neq 1.$$

- The assignment constraint of job 3 is violated: **rounding fails**.

Example: rounding on the reference instances (3/3)

BPP.

- Rounding the LP solution of the reference instance gives $\bar{\boldsymbol{y}} = (0, 0, 1, 1, 0, 1)$ and puts item 3 into bin 3.
- Two capacity constraints are violated:
  - bin 1 contains item 5 but $\bar{y}_1 = 0$ ($-2 + 7 \cdot 0 \geq 0$ is false),
  - bin 3 contains items 3 and 6 ($-(5 + 3) + 7 \cdot 1 \geq 0$ is false).
- **Rounding fails.**

Three outcomes

- The three outcomes summarize the method.
- On the KP, with a single packing constraint, rounding is safe (the observation on rounding down of the chapter «MIP heuristics» of the module *Integer Linear Programming*).
- On the problems with assignment constraints it fails, because the LP spreads a job or an item over several machines or bins, and rounding cannot decide where it goes.

Summary

Rounding:

- rounds $\tilde{\boldsymbol{x}}$ to the nearest integer point and completes the continuous variables
- safe with a single packing constraint: on the KP it gives the solution of GR, of profit 16

Assignment constraints:

- independent roundings almost never sum to 1
- on $P \,||\, C_{\max}$ and on the BPP rounding fails

Diving

Introduction: Diving

- Diving does not round all the variables at once: it fixes them one at a time and re-solves the LP
- We run fractional diving on the KP and on $P \,||\, C_{\max}$, and guided diving on $P \,||\, C_{\max}$
- We see why a dive can take a wrong decision, and why it fails on the BPP

Diving

- *Diving* fixes one fractional variable at a time and re-solves the LP:
  - fractional diving fixes the least fractional variable to its nearest integer,
  - guided diving fixes the variable closest to the incumbent to the incumbent value;

  it fails when the LP becomes infeasible.

- Ties are broken in favour of the variable that comes first in the order of the variables ($x_{11}, x_{12}, \ldots$ for the assignment models, then $C_{\max}$ or $y_1, y_2, \ldots$).

Example: fractional diving on the KP (1/3)

Step 1. The LP solution is $\tilde{\boldsymbol{x}} = \left(1, \frac{1}{4}, 1, 0\right)$, of value 17.75. The only fractional variable is x_2, rounded to $\left[\frac{1}{4}\right] = 0$: fix $x_2 := 0$.

Step 2. Without item 2, the LP takes items 1 and 3 (weight 8) and the fraction $\frac{1}{3}$ of item 4:

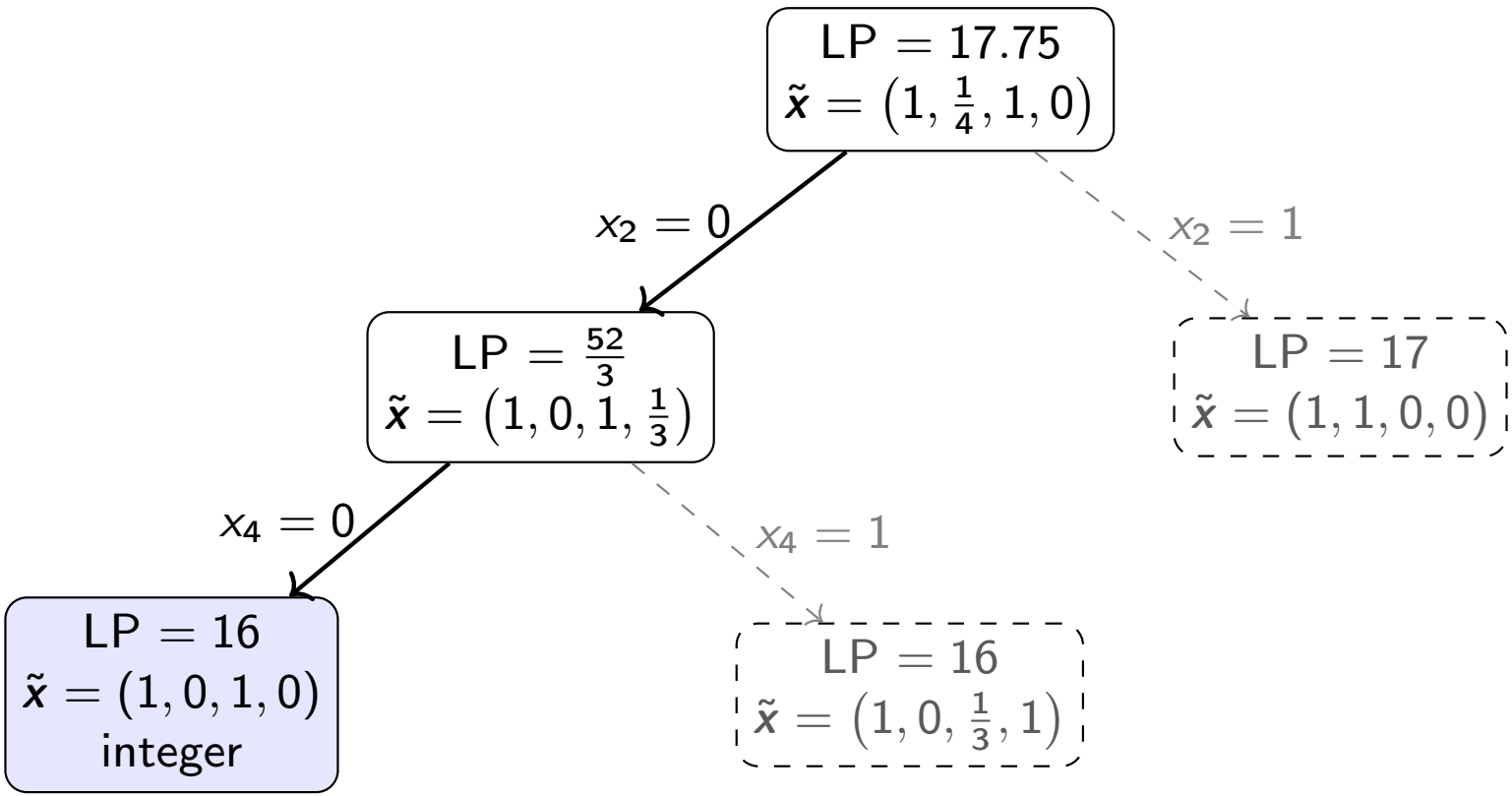
$$\tilde{\boldsymbol{x}} = \left(1, 0, 1, \frac{1}{3}\right), \quad \text{of value } 10 + 6 + \frac{4}{3} = \frac{52}{3} \approx 17.33.$$

Fix $x_4 := \left[\frac{1}{3}\right] = 0$.

Step 3. The LP solution $\tilde{\boldsymbol{x}} = (1, 0, 1, 0)$, of value 16, is integer: the dive returns $\{1, 3\}$, of profit 16.

Example: fractional diving on the KP (2/3)

The dive is the path drawn with solid lines in the branch-and-bound tree below; the dashed nodes are the branches it never visits, with the value of their LP relaxations.



Example: fractional diving on the KP (3/3)

- The branch $x_2 = 1$, abandoned at the first step, contains the optimal solution $\{1, 2\}$ of profit 17.
- $\{1, 2\}$ is an optimal solution of the LP relaxation of that branch.
- The first decision of the dive, $\left[\frac{1}{4} \right] = 0$, was the wrong one.

On the KP the dive needs three LPs.

On $P \,||\, C_{\max}$ the LP has many optimal solutions:
we follow the dive step by step.

Example: fractional diving on $P \,||\, C_{\max}$ (1/3)

- The dive starts from the LP solution of the reference instance.
- The table reports, for each step, the value of the LP, its fractional variables and the fixing.
- A fractionality $|\tilde{x}_j - [\,\tilde{x}_j\,]\,|$ equal for two variables is broken by the order $x_{11}, x_{12}, x_{13}, x_{21}, \ldots$

Example: fractional diving on $P \,||\, C_{\max}$ (2/3)

step	LP value	fractional variables	fixing
1	9	$x_{21} = \frac{3}{5}$, $x_{23} = \frac{2}{5}$, $x_{32} = \frac{1}{2}$, $x_{33} = \frac{1}{2}$	$x_{21} := 1$
2	9	$x_{11} = \frac{1}{5}$, $x_{13} = \frac{4}{5}$, $x_{32} = \frac{3}{4}$, $x_{33} = \frac{1}{4}$	$x_{11} := 0$
3	9	$x_{32} = \frac{3}{4}$, $x_{33} = \frac{1}{4}$, $x_{41} = \frac{1}{4}$, $x_{43} = \frac{3}{4}$	$x_{32} := 1$
4	9	$x_{41} = \frac{1}{4}$, $x_{43} = \frac{3}{4}$, $x_{52} = \frac{2}{3}$, $x_{53} = \frac{1}{3}$	$x_{41} := 0$
5	9	$x_{61} = \frac{1}{3}$, $x_{62} = \frac{2}{3}$	$x_{61} := 0$
6	9	$x_{12} = \frac{3}{5}$, $x_{13} = \frac{2}{5}$, $x_{71} = \frac{1}{3}$, $x_{72} = \frac{2}{3}$	$x_{71} := 0$

Example: fractional diving on $P \,||\, C_{\max}$ (3/3)

step	LP value	fractional variables	fixing
7	9.5	$x_{42} = \frac{1}{8}$, $x_{43} = \frac{7}{8}$	$x_{42} := 0$
8	9.5	$x_{62} = \frac{1}{6}$, $x_{63} = \frac{5}{6}$	$x_{62} := 0$
9	9.5	$x_{12} = \frac{1}{2}$, $x_{13} = \frac{1}{2}$	$x_{12} := 1$
10	9.5	$x_{72} = \frac{1}{6}$, $x_{73} = \frac{5}{6}$	$x_{72} := 0$
11	10	none	—

- The last LP solution is integer: $J_1 = \{2, 5\}$, $J_2 = \{1, 3\}$, $J_3 = \{4, 6, 7\}$, with loads 8, 9, 10 and makespan 10.
- The dive improves on rounding, which failed, and on *LPT* (11), but it does not reach the optimum 9.

A flat LP value

- For six steps the LP value stays at 9: the LP of $P \,||\, C_{\max}$ has a huge set of optimal solutions, and fixing one variable just moves the LP to another of them.
- The LP value tells nothing about the quality of the decisions until the load of a machine is forced above 9, at step 7, and then it is too late to undo the decisions that caused it.
- Min-max objectives are as flat for the LP as they are for local search.

Example: guided diving on $P \,||\, C_{\max}$ (1/3)

- The incumbent is the solution of LPT, $J_1 = \{1, 5, 7\}$, $J_2 = \{2, 6\}$, $J_3 = \{3, 4\}$, of makespan 11, i.e.,

$$\bar{x}_{11} = \bar{x}_{51} = \bar{x}_{71} = \bar{x}_{22} = \bar{x}_{62} = \bar{x}_{33} = \bar{x}_{43} = 1$$

  and all the other $\bar{x}_{jh} = 0$.

- Guided diving fixes, at each step, the fractional variable closest to the incumbent to its incumbent value.

Example: guided diving on $P \,||\, C_{\max}$ (2/3)

Step 1. In the LP solution of the reference instance the fractional variables and their distances from the incumbent are

$$|\tilde{x}_{21} - \bar{x}_{21}| = \tfrac{3}{5}, \quad |\tilde{x}_{23} - \bar{x}_{23}| = \tfrac{2}{5},$$

$$|\tilde{x}_{32} - \bar{x}_{32}| = \tfrac{1}{2}, \quad |\tilde{x}_{33} - \bar{x}_{33}| = \tfrac{1}{2}.$$

The closest is x_{23}: fix $x_{23} := \bar{x}_{23} = 0$.

Step 2. The new LP solution, of value 9, has the fractional variables $x_{12} = \frac{2}{5}$, $x_{13} = \frac{3}{5}$, $x_{31} = \frac{1}{2}$, $x_{32} = \frac{1}{2}$, at distances $\frac{2}{5}$, $\frac{3}{5}$, $\frac{1}{2}$, $\frac{1}{2}$ from the incumbent. Fix $x_{12} := \bar{x}_{12} = 0$.

Example: guided diving on $P \,||\, C_{\max}$ (3/3)

Step 3. The LP solution is integer, of value 9:

$$J_1 = \{5, 6, 7\}, \qquad J_2 = \{2, 4\}, \qquad J_3 = \{1, 3\},$$

all loads equal to 9.

Guided diving finds an **optimal** solution after two LPs.

What the incumbent decides

- Guided by a solution of makespan 11, the dive ends at a solution of makespan 9.
- The incumbent only decides the *values* of the two fixed variables, both equal to 0, while the LP rebuilds the rest of the assignment.

Diving on the BPP

- On the BPP reference instance, fractional diving takes 8 fixings, all with LP value 3.
- The ninth, $x_{12} := 1$, puts item 1 into bin 2, which already contains item 2 by an earlier fixing, and $4 + 4 > 7$ makes the LP infeasible: **the dive fails**.
- The fixings $x_{jb} = 1$ commit items to bins one at a time, and the LP of the BPP is so weak (3 at every step) that it gives no warning before the dead end.

Summary

Fractional diving:

- on the KP it returns $\{1, 3\}$, of profit 16: the first fixing $x_2 := 0$ was wrong
- on $P \,||\, C_{\max}$ it reaches makespan 10: the LP value stays flat at 9 for six steps
- on the BPP it fails after 8 fixings

Guided diving:

- on $P \,||\, C_{\max}$, guided by LPT, it finds the optimum 9 after two LPs

The feasibility pump

Introduction: The feasibility pump

- The feasibility pump looks for a feasible solution when rounding and diving fail
- We run it on $P \,||\, C_{\max}$ and on the BPP
- We see that it finds feasible solutions well, but it does not care about their quality

The feasibility pump

- The *feasibility pump* alternates:
  - the projection of an integer point onto the LP feasible region (one LP minimizing Δ),
  - the rounding of the projection,

  with a flip when the rounding returns the same integer point and a random restart when the pump cycles, until the distance is 0.

- The random numbers are those of the *minimal standard* generator of Park and Miller with the seeds $1, 2, \ldots$, as in the chapter «Metaheuristics».

Example: the feasibility pump on $P \,||\, C_{\max}$ (1/2)

Iteration 0. The LP solution of the reference instance is rounded as in the rounding example: job 3 is assigned to machines 2 and 3, and $\bar{\boldsymbol{x}}^0$ is infeasible.

Iteration 1. The projection minimizes $\Delta(\boldsymbol{x}, \bar{\boldsymbol{x}}^0)$, which is linear since all the integer variables are binary.

- The solver returns the point of $\mathcal{L}$ that keeps all the assignments of $\bar{\boldsymbol{x}}^0$ and removes job 3 from machine 2, i.e., it moves x_{32} from 1 to 0: $\Delta = 1$.
- Removing job 3 from machine 3 would be equally close.

Example: the feasibility pump on $P \,||\, C_{\max}$ (2/2)

Iteration 1 (continued). The result is integer:

$$J_1 = \{2, 6, 7\}, \qquad J_2 = \{4, 5\}, \qquad J_3 = \{1, 3\},$$

with loads 11, 7, 9, and its rounding is the new integer point $\bar{\boldsymbol{x}}^1$.

Iteration 2. $\bar{\boldsymbol{x}}^1 \in \mathcal{L}$, the distance is 0, and the pump returns a solution of makespan 11, after two projections.

Feasibility, not quality

- The solution has the same makespan as LPT.
- The projection takes care of the assignment constraints and ignores the loads, since $C_{\max}$ is continuous and does not enter Δ.
- The pump answers the question "is there a feasible solution?", which for $P \,||\, C_{\max}$ was not a difficult question.

Example: the feasibility pump on the BPP (1/3)

- On the BPP reference instance, rounding and diving fail.
- The pump is run with the seeds $1, 2, \ldots, 10$ of the random numbers.
- The table reports:
  - the number of projections, of flips and of restarts,
  - the number of bins opened by the solution found ($\sum_b \bar{y}_b$, the value of the solution),
  - the number of non-empty bins.

Example: the feasibility pump on the BPP (2/3)

seed	1	2	3	4	5	6	7	8	9	10
projections	15	12	9	9	6	6	19	5	10	20
flips	8	5	3	4	3	3	8	2	5	8
restarts	4	3	3	2	1	1	5	1	2	4
bins opened	4	4	4	5	5	4	3	4	4	5
non-empty bins	4	4	4	3	5	4	3	4	4	4

Example: the feasibility pump on the BPP (3/3)

- The pump finds a feasible solution with every seed, but always after some restarts: without them it would cycle.
- Only with seed 7 the solution is optimal: $\{3, 5\}$, $\{2, 4\}$, $\{1, 6\}$, of value 3.
- With seed 4 the pump opens 5 bins, two of which stay empty: the packing $\{3, 5\}$, $\{1, 6\}$, $\{2, 4\}$ is optimal, but the solution has value 5, because nothing in the distance function asks to close the empty bins.

A first feasible solution

The pump is a method to find
a *first* feasible solution,
and it does that well.

The quality of the solution
is the job of the improvement methods,
which start from an incumbent.

Summary

The feasibility pump:

- projection onto the LP feasible region, rounding, flips and random restarts
- after the first LP it ignores the objective

On the reference instances:

- $P \,||\, C_{\max}$: a solution of makespan 11 after two projections, as LPT
- BPP: a feasible solution with every seed, optimal only with seed 7; empty bins may stay open

RINS, local branching and proximity search

Introduction: RINS, local branching and proximity search

- These are improvement methods: they start from an incumbent and look for a better solution with a sub-MIP
- We run each of them on the KP and on $P \,||\, C_{\max}$
- We see the link between local branching and the neighbourhoods of local search, and how proximity search can prove optimality

Three methods with a sub-MIP

- The three methods start from an incumbent $\bar{\boldsymbol{x}}$ and solve a sub-MIP with the cutoff constraint

$$f(\boldsymbol{x}) \leq f(\bar{\boldsymbol{x}}) - \theta,$$

  which here has $\theta = 1$ since the objective values are integer.

- RINS fixes the integer variables on which the LP solution and the incumbent agree.
- Local branching adds $\Delta(\boldsymbol{x}, \bar{\boldsymbol{x}}) \leq k$ and enlarges k when it finds nothing.
- Proximity search minimizes $\Delta(\boldsymbol{x}, \bar{\boldsymbol{x}})$.

We start with RINS:
which variables does it fix,
and how large is the sub-MIP that is left?

Example: RINS on the KP (1/2)

- The incumbent is the solution $\bar{\boldsymbol{x}} = (1, 0, 1, 0)$ of GR, of profit 16.
- The LP solution is $\tilde{\boldsymbol{x}} = \left(1, \frac{1}{4}, 1, 0\right)$.
- They agree on $x_1 = 1$, $x_3 = 1$, $x_4 = 0$, which are fixed.
- The sub-MIP, with the capacity constraint and the cutoff constraint, is

$$\begin{array}{rrrcr} \max & 16 & + 7\,x_2 & & \\ \text{subject to} & 8 & + 4\,x_2 & \leq & 9 \\ & 16 & + 7\,x_2 & \geq & 17 \\ & & x_2 & \in & \{0, 1\} \end{array}$$

Example: RINS on the KP (2/2)

- The sub-MIP is infeasible: the capacity requires $x_2 = 0$, the cutoff $x_2 = 1$.
- RINS returns the incumbent.
- The optimal solution $\{1, 2\}$ has $x_3 = 0$, but x_3 is fixed to 1 because the LP and the incumbent agree on it.

Example: RINS on $P \,||\, C_{\max}$ (1/3)

- The incumbent is the solution of *LPT* ($J_1 = \{1, 5, 7\}$, $J_2 = \{2, 6\}$, $J_3 = \{3, 4\}$, makespan 11), and $\tilde{\boldsymbol{x}}$ is the LP solution of the reference instance.

- The table compares, for each job, the machine of the incumbent with the LP solution (jobs 1–4):

	$j = 1$	$j = 2$	$j = 3$	$j = 4$
machine in $\bar{\boldsymbol{x}}$	1	2	3	3
machines in $\tilde{\boldsymbol{x}}$	3	1 $(\frac{3}{5})$, 3 $(\frac{2}{5})$	2 $(\frac{1}{2})$, 3 $(\frac{1}{2})$	2
fixed variables	$x_{12} = 0$	—	$x_{31} = 0$	$x_{41} = 0$

Example: RINS on $P \,||\, C_{\max}$ (2/3)

The comparison for jobs 5–7:

	$j = 5$	$j = 6$	$j = 7$
machine in $\bar{\boldsymbol{x}}$	1	2	1
machines in $\tilde{\boldsymbol{x}}$	2	1	1
fixed variables	$x_{53} = 0$	$x_{63} = 0$	$x_{71} = 1,\ x_{72} = x_{73} = 0$

- Only job 7 is on the same machine in both solutions.
- For the other jobs, the two solutions agree only on some machines that the job does *not* use.
- Eight of the 21 binary variables are fixed.

Example: RINS on $P \,||\, C_{\max}$ (3/3)

The sub-MIP with the cutoff $C_{\max} \leq 10$ is solved by Gurobi, which returns

$$J_1 = \{5, 6, 7\}, \qquad J_2 = \{2, 3\}, \qquad J_3 = \{1, 4\},$$

with all loads equal to 9.

RINS finds an optimal solution.

The size of the RINS neighbourhood

- On the KP the incumbent and the LP solution are too similar, and the neighbourhood of RINS has a single free variable.
- On $P \,||\, C_{\max}$ they are very different, and the neighbourhood is large enough to contain the optimum.

Local branching and local search

- With $k = 1$ the neighbourhood of local branching is the flip neighbourhood of the KP in the chapter «Heuristic algorithms: definitions».
- With $k = 2$ it contains the swap neighbourhood as well.
- For the assignment variables of the BPP and of $P \,||\, C_{\max}$ the distance has a simple meaning.

Distance between two assignments

Observation 1

Let $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}$ be two feasible assignments of $P \,||\, C_{\max}$. Then $\Delta(\boldsymbol{x}, \bar{\boldsymbol{x}}) = 2\,r$, where r is the number of jobs assigned to different machines in $\boldsymbol{x}$ and in $\bar{\boldsymbol{x}}$.

Proof

- A job on the same machine in $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}$ contributes 0 to Δ.
- A job on machine h in $\bar{\boldsymbol{x}}$ and on machine $h' \neq h$ in $\boldsymbol{x}$ contributes

$$1 - x_{jh} = 1 \quad \text{and} \quad x_{jh'} = 1,$$

and 0 for the other machines.

□

Move and swap neighbourhoods

- Hence, for $P \,||\, C_{\max}$:
  - the neighbourhood $\Delta(\boldsymbol{x}, \bar{\boldsymbol{x}}) \leq 2$ is the **move** neighbourhood of the chapter «Local search» (one job changes machine),
  - $\Delta(\boldsymbol{x}, \bar{\boldsymbol{x}}) \leq 4$ contains the **swap** neighbourhood (two jobs change machine).
- The same holds for the variables x_{jb} of the BPP.

Example: local branching on the KP (1/2)

Start from $\bar{\boldsymbol{x}}^1 = (1, 0, 1, 0)$, of profit 16, with $k = 1$, $k_{\max} = 3$, $k_{step} = 1$ and $\theta = 1$.

$k = 1$. The neighbours at distance 1 are the flips of $\bar{\boldsymbol{x}}^1$; with the cutoff $\sum_j p_j x_j \geq 17$ no one is feasible. Add $\Delta(\boldsymbol{x}, \bar{\boldsymbol{x}}^1) \geq 2$ and set $k = 2$.

$k = 2$. The sub-MIP with $2 \leq \Delta(\boldsymbol{x}, \bar{\boldsymbol{x}}^1) \leq 2$ finds the swap $(1, 1, 0, 0)$, of profit 17: the new incumbent $\bar{\boldsymbol{x}}^2$. Add $\Delta(\boldsymbol{x}, \bar{\boldsymbol{x}}^1) \geq 3$.

Example: local branching on the KP (2/2)

$k = 2$, **around** $\bar{\boldsymbol{x}}^2$**.** No solution of profit at least 18: add $\Delta(\boldsymbol{x}, \bar{\boldsymbol{x}}^2) \geq 3$ and set $k = 3$.

$k = 3$. No solution of profit at least 18 either; k becomes $4 > k_{\max}$ and the method stops with $\{1, 2\}$, of profit 17: the optimum.

Example: local branching on $P \,||\, C_{\max}$ (1/3)

Start from the solution $\bar{\boldsymbol{x}}^1$ of *LPT*, of makespan 11, with $k = 2$, $k_{\max} = 6$, $k_{step} = 2$ and $\theta = 1$, so that the cutoff is $C_{\max} \leq 10$, then 9, and so on.

$k = 2$. By the observation on the distance between two assignments, the neighbourhood is the move neighbourhood, and *LPT* is a local optimum for it (chapter «Local search»): the sub-MIP is infeasible. Add $\Delta(\boldsymbol{x}, \bar{\boldsymbol{x}}^1) \geq 3$ and set $k = 4$.

Example: local branching on $P \,||\, C_{\max}$ (2/3)

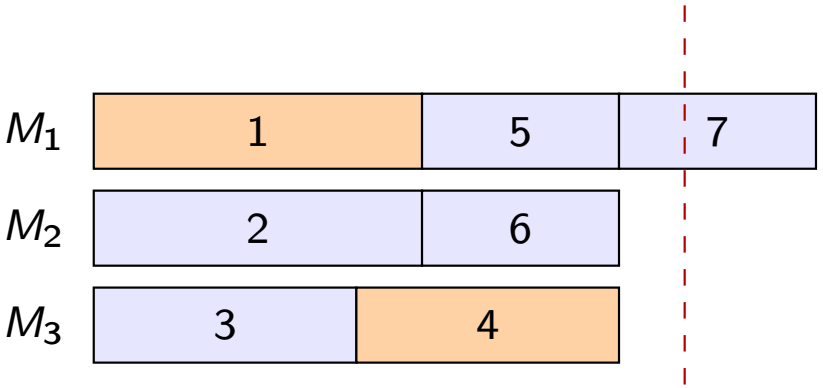
$k = 4$**.** The best solution with two jobs reassigned has makespan 10; the solver returns the swap of jobs 1 and 4, $\bar{\boldsymbol{x}}^2$ in the next figure. Add $\Delta(\boldsymbol{x}, \bar{\boldsymbol{x}}^1) \geq 5$.

$k = 4$**, around** $\bar{\boldsymbol{x}}^2$**.** The solver returns the swap of jobs 4 and 6, of makespan 9: $\bar{\boldsymbol{x}}^3$. Add $\Delta(\boldsymbol{x}, \bar{\boldsymbol{x}}^2) \geq 5$.

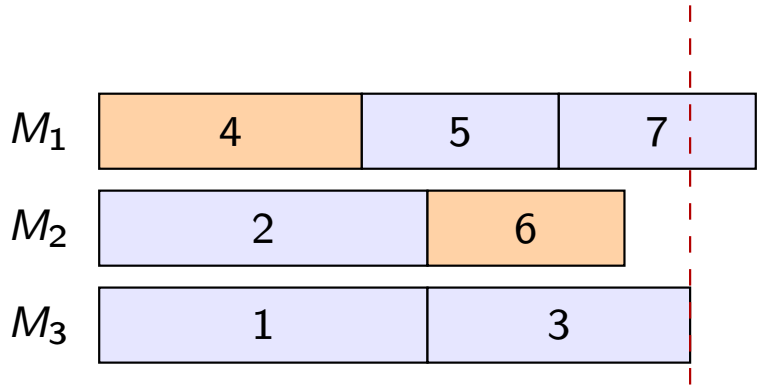
$k = 4$ **and** $k = 6$**, around** $\bar{\boldsymbol{x}}^3$**.** No solution of makespan 8: k becomes $8 > k_{\max}$, and the method stops with makespan 9, the optimum.

Example: local branching on $P \,||\, C_{\max}$ (3/3)

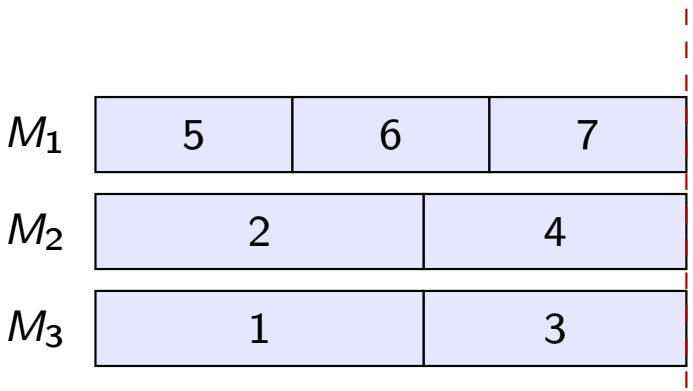
$\bar{x}^1$ (LPT), $C_{\max} = 11$



$\bar{x}^2$, $C_{\max} = 10$



$\bar{x}^3$, $C_{\max} = 9$



The dashed line is the lower bound 9; the jobs swapped to obtain the next solution are coloured.

Local branching as a variable neighbourhood search

- In the language of the chapter «Local search», local branching performs a local search with a neighbourhood whose size is controlled by k, explored exactly by the solver.
- It enlarges the neighbourhood when it gets stuck.
- This is exactly the idea of variable neighbourhood search, with the solver doing the exploration.

Proximity search and optimality

- A proximity sub-MIP that is infeasible with $\theta = 1$ proves that the incumbent is optimal, when the objective takes integer values.
- This is the proposition of the chapter «MIP heuristics» of the module *Integer Linear Programming*.

Example: proximity search on the KP

- From $\bar{\boldsymbol{x}} = (1, 0, 1, 0)$, of profit 16: the closest solution of profit at least 17 is $(1, 1, 0, 0)$, at distance 2 (item 3 out, item 2 in), of profit 17.
- Around it, no solution has profit at least 18.
- By the optimality condition of proximity search (chapter «MIP heuristics» of the module *Integer Linear Programming*), $\{1, 2\}$ is **optimal**.

Example: proximity search on $P \,||\, C_{\max}$ (1/2)

From the solution of LPT, of makespan 11:

iteration	cutoff	min Δ	solution returned	$C_{\max}$
1	$C_{\max} \leq 10$	4	$J_1 = \{5, 6, 7\}$, $J_2 = \{1, 2\}$, $J_3 = \{3, 4\}$	10
2	$C_{\max} \leq 9$	4	$J_1 = \{5, 6, 7\}$, $J_2 = \{1, 4\}$, $J_3 = \{2, 3\}$	9
3	$C_{\max} \leq 8$	—	infeasible	—

Example: proximity search on $P \,||\, C_{\max}$ (2/2)

- By the observation on the distance between two assignments, distance 4 means two jobs reassigned: jobs 1 and 6 at the first iteration, jobs 2 and 4 at the second.
- No solution at distance 2 improves the makespan, as for local branching with $k = 2$.
- At the third iteration the sub-MIP is infeasible, and the optimality condition of proximity search (chapter «MIP heuristics» of the module *Integer Linear Programming*) proves that makespan 9 is optimal.

The proof of optimality comes from the solver

The solver proves the infeasibility of the last sub-MIP with its own dual bounds:

proximity search, used with $\theta = 1$,
is an exact method that improves the incumbent along the way.

Summary (1/2)

RINS:

- on the KP the neighbourhood has a single free variable and the sub-MIP is infeasible
- on $P \,||\, C_{\max}$ eight of the 21 binary variables are fixed, and RINS finds the optimum 9

Local branching:

- $\Delta \leq 2$ is the move neighbourhood, $\Delta \leq 4$ contains the swap neighbourhood
- it finds the optimum on the KP (17) and on $P \,||\, C_{\max}$ (9)

Summary (2/2)

Proximity search:

- it finds the closest improving solution at every iteration
- on the KP and on $P \,||\, C_{\max}$ the last infeasible sub-MIP proves optimality

The cutoff constraint:

- $f(\boldsymbol{x}) \leq f(\bar{\boldsymbol{x}}) - \theta$ with $\theta = 1$, since the objective values are integer

Relax-and-fix and fix-and-optimize on lot sizing

Introduction: Relax-and-fix and fix-and-optimize on lot sizing

- Relax-and-fix and fix-and-optimize work on blocks of integer variables
- Lot sizing, with its periods, is their natural application
- We run relax-and-fix on two instances, and fix-and-optimize on the capacitated one

Blocks of periods

- Relax-and-fix and fix-and-optimize decompose the MIP into blocks of integer variables.
- Lot sizing is a natural application, because its variables are indexed by the periods $t \in T = \{1, 2, \ldots, m\}$ of a planning horizon, and the decisions of a period mainly interact with those of the nearby periods.
- The blocks are the setup variables of consecutive periods.

The lot sizing model: variables

- The model is the $\mathrm{ILP}_{\mathrm{CLSP}}$ model of the chapter «Formulations of lot sizing problems», with the strong big-M value

$$M_t = \min\{cap_t,\, D_{t,m}\}, \qquad D_{t,m} = \sum_{k=t}^{m} d_k;$$

  for an uncapacitated instance, $cap_t = D_{t,m}$.

- Its variables are the production x_t, the inventory s_t at the end of period t and the setup y_t, for every period $t \in T$.

The lot sizing model

Using these variables, an ILP model for the capacitated lot sizing problem (CLS) reads as follows:

$$\min \quad \sum_{t \in T} (sc_t\, y_t + vc_t\, x_t + hc_t\, s_t)$$

$$\begin{aligned}
\text{subject to} \quad & s_{t-1} + x_t = d_t + s_t, && \forall t \in T, \\
& x_t \leq M_t\, y_t, && \forall t \in T, \\
& x_t \leq cap_t, && \forall t \in T, \\
& s_0 = s_m = 0, && \\
& x_t \in \mathbb{Z}_{\geq 0}, && \forall t \in T, \\
& s_t \in \mathbb{Z}_{\geq 0}, && \forall t \in T, \\
& y_t \in \{0, 1\}, && \forall t \in T.
\end{aligned}$$

This ILP model is referred to as the $\mathrm{ILP}_{\mathrm{CLSP}}$ model.

The hard decisions

- Once the setups y are fixed, the rest is an LP with integer data and a network matrix, which has an integer optimal solution.
- The hard decisions are the m binary variables: as in the codes, the heuristics treat x_t and s_t as continuous variables, with $x_t \geq 0$ and $s_t \geq 0$ in place of $x_t \in \mathbb{Z}_{\geq 0}$ and $s_t \in \mathbb{Z}_{\geq 0}$.
- The LP relaxation is weak, because a fraction $y_t = x_t / M_t$ of a setup is enough to produce x_t.

Example: relax-and-fix on an uncapacitated lot sizing instance

- The instance of the module *MIP Modelling*, chapter «Constructive heuristics», has $m = 5$ periods, $d = (20, 10, 30, 40, 10)$, $sc_t = 50$, $vc_t = 0$ and $hc_t = 1$ for every $t \in T$, and no capacity.

- Its LP relaxation has value $\frac{36985}{396} \approx 93.40$, the constructive heuristic least unit cost of that chapter gives 200, and the optimal value is 170.

- Relax-and-fix with one period per block, $B_t = \{y_t\}$, finds a solution of value 170: **optimal**.

Example: relax-and-fix on a capacitated instance (1/3)

- Consider $m = 6$ periods, $d = (50, 30, 10, 10, 40, 10)$, $sc_t = 40$, $vc_t = 0$, $hc_t = 1$ and $cap_t = 80$ for every $t \in T$.
- The LP relaxation has value $\frac{2150}{21} \approx 102.38$.
- The optimal value is 160, with setups in periods 1, 2, 5.
- We run relax-and-fix with $B_t = \{y_t\}$.

Example: relax-and-fix on a capacitated instance (2/3)

step r	value of the MIP of step r	y of the solution	fixing
1	$\frac{2465}{21} \approx 117.38$	$(1, 0.375, 0.143, 0.167, 1, 0)$	$y_1 := 1$
2	$\frac{2780}{21} \approx 132.38$	$(1, 0, 0.143, 0.167, 1, 0)$	$y_2 := 0$
3	$\frac{500}{3} \approx 166.67$	$(1, 0, 1, 0.167, 1, 0)$	$y_3 := 1$
4	170	$(1, 0, 1, 0, 1, 0)$	$y_4 := 0$
5	170	$(1, 0, 1, 0, 1, 0)$	$y_5 := 1$
6	170	$(1, 0, 1, 0, 1, 0)$	$y_6 := 0$

(the fractional values are rounded to three decimals)

Example: relax-and-fix on a capacitated instance (3/3)

- The result has setups in periods 1, 3, 5: production 80, 20, 50, stocks 30, 10, 10 at the end of periods 1, 3, 5, cost $3 \cdot 40 + 50 = 170$.
- The wrong decision is taken at step 2: with the future relaxed, $y_2 = 0$ gives 132.38 and $y_2 = 1$ gives 142.38, so relax-and-fix saves the setup of period 2.
- But then the capacity forces a setup in period 3, which the relaxation had paid only for a fraction 0.143.

Example: fix-and-optimize on the capacitated instance (1/2)

Start from the solution of relax-and-fix of the capacitated example, $\bar{\boldsymbol{y}} = (1, 0, 1, 0, 1, 0)$ of cost 170, with the windows of two consecutive periods $W_t = \{y_t, y_{t+1}\}$, $t = 1, 2, \ldots, 5$.

window	incumbent after the window	
$\{y_1, y_2\}$	170	no improvement
$\{y_2, y_3\}$	160	$\bar{\boldsymbol{y}} = (1, 1, 0, 0, 1, 0)$
$\{y_3, y_4\}$, $\{y_4, y_5\}$, $\{y_5, y_6\}$	160	no improvement
second pass, all windows	160	no improvement: stop

Example: fix-and-optimize on the capacitated instance (2/2)

- The window $\{y_2, y_3\}$ revises the wrong decision of relax-and-fix: the setup moves from period 3 to period 2.
- The plan produces 50, 50, 50 in periods 1, 2, 5, with stocks 20 and 10 at the end of periods 2, 3 and 10 at the end of period 5.
- Cost $3 \cdot 40 + 40 = 160$, the **optimum**.

Summary

Relax-and-fix:

- one block of setups per period, the future relaxed
- uncapacitated instance: 170, optimal; capacitated instance: 170, against the optimum 160
- it is myopic: the cheap decision $y_2 := 0$ forces a setup in period 3

Fix-and-optimize:

- windows of two consecutive periods, re-optimized one at a time
- the window $\{y_2, y_3\}$ revises the wrong decision and reaches the optimum 160

Heuristics inside the solver

Introduction: Heuristics inside the solver

- A MIP solver runs its own heuristics during the branch-and-bound
- We measure their effect on a medium BPP instance, changing the parameters of Gurobi
- We see that, when the bound is strong, finding the solution is the whole difficulty

The heuristics of the solver

- The heuristics of Gurobi, their parameters and the MIP start are described in the chapters «MIP heuristics» and «MIP solvers: what they do» of the module *Integer Linear Programming*.

- A heuristic solution computed outside the solver — by *FFD*, *LPT*, a local search — can be passed to it as a MIP start (the attribute `Start` of the variables).

Example: the heuristics of Gurobi on a medium BPP instance (1/6)

The instance has $n = 60$ items, with weights $w_j = 20 + (s_j \bmod 41)$, where $s_1, s_2, \ldots$ is the Park–Miller sequence from the seed 1, and capacity $c = 100$:

58	29	22	22	55	45	38	40	34	32
43	57	20	57	53	32	40	57	50	33
29	49	37	20	27	22	55	40	48	40
52	24	21	48	31	36	38	46	30	20
31	58	40	28	54	50	26	31	27	54
26	26	22	30	33	55	27	50	47	26

Example: the heuristics of Gurobi on a medium BPP instance (2/6)

- The total weight is 2271, so $LB = \lceil 2271/100 \rceil = 23$.
- *FFD* uses 24 bins, and the $\mathrm{ILP}_{\mathrm{BPP}}$ model is built with $m = 24$ bins.
- Gurobi 13.0.3 is run on one thread with a limit of 10 *work units* (a deterministic measure of the computing effort, roughly a second each), with different parameters.

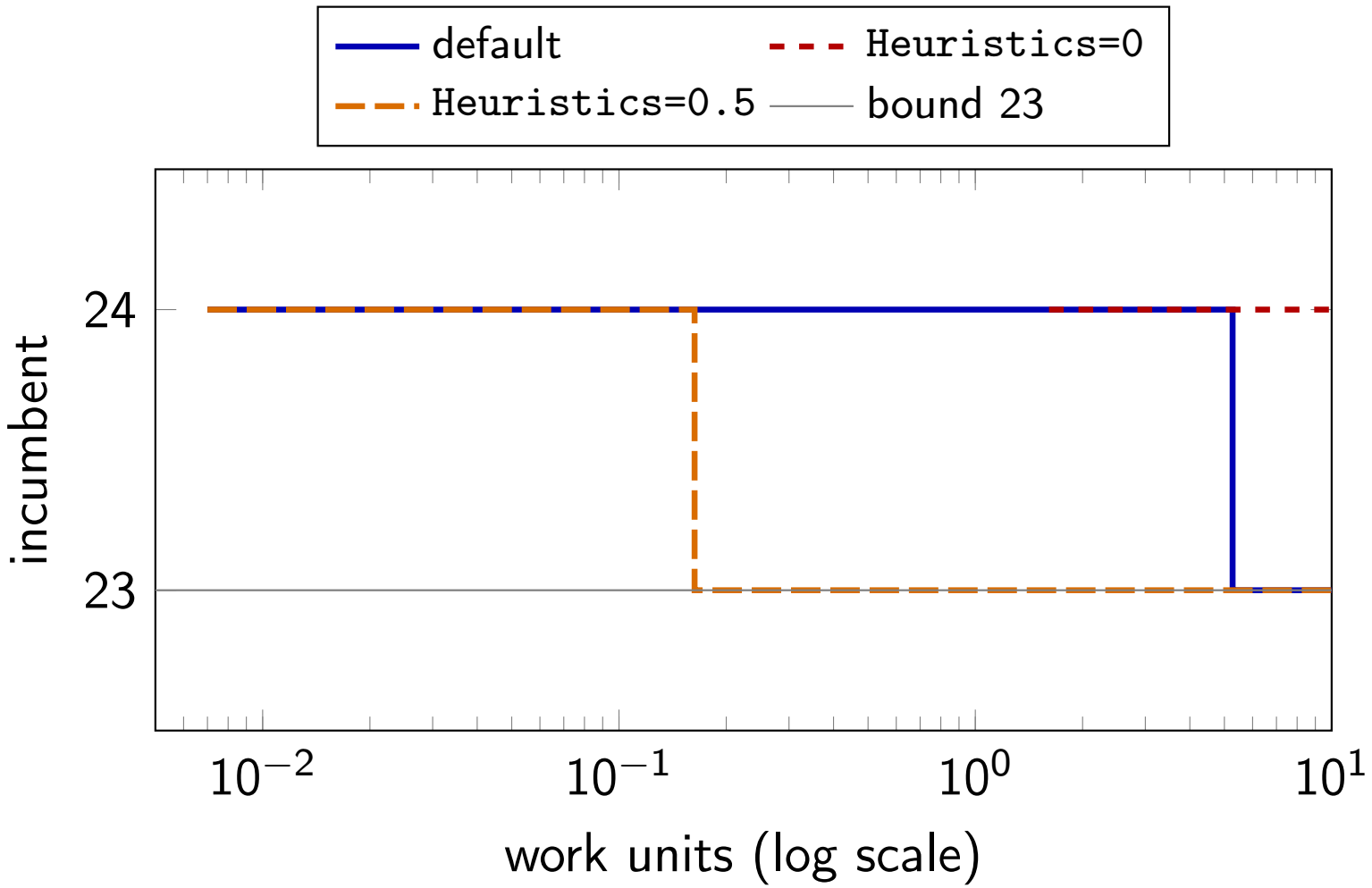
Example: the heuristics of Gurobi on a medium BPP instance (3/6)

parameters	incumbent	bound	nodes	work
default	23	23	2533	5.27
`Heuristics=0`	24	23	4809	10.00
`Heuristics=0.5`	23	23	1	0.16
`MIPFocus=1`	23	23	1	0.16
`NoRelHeurWork=1`	23	23	2533	6.48
`RINS=0`	23	23	2959	5.86

Example: the heuristics of Gurobi on a medium BPP instance (4/6)

parameters	incumbents found (work: value)
default	0.007: 24; 5.273: 23
`Heuristics=0`	1.610: 24
`Heuristics=0.5`	0.007: 24; 0.163: 23
`MIPFocus=1`	0.007: 24; 0.160: 23
`NoRelHeurWork=1`	0.007: 24; 6.477: 23
`RINS=0`	0.007: 24; 5.861: 23

Example: the heuristics of Gurobi on a medium BPP instance (5/6)



Example: the heuristics of Gurobi on a medium BPP instance (6/6)

- With the default parameters, a solution of 24 bins is found at once, and the optimal one only after 2533 nodes.
- Spending more effort in the heuristics (`Heuristics=0.5` or `MIPFocus=1`) finds 23 at the root, about 30 times faster.
- Without heuristics the first solution arrives after 1.61 work units, and the optimum is not found within the limit.
- Since the bound is 23 from the root, the whole difficulty of this instance is primal: finding the solution, not proving it optimal.

A qualitative lesson

- The numbers depend on the instance, on the version of the solver and on the machine; a single instance proves nothing (chapter «Heuristic algorithms: definitions»).
- The lesson is qualitative: on instances whose bound is strong, as the BPP with $LB = z(I)$, the time to solve is the time to find a good solution.
- The heuristics of the solver — or a good MIP start — decide it.

Summary

Heuristics inside the solver:

- Gurobi runs its own heuristics; a solution found outside can be passed as a MIP start
- on the medium BPP instance the bound 23 is known from the root

The effect of the parameters:

- more effort in the heuristics finds 23 at the root, about 30 times faster
- without heuristics the optimum is not found within 10 work units
- when the bound is strong, the difficulty is primal

What the MIP-based heuristics give

Introduction: What the MIP-based heuristics give

- We compare the heuristics of the previous chapters with the MIP-based heuristics on all the instances of the chapter
- The key difference is the dual bound: with a MIP model, the gap of a solution is known
- A gap equal to 0 certifies that the solution is optimal

Primal and dual bounds

- The chapter «Heuristic algorithms: definitions» ended with a warning: a heuristic gives only one side of the gap, and without a dual bound nothing can be said about the quality of its solution.
- The heuristics of the previous chapters give *only* primal bounds.

The heuristic knows how far it is from the optimum

- A MIP-based heuristic works with a MIP model, and the model comes with its LP relaxation.
- Run inside a MIP solver, it comes with the dual bound of the branch-and-bound as well.
- For the first time, the heuristic **knows how far it is from the optimum** while it runs: the gap of the chapter «Heuristic algorithms: definitions» is computed at every moment, and when it is 0 the solution is *proved* optimal.

The gaps of the two families (1/2)

Heuristics of the previous chapters:

instance	solution	dual bound	gap
KP	*GR*: 16	LP: 17	6.25%
BPP	*FF*: 4	$\lceil 21/7 \rceil = 3$	33.3%
TSP	*NN*: 25	two-edge: 17	47.1%
$P \mid\mid C_{\max}$	*LPT*: 11	9	22.2%
lot sizing, capacitated	RF: 170	LP: 102.38	66.0%
medium BPP	*FFD*: 24	23	4.3%

The gaps of the two families (2/2)

MIP-based heuristics:

instance	solution	dual bound	gap
KP	LB, PS: 17	PS: 17	0%
BPP	pump (seed 7): 3	LP: 3	0%
TSP	rounding: 18	LP: 18	0%
$P \mid\mid C_{\max}$	RINS, LB, PS: 9	LP: 9	0%
lot sizing, capacitated	FO: 160	solver: 160	0%
medium BPP	Gurobi: 23	23	0%

Where the dual bounds come from

- On the KP the integer rounding of the LP value is $\lfloor 17.75 \rfloor = 17$, and proximity search proves the optimality of 17 by itself, by the optimality condition of proximity search (chapter «MIP heuristics» of the module *Integer Linear Programming*).

- On the capacitated lot sizing instance the LP bound is weak, and the gap is closed by the branch-and-bound of the solver, which proves that 160 is optimal.

Solutions with a certificate

A MIP-based heuristic does not only find solutions:
inside a MIP solver it works with a dual bound,

so the gap of its solution is always known,
and a gap equal to 0 certifies optimality.

Summary

Previous chapters:

- only primal bounds: gaps from 4.3% to 66.0% on the instances of the chapter

MIP-based heuristics:

- the model gives the LP bound, the solver the bound of the branch-and-bound
- on every instance of the chapter the gap is 0%: the solutions are proved optimal

Typical mistakes

Introduction: Typical mistakes

- We collect the mistakes that are made most often with the MIP-based heuristics
- Each of them was visible in one of the examples of the chapter
- Check them before trusting the output of a method

Typical mistakes (1/4)

Observation 2: Rounding a problem with equality constraints

Independent roundings of an assignment constraint almost never sum to 1*: on the BPP and on* $P \,||\, C_{\max}$ *rounding fails.*

Observation 3: Believing that a dive can undo a decision

A dive is one path of the tree, without backtracking: the first wrong fixing (on the KP, $x_2 := 0$*) is never revised.*

Typical mistakes (2/4)

Observation 4: Using the feasibility pump to get a good solution

After the first LP the pump ignores the objective: it returns a feasible solution, possibly with empty bins left open.

Observation 5: Trusting hard fixing

RINS fixes the variables on which the LP and the incumbent agree, so it cannot find a better solution that changes one of them.

Typical mistakes (3/4)

Observation 6: Forgetting the cutoff constraint

Without $f(\boldsymbol{x}) \leq f(\bar{\boldsymbol{x}}) - \theta$ the sub-MIP returns the incumbent solution again; with it, an infeasible sub-MIP proves a local optimum.

Observation 7: Claiming optimality without the conditions

The optimality condition of proximity search (chapter «MIP heuristics» of the module Integer Linear Programming) needs $\theta = 1$ and integer objective values on all the feasible solutions.

Typical mistakes (4/4)

Observation 8: Ignoring the myopia of relax-and-fix

The relaxed future is optimistic: a decision that looks cheap can force a setup later.

Observation 9: Generalizing from one instance or one solver run

Paths and times depend on the version of the solver and on the tie-breaking rules, so the lessons are qualitative.

Summary

Constructing a solution:

- rounding fails on equality constraints; a dive never revises a wrong fixing
- the pump looks for feasibility, not for quality

Improving and certifying:

- hard fixing can exclude the better solutions; never forget the cutoff
- optimality from proximity search needs $\theta = 1$ and integer objective values
- relax-and-fix is myopic; one instance or one run proves nothing

Python implementation

Introduction: `Python` implementation

- The methods of the chapter are run with Gurobi, from Python
- We look at short excerpts: how the models are built with `gurobipy`, and how the experiment with the parameters of Gurobi is run
- The complete codes are in the folder `MIP_HEUR_CODE`

The codes of the chapter

- The codes of the methods (rounding, diving, the feasibility pump, RINS, local branching, proximity search, relax-and-fix and fix-and-optimize) are those of the chapter «MIP heuristics» of the module *Integer Linear Programming*.
- Here we give the models of the classical problems (`models.py`) and the experiment with the parameters of Gurobi (`gurobi_heuristics.py`).
- The folder `MIP_HEUR_CODE` contains all the functions and `instances.py`, with the instances of the chapter and of the exercises.
- The command `python3 run_tests.py` runs every example of the chapter and every instance of the exercises, checking each value against the expected value, written in the file.

The ILP_{KP} model in gurobipy

```
def build_kp(p, w, c):
    m = gp.Model("KP", env=ENV)
    n = len(p)
    x = m.addVars(n, vtype=GRB.BINARY, name="x")
    m.setObjective(gp.quicksum(p[j] * x[j]
                   for j in range(n)), GRB.MAXIMIZE)
    m.addConstr(gp.quicksum(w[j] * x[j]
                for j in range(n)) <= c,
                name="capacity")
    m.update()
    return m
```

The ILP$_{P||C_{max}}$ model in gurobipy

```
def build_pcm(t, m):
    model = gp.Model("PCmax", env=ENV)
    n = len(t)
    x = model.addVars(n, m, vtype=GRB.BINARY,
                      name="x")
    Cmax = model.addVar(name="Cmax")
    model.setObjective(Cmax, GRB.MINIMIZE)
    model.addConstrs((x.sum(j, "*") == 1
        for j in range(n)), name="assign")
    model.addConstrs((-gp.quicksum(t[j] * x[j, h]
        for j in range(n)) + Cmax >= 0
        for h in range(m)), name="load")
    model.update()
    return model
```

The parameters of Gurobi

```
s = m.copy()
s.Params.WorkLimit = work_limit
for name, value in params.items():
    s.setParam(name, value)
found = []
# called by Gurobi at every new incumbent
def callback(model, where):
    if where == GRB.Callback.MIPSOL:
        found.append(
            (model.cbGet(GRB.Callback.WORK),
             model.cbGet(GRB.Callback.MIPSOL_OBJ)))
s.optimize(callback)
```

Summary

The codes:

- the methods are those of the chapter «MIP heuristics» of the module *Integer Linear Programming*
- `models.py` builds the models of the classical problems with `gurobipy`
- `gurobi_heuristics.py` runs the experiment with the parameters of Gurobi, recording every new incumbent with a callback

Tests:

- `python3 run_tests.py` checks every example and every exercise against the expected values

In summary

Introduction: In summary

- We collect the key ideas of the chapter on the MIP heuristics for the classical problems
- The dashboard gathers the best results on the reference instances

Summary (1/2)

- The MIP-based heuristics of the chapter «MIP heuristics» of the module *Integer Linear Programming* apply to the classical problems through their MIP models.
- Rounding and diving build a solution from the LP; the feasibility pump looks only for feasibility; RINS, local branching and proximity search improve an incumbent solution with a sub-MIP.
- Relax-and-fix builds a solution along the time, fix-and-optimize improves it window by window.
- Inside a MIP solver the dual bound is available: the gap of the heuristic solution is known, and a gap equal to 0 certifies optimality.

Summary (2/2)

Dashboard of the reference instances. Best solution found by the methods of this chapter against the dual bound (max for the KP, min for the others):

instance	solution	dual bound	gap
KP	17	17	0%
BPP	3	3	0%
$P \mid\mid C_{\max}$	9	9	0%
lot sizing, capacitated	160	160	0%
medium BPP	23	23	0%