



SAPIENZA
UNIVERSITÀ DI ROMA

Integer Linear Programming

Volume 4 — MIP heuristics

Rounding, diving, feasibility pump, RINS, local branching, relax-and-fix and fix-and-optimize. Examples and exercises with complete solutions.

Fabio Furini

fabiofurini.github.io

Department of Computer, Control and Management Engineering — Sapienza University of Rome

Online version and downloadable material:
fabiofurini.github.io/integer-linear-programming/
github.com/fabiofurini/integer-linear-programming

Academic year 2026–2027

Contents

V	MIP heuristics	2
10	MIP heuristics	3
10.1	MIP-based heuristics	4
10.2	The examples of the chapter	5
10.3	Rounding	6
10.4	Diving	8
10.5	The feasibility pump	10
10.6	Improving an incumbent with a sub-MIP	15
10.7	Relax-and-fix and fix-and-optimize	22
10.8	Heuristics inside the solver	25
10.9	What MIP-based heuristics give	26
10.10	Python implementation	27
10.11	Typical mistakes	34
10.12	In summary	35



Part V

MIP heuristics



MIP heuristics

Prerequisites

- From this module, the chapters «Primal and dual bounds» and «LP-based branch-and-bound algorithms»: the LP relaxation, the lower and upper bounds, the optimality gap, and the branching operations.
- From this module, the chapter «MIP solvers: what they do»: the incumbent solution, the sub-problems solved by a solver, and the heuristics inside it.
- From the module *Linear Programming*: the [dual simplex algorithm \(Chapter 10\)](#), which re-optimizes an LP after a change of a bound.

Objectives

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

- explain how a heuristic algorithm uses the LP relaxation and a MIP solver;
- execute by hand rounding, diving, the feasibility pump, RINS, local branching and proximity search on a small MIP;
- execute relax-and-fix and fix-and-optimize on a MIP whose variables are split into blocks;
- decide which method to use when a feasible solution is missing, or when an incumbent solution must be improved;
- compute the gap of a heuristic solution when a dual bound is available.

10.1 MIP-based heuristics

Definition 10.1: MIP in general form

A **MIP** with N variables $\mathbf{x} = (x_1, x_2, \dots, x_N)$ reads

$$\min f(\mathbf{x}) \quad (10.1a)$$

$$\text{subject to } A\mathbf{x} \leq b, \quad (10.1b)$$

$$l_j \leq x_j \leq u_j, \quad \forall j \in \{1, 2, \dots, N\}, \quad (10.1c)$$

$$x_j \in \mathbb{Z}, \quad \forall j \in \mathcal{D}, \quad (10.1d)$$

where f is a linear function and $\mathcal{D} \subseteq \{1, 2, \dots, N\}$ is the set of the indices of the **integer variables**. The feasible region of the **LP relaxation** is

$$\mathcal{L} = \{\mathbf{x} \in \mathbb{R}^N : A\mathbf{x} \leq b, l \leq \mathbf{x} \leq u\},$$

and $z(\text{LP}) = \min\{f(\mathbf{x}) : \mathbf{x} \in \mathcal{L}\}$ is the value of the LP relaxation.

- A maximization problem, such as the car example, is handled in the same way, exchanging min and max, $\leq$ and $\geq$ in the comparisons of objective values.

Definition 10.2: MIP-based heuristic

A **heuristic algorithm** returns a feasible solution of a MIP problem (10.1) without any guarantee of optimality. It is **MIP-based** (a **matheuristic**) if it uses the LP relaxation and a MIP solver to search for the solution.

- A heuristic gives a primal bound (an upper bound for a minimization problem, a lower bound for a maximization problem), never a dual bound. The branch-and-bound algorithm of this module finds the incumbent solution only by chance, as a leaf of the tree.
- The optimal solution of the **LP relaxation** tells *where to look*: it is the solution of a simpler problem, often close to the good solutions of the original one.
- The **MIP solver** explores *exactly* a set of solutions far too large to be enumerated, provided that the set is described by linear constraints.

Definition 10.3: incumbent, rounding and completion

- A feasible solution $\bar{\mathbf{x}}$ of the MIP (10.1), the best one known, is the **incumbent**, and $f(\bar{\mathbf{x}})$ is its value.
- The **rounding** of a real number v to the nearest integer is $\lceil v \rceil = \lfloor v + \frac{1}{2} \rfloor$, so that a value $\frac{1}{2}$ is rounded up.
- A point $\bar{\mathbf{x}}$ with $\bar{x}_j \in \mathbb{Z}$ for all $j \in \mathcal{D}$ is **completed** by fixing the integer variables to the values $\bar{x}_j$ and solving the LP on the continuous variables.

- Throughout the chapter, $\tilde{\mathbf{x}}$ denotes an optimal solution of an LP relaxation, and $\bar{\mathbf{x}}$ a point whose components $\bar{x}_j$, $j \in \mathcal{D}$, are integer.
- If the LP of the completion has an optimal solution, $\bar{\mathbf{x}}$ is turned into a feasible solution of the MIP; if it is infeasible, so is $\bar{\mathbf{x}}$. When all the variables are integer, the completion simply checks the constraints.
- The methods are built on five ideas, one or two per section.

idea	methods
round the relaxation	rounding; diving (fix one variable, re-solve, repeat)
chase feasibility	the feasibility pump
fix where relaxation and incumbent agree	RINS
bound the distance from the incumbent	local branching; proximity search
decompose into blocks of variables	relax-and-fix; fix-and-optimize

- The examples are solved with the MIP solver Gurobi, used from Python through `gurobipy`; the module *MIP Modelling* (fabiofurini.github.io/mip-modelling, chapter 3 «From the model to Python/Gurobi») explains how to use it. When an LP has several optimal solutions, the text reports the one returned by the solver; all the codes are in the last section.

10.2 The examples of the chapter

- The methods are illustrated on the **car example** of this module and on four small problems of the **catalogue of MIP problems**. All of them are ILP problems in canonical maximization form, with integer data and $x_j \in \mathbb{Z}_{\geq 0}$ for every j , so that $\mathcal{D} = \{1, 2, \dots, N\}$.
- The **car example** reads

$$\max 3x_1 + 2x_2 \quad \text{s.t.} \quad 10x_1 + 4x_2 \leq 45, \quad 4x_1 + 14x_2 \leq 49, \quad x_1, x_2 \in \mathbb{Z}_{\geq 0}$$

problem	objective max	constraints
P7	$3x_1 + x_2$	$x_1 + 4x_2 \leq 12, \quad 2x_1 + x_2 \leq 5$
P42	$3x_1 + 2x_2 + 4x_3$	$x_1 + x_2 + 2x_3 \leq 4, \quad 2x_1 + 3x_3 \leq 5, \quad 2x_1 + x_2 + 3x_3 \leq 7$
P46	$5x_1 + 6x_2 + 10x_3$	$9x_1 + 2x_2 + 2x_3 \leq 11, \quad 2x_1 + 6x_2 + 6x_3 \leq 16, \quad 4x_1 + 2x_2 + 7x_3 \leq 10$
P72	$7x_1 + 6x_2 + 5x_3 - 2x_4 + 3x_5$	$x_1 + 3x_2 + 5x_3 - 2x_4 + 2x_5 \leq 4, \quad 4x_1 + 2x_2 - 2x_3 + x_4 + x_5 \leq 3, \quad 2x_1 + 4x_2 + 4x_3 - 2x_4 + 5x_5 \leq 5, \quad 3x_1 + x_2 + 2x_3 - x_4 - 2x_5 \leq 1$

- The following table reports the value $z(\text{LP})$ of the LP relaxation, the upper bound $\lfloor z(\text{LP}) \rfloor$ (the objective takes integer values), and the optimal value $z(\text{ILP})$, all computed with the exact simplex algorithm and checked with Gurobi.

problem	variables	constraints	$z(\text{LP})$	$\lfloor z(\text{LP}) \rfloor$	$z(\text{ILP})$
car	2	2	$\frac{31}{2}$	15	14
P7	2	2	$\frac{15}{2}$	7	7
P42	3	3	$\frac{21}{2}$	10	10
P46	3	3	$\frac{499}{25}$	19	16
P72	5	4	$\frac{507}{59}$	8	7

Example 10.1: The LP solutions of the examples

problem	optimal solution $\tilde{\mathbf{x}}$ of the LP relaxation
car	$(\frac{7}{2}, \frac{5}{2})$
P7	$(\frac{5}{2}, 0)$
P42	$(\frac{5}{2}, \frac{3}{2}, 0)$
P46	$(\frac{17}{25}, \frac{49}{25}, \frac{12}{25})$
P72	$(\frac{39}{59}, 0, \frac{91}{59}, \frac{166}{59}, \frac{37}{59})$

The first three solutions have at most two fractional components, with fractional part $\frac{1}{2}$; the last two are the examples where the methods have more room to fail.

10.3 Rounding

Definition 10.4: rounding

Rounding is the simplest MIP-based heuristic. It solves the LP relaxation, rounds every integer variable of the LP solution to the nearest integer, and completes the continuous variables. It returns a feasible solution, or it fails if the rounded point violates a constraint.

- The pseudo-code of rounding reads as follows:

Algorithm**Algorithm:** Rounding**Input:** MIP (10.1) with integer variables $\mathcal{D}$ **Output:** Feasible solution $\bar{x}$, or FAIL

```

1 Function Rounding(MIP)
2    $\tilde{x} :=$  optimal solution of the LP relaxation           //  $\min\{f(x) : x \in \mathcal{L}\}$ 
3   for  $j \in \mathcal{D}$  do
4      $\tilde{x}_j \leftarrow \lfloor \tilde{x}_j \rfloor$                        // round to the nearest integer
5   end
6   fix  $x_j := \tilde{x}_j$  for all  $j \in \mathcal{D}$  and solve the LP on the continuous variables
7   if the LP is feasible then
8     return  $\bar{x}$  completed with the continuous part of the LP solution
9   else
10    return FAIL                                     // the rounded point violates a constraint
11  end

```

- **Why it can work.** When the formulation is strong, the LP solution is close to an optimal integer solution, and many of its integer variables are already integer: rounding changes only a few of them, by less than $\frac{1}{2}$ each.
- **Why it can fail.** Rounding treats the variables one at a time, while the constraints link them. A constraint with all non-negative coefficients and right-hand side $b_i \geq 0$ stays satisfied if the variables are rounded *down*, and one of the form $\geq$ if they are rounded *up*.
- An **equality** constraint is almost always violated by independent roundings: two fractional values that sum to 1 become, for instance, 1 and 1.

Observation 10.5

Let all the coefficients of A be non-negative, $b \geq 0$ and $l = 0$. If $\tilde{x} \in \mathcal{L}$, then the point $\lfloor \tilde{x} \rfloor$, obtained by rounding down every integer variable, belongs to $\mathcal{L}$.

Proof. For every j , $0 \leq \lfloor \tilde{x}_j \rfloor \leq \tilde{x}_j \leq u_j$, so the bounds hold. For every row i , since $a_{ij} \geq 0$, $\sum_j a_{ij} \lfloor \tilde{x}_j \rfloor \leq \sum_j a_{ij} \tilde{x}_j \leq b_i$. $\square$

- The car example satisfies the assumptions of the observation: rounding down is always feasible, and in general not optimal.

Example 10.2: Rounding on the car example and on $P42$

Car example. The LP solution $\tilde{x} = (\frac{7}{2}, \frac{5}{2})$ is rounded to $\bar{x} = (4, 3)$, which violates the first constraint:

$$10 \cdot 4 + 4 \cdot 3 = 52 > 45.$$

Rounding fails. Rounding down instead gives $(3, 2)$: $38 \leq 45$, $40 \leq 49$, of value 13, while $z(\text{ILP}) = 14$.

P42. The LP solution $(\frac{5}{2}, \frac{3}{2}, 0)$ is rounded to $(3, 2, 0)$, which violates the second constraint: $2 \cdot 3 = 6 > 5$. **Rounding fails.** Rounding down gives $(2, 1, 0)$, of value 8, while $z(\text{ILP}) = 10$.

P72. The LP solution is rounded to $(1, 0, 2, 3, 1)$, which violates the first constraint: $1 + 10 - 6 + 2 = 7 > 4$. **Rounding fails.**

- The two examples summarize the method: the rounding to the nearest integer fails when several variables are fractional at the same time and the constraints are tight; the rounding down is safe, but leaves part of the LP value on the table.

10.4 Diving

- Rounding takes all its decisions at once, on the basis of a single LP solution. **Diving** takes them one at a time, and after each decision it asks the LP again: the LP solution is re-computed with the decisions taken so far, and it adapts to them.

Definition 10.6: diving

Diving repeatedly solves the LP relaxation, selects one integer variable with a fractional value, fixes it to an integer value, and solves the LP again. It stops with a feasible solution when the LP solution is integer on all the integer variables, and it fails when the LP becomes infeasible. A *rule* decides which variable is fixed and to which value.

- The pseudo-code of diving reads as follows:

Algorithm

Algorithm: Diving

Input: MIP (10.1); a rule; for guided diving, an incumbent $\bar{x}$

Output: Feasible solution, or FAIL

```

1 Function Diving(MIP, rule)
2   while true do
3     solve the LP relaxation with the current bounds  $l, u$ 
4     if the LP is infeasible then
5       return FAIL                                // the dive ends in an infeasible LP
6     end
7      $\tilde{x} :=$  optimal solution of the LP
8      $F := \{j \in \mathcal{D} : \tilde{x}_j \notin \mathbb{Z}\}$                 // fractional integer variables
9     if  $F = \emptyset$  then
10      return  $\tilde{x}$                                     // integer on  $\mathcal{D}$ : feasible
11    end
12    select  $j \in F$  and a value  $v \in \mathbb{Z}$  according to the rule
13     $l_j \leftarrow v$ ;  $u_j \leftarrow v$                 // fix  $x_j := v$ 
14  end
```

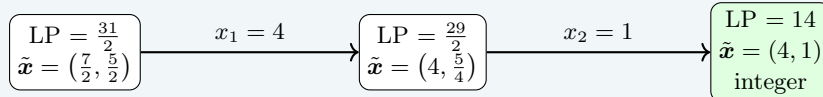
- **Fractional diving.** Select the variable that is *least* fractional, i.e., the $j \in F$ that minimizes $|\tilde{x}_j - \lfloor \tilde{x}_j \rfloor|$, and fix it to $v = \lfloor \tilde{x}_j \rfloor$. The LP is almost sure about this variable, so the fixing changes the LP solution as little as possible. In the codes of this chapter, ties are broken in favour of the variable with the smallest index.
- **Guided diving** [3]. With an incumbent $\bar{x}$, select the $j \in F$ that minimizes $|\tilde{x}_j - \bar{x}_j|$, and fix it to $v = \bar{x}_j$. The incumbent guides the dive toward a region of good solutions, and the LP decides in which order its values are adopted.
- Many other rules exist: *coefficient diving* fixes the variable that appears in the fewest constraints that can be violated by rounding it; *pseudo-cost diving* uses the history of the branch-and-bound tree; *vector-length diving* is tailored to set covering constraints. Modern solvers run several of them.
- **Diving and branch-and-bound.** Fixing $x_j := \lfloor \tilde{x}_j \rfloor$ is a branching decision. A dive is a single path from the root of the branch-and-bound tree to a leaf, always taking one of the two children, without ever coming back: depth-first search without backtracking.
- **Cost.** Every step solves an LP that differs from the previous one by a single bound, so the dual simplex algorithm re-optimizes it starting from the previous basis in a few iterations. A dive with $|\mathcal{D}|$ integer variables needs at most $|\mathcal{D}|$ LPs, usually far fewer, since an LP solution has many integer components already.
- **Failure.** The LP becomes infeasible when the fixings taken so far are incompatible. Implementations then undo the last fixing and try the other value (a single backtrack), or give up.

Example 10.3: Fractional diving on the car example

Step 1. The LP solution is $\tilde{x} = (\frac{7}{2}, \frac{5}{2})$, of value $\frac{31}{2}$. Both variables are at distance $\frac{1}{2}$ from the nearest integer; the tie is broken by the index: $x_1 := \lfloor \frac{7}{2} \rfloor = 4$.

Step 2. With $x_1 = 4$ the constraints become $4x_2 \leq 5$ and $14x_2 \leq 33$, so the LP solution is $\tilde{x} = (4, \frac{5}{4})$, of value $12 + \frac{5}{2} = \frac{29}{2}$. Fix $x_2 := \lfloor \frac{5}{4} \rfloor = 1$.

Step 3. The LP solution $(4, 1)$, of value 14, is integer: the dive returns $(4, 1)$, which is the **optimal** solution of the ILP problem.



Example 10.4: Guided diving on the car example and on $P7$

Car example. The incumbent is $\bar{x} = (3, 2)$, of value 13 (rounding down). In the LP solution $(\frac{7}{2}, \frac{5}{2})$ both variables are at distance $\frac{1}{2}$ from $\bar{x}$; fix $x_1 := 3$. The new LP solution is $(3, \frac{37}{14})$, of value $\frac{100}{7}$; fix $x_2 := \bar{x}_2 = 2$. The LP solution $(3, 2)$ is integer, of value 13: the dive **returns the incumbent**, without improving it.

$P7$. The incumbent is $\bar{x} = (2, 0)$, of value 6 (rounding down of $(\frac{5}{2}, 0)$). The only fractional variable is x_1 : fix $x_1 := 2$. The LP solution $(2, 1)$, of value 7, is integer: guided diving finds the **optimal** solution after one fixing, even if the incumbent has $\bar{x}_2 = 0$, because the LP rebuilds the variables that are not fixed.

- The incumbent only decides the *values* of the fixed variables: on the car example both fixed values agree with the poor incumbent, on $P7$ the fixed value is the right one.

Example 10.5: A dive that fails on $P42$ and on $P72$

$P42$. The LP solution is $(\frac{5}{2}, \frac{3}{2}, 0)$; x_1 and x_2 are at distance $\frac{1}{2}$, and the tie is broken by the index: $x_1 := \lfloor \frac{5}{2} \rfloor = 3$. The second constraint gives $2 \cdot 3 + 3x_3 \leq 5$, which is violated for every $x_3 \geq 0$: the LP is infeasible, **the dive fails at the first step**.

$P72$. The fractional parts of the LP solution are $\frac{39}{59}, \frac{91}{59}, \frac{166}{59}, \frac{37}{59}$ (for x_1, x_3, x_4, x_5), and the least fractional is x_4 , at distance $\frac{11}{59}$ from 3: $x_4 := 3$. The new LP has value $\frac{728}{85}$ and solution $(\frac{56}{85}, 0, \frac{138}{85}, 3, \frac{52}{85})$; the least fractional variable is x_1 , rounded to 1. Fixing $x_1 := 1$ and $x_4 := 3$ makes the LP **infeasible**.

- In both cases the first fixings follow the LP, which does not see that the constraints, together, leave no room. A dive that fails gives no solution at all: this is the reason for the feasibility pump of the next section.

10.5 The feasibility pump

- Rounding and diving fail on $P42$ and on $P72$, and on many MIPs they fail most of the time: for a general MIP, even *deciding whether a feasible solution exists* is NP-complete. The **feasibility pump** of Fischetti, Glover and Lodi [4] gives up the objective function and concentrates on a single goal: to find a feasible solution.
- The idea is to work with two points at the same time. A point of $\mathcal{L}$ satisfies all the linear constraints, but may be fractional; an integer point is integer, but may violate the constraints. A feasible solution is a point that is both. The pump makes the two points chase each other until they coincide, and to do so it needs to measure how far apart they are.

Definition 10.7: Distance from an integer point

Let $\bar{\mathbf{x}}$ be a point with $\bar{x}_j \in \mathbb{Z}$ for all $j \in \mathcal{D}$. The **distance** of a point $\mathbf{x} \in \mathbb{R}^N$ from $\bar{\mathbf{x}}$ is

$$\Delta(\mathbf{x}, \bar{\mathbf{x}}) = \sum_{j \in \mathcal{D}} |x_j - \bar{x}_j|,$$

the L_1 distance computed on the integer variables only.

- When all the integer variables are **binary**, $|x_j - \bar{x}_j|$ equals x_j if $\bar{x}_j = 0$ and $1 - x_j$ if $\bar{x}_j = 1$, for every $x_j \in [0, 1]$. The distance is then a *linear* function of $\mathbf{x}$:

$$\Delta(\mathbf{x}, \bar{\mathbf{x}}) = \sum_{j \in \mathcal{D}: \bar{x}_j=0} x_j + \sum_{j \in \mathcal{D}: \bar{x}_j=1} (1 - x_j) \quad \text{for binary variables}$$

- For a general integer variable the same holds if $\bar{x}_j = l_j$ (the term is $x_j - l_j$) or $\bar{x}_j = u_j$ (the term is $u_j - x_j$). Otherwise $|x_j - \bar{x}_j|$ is replaced by an auxiliary variable d_j with the two constraints $d_j \geq x_j - \bar{x}_j$ and $d_j \geq \bar{x}_j - x_j$; at the minimum, $d_j = |x_j - \bar{x}_j|$. In all cases, the point of $\mathcal{L}$ closest to $\bar{\mathbf{x}}$ is computed by solving **one LP**.

Definition 10.8: feasibility pump

The *feasibility pump* builds two sequences of points: points $\tilde{\mathbf{x}}^0, \tilde{\mathbf{x}}^1, \tilde{\mathbf{x}}^2, \dots$ of $\mathcal{L}$, which satisfy the linear constraints, and integer points $\bar{\mathbf{x}}^0, \bar{\mathbf{x}}^1, \bar{\mathbf{x}}^2, \dots$, which satisfy the integrality constraints. It starts from the LP solution $\tilde{\mathbf{x}}^0$; each integer point is the *rounding* of the last point of $\mathcal{L}$, and each point of $\mathcal{L}$ is the *projection* of the last integer point, i.e., the point of $\mathcal{L}$ at minimum distance from it. It stops when the distance is 0: the two points coincide and are a feasible solution. When the rounding returns an integer point already visited, the pump *perturbs* it.

- The pseudo-code of the feasibility pump reads as follows:

Algorithm

Algorithm: Feasibility pump

Input: MIP (10.1); maximum number of iterations *maxIter*; seed of the random numbers

Output: Feasible solution, or FAIL

```

1 Function FeasibilityPump(MIP, maxIter, seed)
2    $\tilde{x} :=$  optimal solution of the LP relaxation
3   if  $\tilde{x}_j \in \mathbb{Z}$  for all  $j \in \mathcal{D}$  then
4     return  $\tilde{x}$ 
5   end
6    $\bar{x} \leftarrow \lfloor \tilde{x} \rfloor$  // round the integer components
7   for  $it := 1$  to maxIter do
8      $\tilde{x} \leftarrow \arg \min \{ \Delta(x, \bar{x}) : x \in \mathcal{L} \}$  // projection: one LP
9     if  $\Delta(\tilde{x}, \bar{x}) = 0$  then
10      return  $\tilde{x}$  //  $\tilde{x}$  is integer on  $\mathcal{D}$ : feasible
11    end
12     $\hat{x} \leftarrow \lfloor \tilde{x} \rfloor$ 
13    if  $\hat{x}_j = \bar{x}_j$  for all  $j \in \mathcal{D}$  then // short cycle
14       $j \leftarrow \arg \max_{k \in \mathcal{D}} |\tilde{x}_k - \bar{x}_k|$ 
15      move  $\hat{x}_j$  by one unit toward  $\tilde{x}_j$  // flip
16    end
17    if  $\hat{x}$  equals one of the last 3 points  $\bar{x}$  then // long cycle
18      for  $j \in \mathcal{D}$  do
19         $\rho_j \leftarrow$  uniform random number in  $[-0.3, 0.7]$ 
20        if  $|\tilde{x}_j - \hat{x}_j| + \max\{\rho_j, 0\} > \frac{1}{2}$  then
21           $\hat{x}_j \leftarrow 1 - \hat{x}_j$  // restart
22        end
23      end
24    end
25     $\bar{x} \leftarrow \hat{x}$ 
26  end
27  return FAIL

```

- **Why the pump works.** Each projection moves the point of $\mathcal{L}$ toward the last integer point, and each rounding moves the integer point to the nearest one. The distance between the two sequences tends to decrease, and the points are *pumped* toward a region where $\mathcal{L}$ contains integer points. Each iteration costs one LP and a rounding.
- **Short cycles.** If the rounding of the projection gives back the same integer point, the pump would repeat the same two points forever. It then changes the components of $\bar{x}$ on which $\tilde{x}$ is most uncertain, those with the largest $|\tilde{x}_j - \bar{x}_j|$.
- The original pump flips a random number of components between 10 and 30. On the small instances of these notes one flip is enough: the code flips the single component with the largest $|\tilde{x}_j - \bar{x}_j|$ (the first one, in case of ties).

- **Long cycles.** The pump can also cycle among several integer points. When the new integer point equals one of the last ones, the original pump makes a **restart**: a binary component is flipped when $|\tilde{x}_j - \hat{x}_j| + \max\{\rho_j, 0\} > \frac{1}{2}$, with ρ_j uniformly random in $[-0.3, 0.7]$, so that the components where $\tilde{x}_j$ is far from $\hat{x}_j$ are flipped with high probability.
- **Random numbers.** In the codes of this chapter, random numbers are produced by the *minimal standard* generator of Park and Miller, with integer arithmetic on 64 bits:

$$seed \leftarrow (16807 \cdot seed) \bmod (2^{31} - 1), \quad u = \frac{seed}{2^{31} - 1} \in (0, 1),$$

and in the pump $\rho_j = u - 0.3$. Fixing the initial seed makes every run reproducible.

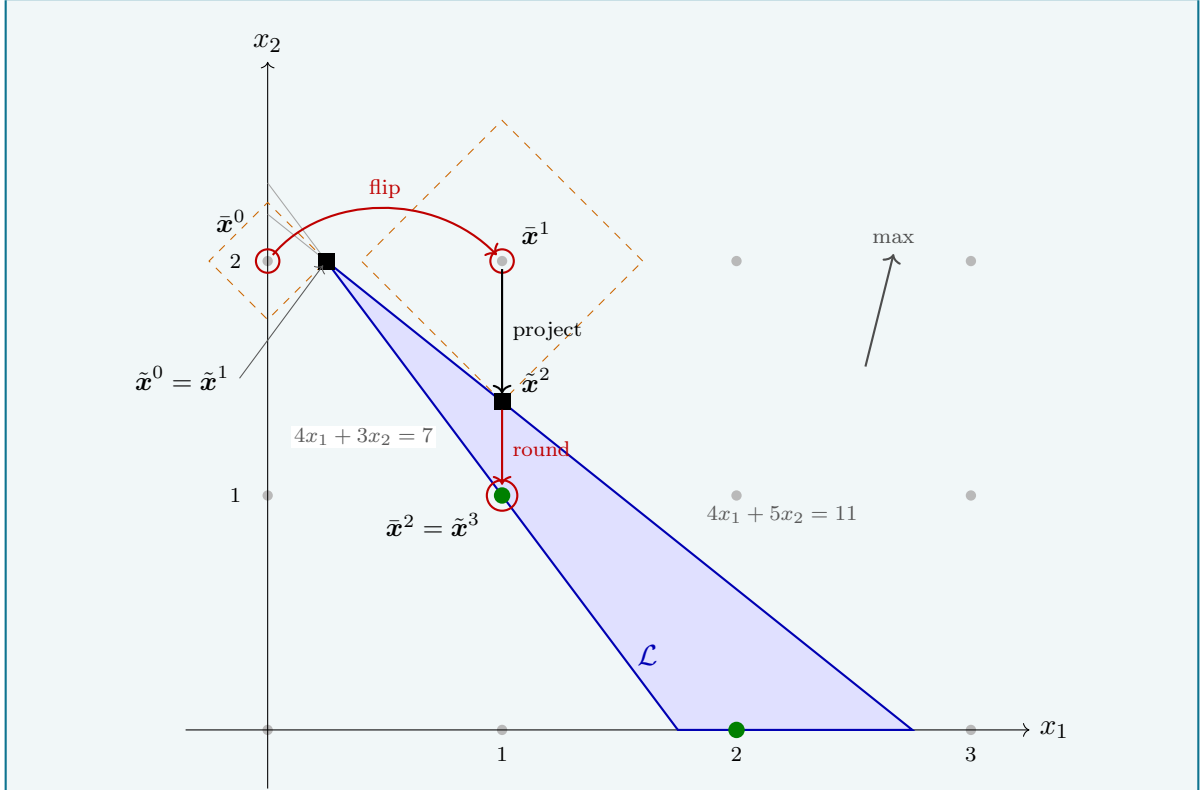
- **The objective.** After the first LP, the original objective f is ignored: the pump finds a feasible solution, but not necessarily a good one. The *objective feasibility pump* of Achterberg and Berthold [1] minimizes a combination $(1 - \alpha) \Delta(\mathbf{x}, \bar{\mathbf{x}}) + \alpha f(\mathbf{x})$ with a weight α that decreases at every iteration: the solutions found are much better, at the price of a few more iterations.
- **General integer variables.** The pump was designed for binary variables, where the flip changes $\bar{x}_j$ into $1 - \bar{x}_j$; for general integer variables it moves $\bar{x}_j$ by one unit toward $\tilde{x}_j$, and the distance uses the auxiliary variables d_j [2].

Example 10.6: The feasibility pump in two variables

Consider the integer program

$$\begin{aligned} \max \quad & x_1 + 4x_2 \\ \text{subject to} \quad & 4x_1 + 3x_2 \geq 7 \\ & 4x_1 + 5x_2 \leq 11 \\ & x_1 \in \{0, 1, \dots, 5\} \\ & x_2 \in \{0, 1, \dots, 5\} \end{aligned}$$

The region $\mathcal{L}$ is the triangle with vertices $(\frac{1}{4}, 2)$, $(\frac{7}{4}, 0)$, $(\frac{11}{4}, 0)$, which contains only two integer points, $(1, 1)$ and $(2, 0)$. The figure shows the two sequences of points: squares for the points of $\mathcal{L}$, circles for the integer points, and the diamonds $\{\mathbf{x} : \Delta(\mathbf{x}, \bar{\mathbf{x}}) \leq r\}$ of the two projections.



The iterations are the following.

k	integer point	projection $\tilde{x}^k$	$\Delta(\tilde{x}^k, \bar{x}^{k-1})$	next integer point
0	—	LP optimum $(\frac{1}{4}, 2)$	—	rounding: $\bar{x}^0 = (0, 2)$
1	$\bar{x}^0 = (0, 2)$	$(\frac{1}{4}, 2)$	$\frac{1}{4}$	$[\tilde{x}^1] = \bar{x}^0$: flip x_1 , $\bar{x}^1 = (1, 2)$
2	$\bar{x}^1 = (1, 2)$	$(1, \frac{7}{5})$	$\frac{3}{5}$	rounding: $\bar{x}^2 = (1, 1)$
3	$\bar{x}^2 = (1, 1)$	$(1, 1)$	0	stop: feasible

Iteration 0. The LP optimum is the vertex $(\frac{1}{4}, 2)$, of value $\frac{33}{4}$. Its rounding $(0, 2)$ violates the first constraint: $4 \cdot 0 + 3 \cdot 2 = 6 < 7$.

Iteration 1. With $\bar{x}^0 = (0, 2)$, x_1 is at its lower bound and $\Delta(x, \bar{x}^0) = x_1 + |x_2 - 2|$. The L_1 distance from a point to a segment that is neither horizontal, vertical nor of slope ± 1 is minimized at an endpoint or where the segment crosses the horizontal or the vertical line through the point; so the candidates are the three vertices and these crossings. The horizontal line $x_2 = 2$ touches $\mathcal{L}$ only at the vertex $(\frac{1}{4}, 2)$, and the vertical line $x_1 = 0$ misses it; the closest point is $\tilde{x}^1 = (\frac{1}{4}, 2)$, at distance $\frac{1}{4}$. Its rounding is $(0, 2) = \bar{x}^0$ again: a short cycle. The component with the largest $|\tilde{x}_j^1 - \bar{x}_j^0|$ is x_1 ($\frac{1}{4}$ against 0), which moves one unit toward $\frac{1}{4}$: $\bar{x}^1 = (1, 2)$.

Iteration 2. The point $(1, 2)$ violates the second constraint ($14 > 11$). The vertical line $x_1 = 1$ crosses $\mathcal{L}$ between $(1, 1)$ and $(1, \frac{7}{5})$, at distances 1 and $\frac{3}{5}$ from $(1, 2)$; the horizontal line $x_2 = 2$ gives $(\frac{1}{4}, 2)$, at distance $\frac{3}{4}$; the other two vertices are farther. Hence $\tilde{x}^2 = (1, \frac{7}{5})$, and its rounding $(1, 1)$ is a new point.

Iteration 3. The point $(1, 1)$ belongs to $\mathcal{L}$ ($4 + 3 = 7 \geq 7$ and $4 + 5 = 9 \leq 11$): its distance from $\mathcal{L}$ is 0, and the pump returns $(1, 1)$, of value 5. It is optimal, since the other integer point $(2, 0)$ has value 2.

Example 10.7: The feasibility pump on the car example and on P72

Car example. Rounding the LP solution gives $\bar{x}^0 = (4, 3)$, which is infeasible.

k	projection $\tilde{x}^k$	$\Delta(\tilde{x}^k, \bar{x}^{k-1})$	next integer point
1	$(\frac{7}{2}, \frac{5}{2})$	1	$\lceil \tilde{x}^1 \rceil = \bar{x}^0$: flip x_1 , $\bar{x}^1 = (3, 3)$
2	$(3, \frac{37}{14})$	$\frac{5}{14}$	$\lceil \tilde{x}^2 \rceil = \bar{x}^1$: flip x_2 , $\bar{x}^2 = (3, 2)$
3	$(3, 2)$	0	stop: feasible

At iteration 1 the two components of $\tilde{x}^1$ are at the same distance $\frac{1}{2}$ from $\bar{x}^0$, and the flip takes the first one. The pump needs **two flips** and returns $(3, 2)$, of value 13: it is feasible, but not optimal ($z(\text{ILP}) = 14$), because after the first LP the objective is ignored.

P72. Rounding and diving fail (Examples 10.2, 10.5). The pump rounds the LP solution to $\bar{x}^0 = (1, 0, 2, 3, 1)$, which violates the first constraint, and needs three projections and one flip (of x_3 , from 2 to 1) to reach the feasible solution

$$\bar{x} = (0, 0, 1, 3, 1), \quad f(\bar{x}) = 2,$$

very far from $z(\text{ILP}) = 7$ and from the upper bound 8.

- 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 of the next sections, which start from an incumbent.

10.6 Improving an incumbent with a sub-MIP

- The methods of the previous sections build a solution from the LP relaxation. The three methods of this section start from an **incumbent** $\bar{x}$, computed by any heuristic — rounding, diving, the feasibility pump — and try to improve it.
- They all follow the same pattern: add to the MIP some constraints that restrict it to a set of solutions *related to* $\bar{x}$, and solve the restricted problem, called the **sub-MIP**, with the MIP solver.
- The sub-MIP is a neighbourhood of $\bar{x}$, but a very large one, explored exactly by the solver rather than by enumeration. It is the idea of **large neighbourhood search**, with the solver as the repair method.
- Only solutions *better* than the incumbent are of interest. With an improvement step $\theta > 0$, the sub-MIP contains the **cutoff constraint**

$$f(x) \leq f(\bar{x}) - \theta \quad (\text{minimization}), \quad f(x) \geq f(\bar{x}) + \theta \quad (\text{maximization}).$$

- When f takes integer values on all the feasible solutions, as for all the examples of this chapter, $\theta = 1$ excludes exactly the solutions that are not better than $\bar{x}$. If the sub-MIP with the cutoff is infeasible, the neighbourhood contains no better solution: the solver has *proved* that $\bar{x}$ is a local optimum for that neighbourhood.
- The methods differ in *which* constraints define the neighbourhood: RINS fixes variables (**hard fixing**), local branching and proximity search bound the number of variables that change (**soft fixing**).

10.6.1 RINS

Definition 10.9: RINS

RINS (Relaxation Induced Neighbourhood Search) [3] compares the incumbent $\bar{x}$ with the solution $\tilde{x}$ of the LP relaxation. The integer variables on which the two solutions *agree* are fixed to their common value; the sub-MIP on the remaining variables, with the cutoff constraint, is solved by the MIP solver.

- The pseudo-code of RINS reads as follows:

Algorithm

Algorithm: RINS

Input: MIP (10.1); incumbent $\bar{x}$; improvement step θ

Output: A solution at least as good as $\bar{x}$

```

1 Function RINS(MIP,  $\bar{x}$ ,  $\theta$ )
2    $\tilde{x} :=$  optimal solution of the LP relaxation
3   for  $j \in \mathcal{D}$  do
4     if  $\tilde{x}_j = \bar{x}_j$  then
5        $\text{fix } x_j := \bar{x}_j$                                 // LP and incumbent agree
6     end
7   end
8   add the cutoff constraint  $f(x) \leq f(\bar{x}) - \theta$ 
9   solve the sub-MIP with the MIP solver (with a node limit)
10  if the sub-MIP has a feasible solution  $x'$  then
11    return  $x'$                                           // better than  $\bar{x}$ 
12  else
13    return  $\bar{x}$                                           // no better solution in the neighbourhood
14  end

```

- **Why the agreement.** The LP solution is optimistic and possibly infeasible, the incumbent is feasible and possibly poor: two advisers with opposite defects. Where they agree, the value is probably right in good solutions too; where they disagree, there is something to decide, and the decision is left to the solver.

- **The size of the neighbourhood is not chosen.** It is the number of variables on which $\tilde{x}$ and $\bar{x}$ disagree: small if the incumbent is already close to the LP solution, large otherwise. For this reason the sub-MIP is solved with a limit on the number of nodes.
- **RINS inside branch-and-bound.** Danna, Rothberg and Le Pape apply RINS at the nodes of the branch-and-bound tree, with the LP solution of the node: deeper nodes have different LP solutions, hence different neighbourhoods of the same incumbent. Every modern MIP solver runs RINS in this way.
- **Hard fixing.** A fixed variable cannot change in the sub-MIP. If a better solution needs to change a variable on which $\tilde{x}$ and $\bar{x}$ agree, RINS cannot find it.

Example 10.8: RINS on $P7$ and on $P42$

$P7$. The incumbent is $\bar{x} = (2, 0)$, of value 6 (rounding down), and the LP solution is $\tilde{x} = (\frac{5}{2}, 0)$. They agree on $x_2 = 0$, which is fixed. The sub-MIP

$$\max 3x_1 \quad \text{subject to} \quad x_1 \leq 12, \quad 2x_1 \leq 5, \quad 3x_1 \geq 7, \quad x_1 \in \mathbb{Z}_{\geq 0}$$

is infeasible: the second constraint gives $x_1 \leq 2$, the cutoff $x_1 \geq 3$. RINS returns the incumbent. The optimal solution $(2, 1)$ has $x_2 = 1$, but x_2 is fixed to 0 because the LP and the incumbent agree on it.

$P42$. The incumbent is $\bar{x} = (2, 1, 0)$, of value 8 (rounding down), and $\tilde{x} = (\frac{5}{2}, \frac{3}{2}, 0)$. They agree on $x_3 = 0$ only. The sub-MIP on x_1, x_2 with the cutoff $3x_1 + 2x_2 \geq 9$ is solved by Gurobi, which returns $(2, 2, 0)$, of value 10: **RINS finds an optimal solution.**

- On $P7$ the variable that must change is exactly the one on which the LP and the incumbent agree; on $P42$ the agreement concerns a variable that does not need to change, and the neighbourhood contains the optimum.

10.6.2 Local branching

- RINS decides *which* variables may change. Local branching [5] decides only *how many* of them may change, and leaves the choice of which ones to the solver. For a MIP whose integer variables are binary (or, with the auxiliary variables d_j of the distance, general integers), the **local branching constraint** with radius $k \in \mathbb{Z}_{\geq 1}$ is

$$\Delta(x, \bar{x}) = \sum_{j \in \mathcal{D}: \bar{x}_j=1} (1 - x_j) + \sum_{j \in \mathcal{D}: \bar{x}_j=0} x_j \leq k$$

- $\Delta(x, \bar{x})$ is the distance of Definition 10.7 for binary variables: on binary points, it counts the variables whose value differs from the incumbent. The constraint is linear, so it can be added to the MIP like any other constraint.

Definition 10.10: local branching

Local branching solves, with the MIP solver, the sub-MIP made of the MIP, the cutoff constraint and the local branching constraint $\Delta(\mathbf{x}, \bar{\mathbf{x}}) \leq k$. If it finds a better solution, the new solution becomes the incumbent and the neighbourhood is centred on it. If it proves that the neighbourhood contains no better solution, the radius k is enlarged. In both cases the explored neighbourhood is excluded from the following sub-MIPs by the *reverse* constraint $\Delta(\mathbf{x}, \bar{\mathbf{x}}) \geq k + 1$.

Algorithm**Algorithm:** Local branching

Input: MIP (10.1); incumbent $\bar{\mathbf{x}}$; radius k ; maximum radius $k_{\max}$; step k_{step} ; improvement step θ

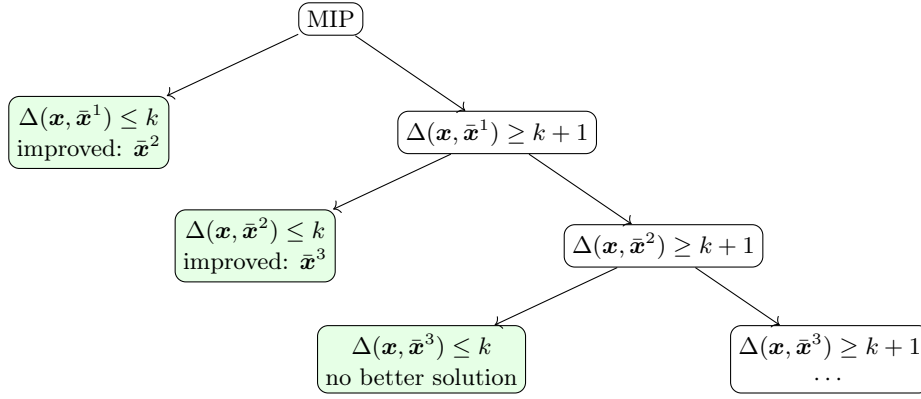
Output: A solution at least as good as $\bar{\mathbf{x}}$

```

1 Function LocalBranching(MIP,  $\bar{\mathbf{x}}$ ,  $k$ ,  $k_{\max}$ ,  $k_{\text{step}}$ ,  $\theta$ )
2    $\mathcal{R} := \emptyset$  // reverse constraints added so far
3   while  $k \leq k_{\max}$  do
4     solve the MIP with the constraints  $\mathcal{R}$ ,  $\Delta(\mathbf{x}, \bar{\mathbf{x}}) \leq k$  and  $f(\mathbf{x}) \leq f(\bar{\mathbf{x}}) - \theta$ 
5      $\mathcal{R} \leftarrow \mathcal{R} \cup \{\Delta(\mathbf{x}, \bar{\mathbf{x}}) \geq k + 1\}$  // never explore it again
6     if a feasible solution  $\mathbf{x}'$  is found then
7        $\bar{\mathbf{x}} \leftarrow \mathbf{x}'$  // re-centre the neighbourhood
8     else
9        $k \leftarrow k + k_{\text{step}}$  // no better solution within distance  $k$ 
10    end
11  end
12  return  $\bar{\mathbf{x}}$ 

```

- **Soft fixing.** With $|\mathcal{D}|$ binary variables, the neighbourhood $\Delta(\mathbf{x}, \bar{\mathbf{x}}) \leq k$ contains $\sum_{i=0}^k \binom{|\mathcal{D}|}{i}$ binary points: for k small it is much smaller than the whole space, but it does not commit to any particular variable. Any k variables may change.
- **A branching scheme.** The name comes from the original use of the constraint as a *branching rule*: the left branch $\Delta(\mathbf{x}, \bar{\mathbf{x}}) \leq k$ is solved first, and the right branch $\Delta(\mathbf{x}, \bar{\mathbf{x}}) \geq k + 1$ contains the rest of the problem. If the left branch improves the incumbent, the right branch is split again around the new incumbent; the tree is a long chain of right branches:



- **The radius.** k must be small enough for the sub-MIP to be solved quickly, and large enough for the neighbourhood to contain better solutions. The original method uses small values compared with $|\mathcal{D}|$, a time limit on each sub-MIP, and *diversification* — a larger k , or a jump to a different solution — when no improvement is found.
- **Local branching and local search.** With $k = 1$ the neighbourhood contains the points obtained from $\bar{x}$ by changing a single integer variable by one unit (a binary variable flips); with larger k it contains the points that differ from $\bar{x}$ in several variables, or by several units. It is a local search whose neighbourhood is explored exactly by the solver.

Example 10.9: Local branching on $P7$ and on $P72$

$P7$. Start from $\bar{x}^1 = (2, 0)$, of value 6, with $k = 1$, $k_{\max} = 3$, $k_{\text{step}} = 1$ and $\theta = 1$.

$k = 1$. The sub-MIP with $\Delta(x, \bar{x}^1) \leq 1$ and the cutoff $3x_1 + x_2 \geq 7$ returns $(2, 1)$, of value 7: the new incumbent $\bar{x}^2$, found at distance 1 (the variable x_2 changes by one unit). Add $\Delta(x, \bar{x}^1) \geq 2$.

$k = 1, 2, 3$, **around $\bar{x}^2$** . No solution of value at least 8: k becomes $4 > k_{\max}$ and the method stops with $(2, 1)$, the optimum. Where RINS failed, local branching lets the solver choose that x_2 changes.

$P72$. Start from the solution of the feasibility pump, $\bar{x}^1 = (0, 0, 1, 3, 1)$, of value 2, with the same parameters.

k	around	solution returned	value
1	$\bar{x}^1$	$\bar{x}^2 = (0, 0, 1, 2, 1)$, at distance 1	4
1	$\bar{x}^2$	no solution of value ≥ 5	—
2	$\bar{x}^2$	$\bar{x}^3 = (0, 1, 1, 2, 0)$, at distance 2	7
2, 3	$\bar{x}^3$	no solution of value ≥ 8	—

With $k_{\max} = 3$ the method stops with value 7, the optimum, after two improvements. The radius enlarged from 1 to 2 because the improvement of x_2 and x_5 together needs two units of distance.

- Local branching performs a local search with a neighbourhood whose size is controlled by k , explored exactly by the solver, and it enlarges the neighbourhood when it gets stuck — exactly the idea of variable neighbourhood search, with the solver doing the exploration.

10.6.3 Proximity search

- Local branching keeps the objective and constrains the distance. **Proximity search** of Fischetti and Monaci [6] does the opposite: it keeps the cutoff as a constraint and *minimizes* the distance.

Definition 10.11: proximity search

The **proximity sub-MIP** around the incumbent $\bar{x}$ reads

$$\begin{aligned}
 & \min \quad \Delta(\mathbf{x}, \bar{\mathbf{x}}) \\
 & \text{subject to} \quad A\mathbf{x} \leq b, \\
 & \quad l_j \leq x_j \leq u_j, \quad \forall j \in \{1, 2, \dots, N\}, \\
 & \quad f(\mathbf{x}) \leq f(\bar{\mathbf{x}}) - \theta, \\
 & \quad x_j \in \mathbb{Z}, \quad \forall j \in \mathcal{D}.
 \end{aligned}$$

Proximity search solves it, replaces the incumbent with its solution, and repeats, until the proximity sub-MIP is infeasible.

Algorithm

Algorithm: Proximity search

Input: MIP (10.1); incumbent $\bar{x}$; improvement step θ

Output: A solution at least as good as $\bar{x}$

```

1 Function ProximitySearch(MIP,  $\bar{x}$ ,  $\theta$ )
2   while true do
3     solve min{ $\Delta(\mathbf{x}, \bar{\mathbf{x}}) : \mathbf{x}$  feasible for the MIP,  $f(\mathbf{x}) \leq f(\bar{\mathbf{x}}) - \theta$ }
4     if the proximity sub-MIP is infeasible then
5       | return  $\bar{x}$  // no solution better than  $f(\bar{\mathbf{x}}) - \theta$ 
6     end
7      $\mathbf{x}' :=$  its optimal solution
8     fix  $x_j := x'_j$  for  $j \in \mathcal{D}$  and re-optimize  $f$  on the continuous variables
9      $\bar{x} \leftarrow \mathbf{x}'$  // the closest improving solution
10  end

```

- **Why minimize the distance.** The original objective is often what makes a MIP hard: its LP relaxation drives the solver toward fractional points far from feasible solutions. In the proximity sub-MIP the objective is the distance, and the LP solution stays close to $\bar{x}$, which is almost feasible: the heuristics of the solver work well, and a feasible solution is usually found quickly.

- In practice the proximity sub-MIP is not solved to optimality: any feasible solution is an improvement.
- **No radius to choose.** The size of the move is decided by the sub-MIP itself: the smallest change that improves the objective by at least θ .
- **Re-optimizing the continuous part.** The solution of the sub-MIP has the right integer part, but its continuous variables were chosen to minimize the distance, not f . Fixing the integer variables and solving the LP on the continuous ones gives the true value of the new incumbent.
- **The step θ .** A large θ asks for large improvements, hence large moves and hard sub-MIPs; a small θ gives easy sub-MIPs and many iterations. With $\theta = 1$ and integer objective values, the method also proves optimality.

Proposition 10.12

If f takes integer values on all the feasible solutions of the MIP and the proximity sub-MIP around $\bar{x}$ with $\theta = 1$ is infeasible, then $f(\bar{x}) = z(I)$.

Proof. The feasible region of the proximity sub-MIP is the set of the feasible solutions x of the MIP with $f(x) \leq f(\bar{x}) - 1$; it does not depend on the objective Δ . If it is empty, every feasible solution has $f(x) > f(\bar{x}) - 1$, hence $f(x) \geq f(\bar{x})$ since both values are integer. So $\bar{x}$ is optimal. $\square$

Example 10.10: Proximity search on $P7$ and on $P72$

$P7$. From $\bar{x} = (2, 0)$, of value 6: the closest solution of value at least 7 is $(2, 1)$, at distance 1. Around it, no solution has value at least 8 (the LP bound is $\frac{15}{2}$): by Proposition 10.12, $(2, 1)$ is **optimal**.

$P72$. From the solution of the feasibility pump, of value 2:

iteration	cutoff	min Δ	solution returned	value
1	$f(x) \geq 3$	1	$(0, 0, 1, 2, 1)$	4
2	$f(x) \geq 5$	2	$(0, 1, 1, 2, 0)$	7
3	$f(x) \geq 8$	—	infeasible	—

At the third iteration the sub-MIP is infeasible, and Proposition 10.12 proves that 7 is optimal, although the upper bound $\lfloor z(LP) \rfloor = 8$ of the root does not.

- The proof of optimality comes from the solver, which 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.

10.7 Relax-and-fix and fix-and-optimize

- The variables of many MIPs come in **blocks** with a natural order, for instance because they are indexed by a sequence, and the decisions of a block mainly interact with those of the nearby blocks. The two methods of this section exploit a partition of the integer variables into blocks: they solve a sequence of small MIPs instead of a large one.
- Once the integer variables are fixed, the rest is an LP: the hard decisions are the integer variables, and the LP relaxation is the tool that estimates the effect of the decisions not yet taken.

10.7.1 Relax-and-fix

Definition 10.13: relax-and-fix

Relax-and-fix splits the integer variables into blocks $B_1, B_2, \dots, B_p$, in a given order. At step r it solves the MIP in which the variables of B_r are integer, those of $B_1, \dots, B_{r-1}$ are fixed to the values decided at the previous steps, and those of $B_{r+1}, \dots, B_p$ are *relaxed* (continuous). Then it fixes the variables of B_r to their values, and moves to the next block. The solution of the last step is feasible.

Algorithm

Algorithm: Relax-and-fix

Input: MIP (10.1); partition $B_1, B_2, \dots, B_p$ of $\mathcal{D}$, in a given order

Output: Feasible solution, or FAIL

```

1 Function RelaxAndFix(MIP,  $B_1, \dots, B_p$ )
2   for  $r := 1$  to  $p$  do
3     solve the MIP with:  $x_j$  fixed for  $j \in B_1 \cup \dots \cup B_{r-1}$ ,  $x_j$  integer for  $j \in B_r$ ,  $x_j$ 
       continuous for  $j \in B_{r+1} \cup \dots \cup B_p$ 
4     if it is infeasible then
5       return FAIL
6     end
7      $x' :=$  its optimal solution
8     fix  $x_j := x'_j$  for all  $j \in B_r$ 
9   end
10  return  $x'$                                      // all the integer variables are integer

```

- **Why.** The decisions of the first blocks are taken with care (integer), while the others are only estimated (relaxed). Each MIP has the integer variables of a single block, so it is small.
- **Myopia.** The relaxed blocks are optimistic: the relaxation can buy a fraction of a decision for a fraction of its cost. A decision that looks good because the rest is cheap may turn out to be bad when the rest becomes integer. The fixed decisions can even make a later step infeasible.

- **Variants.** Larger blocks give better solutions and harder MIPs; *overlapping* blocks solve $B_r \cup B_{r+1}$ as integer but fix only B_r , to see a little further.

Example 10.11: Relax-and-fix on $P46$

The integer variables are x_1, x_2, x_3 . The LP relaxation has value $\frac{499}{25}$, the upper bound is 19 and $z(\text{ILP}) = 16$, attained by $(0, 1, 1)$.

One variable per block, $B_1 = \{x_1\}$, $B_2 = \{x_2\}$, $B_3 = \{x_3\}$.

step r	value of the MIP of step r	solution of the MIP	fixing
1	$\frac{296}{15} \approx 19.73$	$(0, \frac{26}{15}, \frac{14}{15})$	$x_1 := 0$
2	$\frac{56}{3} \approx 18.67$	$(0, 2, \frac{2}{3})$	$x_2 := 2$
3	12	$(0, 2, 0)$	$x_3 := 0$

The result is $(0, 2, 0)$, of value 12, with a gap of $\frac{19-12}{12} \approx 58.3\%$. The wrong decision is taken at step 2: with x_3 relaxed, $x_2 = 2$ gives $\frac{56}{3} \approx 18.67$ and $x_2 = 1$ gives $\frac{122}{7} \approx 17.43$, so $x_2 := 2$; but then the second constraint leaves room only for $x_3 \leq \frac{2}{3}$, and x_3 becomes 0 when it is integer.

Larger blocks, $B_1 = \{x_1\}$, $B_2 = \{x_2, x_3\}$. Step 1 is the same. At step 2 the MIP on x_2, x_3 is solved with both variables integer: it returns $(0, 1, 1)$, of value 16, the **optimum**.

- The size of the blocks trades the quality of the solution against the difficulty of the MIPs: the larger block sees the interaction between x_2 and x_3 that the single-variable blocks miss.

10.7.2 Fix-and-optimize

Definition 10.14: fix-and-optimize

Fix-and-optimize improves an incumbent $\bar{x}$. It considers a sequence of *windows* $W_1, W_2, \dots, W_q \subseteq \mathcal{D}$, typically consecutive periods. For each window it fixes all the integer variables *outside* the window to their values in $\bar{x}$, and solves the sub-MIP on the variables of the window, with the cutoff constraint. A better solution becomes the incumbent. The passes over the windows are repeated until a whole pass gives no improvement.

Algorithm

Algorithm: Fix-and-optimize**Input:** MIP (10.1); incumbent $\bar{x}$; windows $W_1, \dots, W_q$; improvement step θ **Output:** A solution at least as good as $\bar{x}$

```

1 Function FixAndOptimize(MIP,  $\bar{x}$ ,  $W_1, \dots, W_q$ ,  $\theta$ )
2   improved := true
3   while improved do
4     improved  $\leftarrow$  false
5     for  $r := 1$  to  $q$  do
6       solve the MIP with  $x_j := \bar{x}_j$  for  $j \in \mathcal{D} \setminus W_r$  and  $f(x) \leq f(\bar{x}) - \theta$ 
7       if a feasible solution  $x'$  is found then
8          $\bar{x} \leftarrow x'$ ; improved  $\leftarrow$  true
9       end
10    end
11  end
12  return  $\bar{x}$ 

```

Example 10.12: Fix-and-optimize on *P46* and on *P72*

P46. Start from the solution of relax-and-fix, $\bar{x} = (0, 2, 0)$, of value 12, with the windows $W_1 = \{x_1\}$ and $W_2 = \{x_2, x_3\}$ and $\theta = 1$.

window	incumbent after the window	
$\{x_1\}$	12	no improvement ($x_1 = 1$ violates the first constraint)
$\{x_2, x_3\}$	16	$\bar{x} = (0, 1, 1)$
second pass, both windows	16	no improvement: stop

With the three windows of one variable each, $\{x_1\}$, $\{x_2\}$, $\{x_3\}$, no window improves $(0, 2, 0)$: it is a local optimum for those windows. The window $\{x_2, x_3\}$ revises the wrong decision of relax-and-fix and reaches the optimum 16.

P72. Start from the solution of the feasibility pump, $(0, 0, 1, 3, 1)$, of value 2, with the windows $\{x_1, x_2\}$, $\{x_3, x_4\}$, $\{x_5\}$. The first window gives no improvement; the second gives $(0, 0, 2, 4, 1)$, of value 5; the third and the second pass give none. The method stops with $5 < 7 = z(\text{ILP})$: a local optimum with respect to these windows, since the optimal solution $(0, 1, 1, 2, 0)$ differs from it in x_2, x_3, x_4 and x_5 , more variables than any window contains.

- Fix-and-optimize is a **local search** whose moves change all the integer variables of a window at once, explored exactly by the solver: the integer-programming counterpart of the neighbourhoods of local search. It stops in a local optimum with respect to all the windows.
- Relax-and-fix and fix-and-optimize are natural partners: the first builds a solution block by block, the second repairs the myopic decisions looking at several blocks together.

10.8 Heuristics inside the solver

- A modern MIP solver is not only a branch-and-bound algorithm: it runs, automatically, most of the heuristics of this chapter (the chapter «MIP solvers: what they do» shows real logs). In Gurobi:
 - *before the root LP*, heuristics that need no relaxation look for a first solution;
 - *at the root*, after the LP, rounding, several diving heuristics and the feasibility pump are run;
 - *during the branch-and-bound*, RINS and other sub-MIP heuristics are run at some nodes, with the LP solution of the node, and the new incumbents are reported in the log by a line starting with H (heuristic) instead of * (integer LP solution of a node).
- The effort spent in heuristics is controlled by parameters. The most important ones are:

parameter	default	meaning
Heuristics	0.05	fraction of the work spent in the heuristics (0 to 1)
MIPFocus	0	1: focus on finding good feasible solutions rather than on the bound
RINS	−1	frequency of RINS (every n nodes; −1 automatic, 0 off)
PumpPasses	−1	passes of the feasibility pump at the root (−1 automatic)
SubMIPNodes	500	node limit of the sub-MIPs of RINS and similar heuristics
NoRelHeurWork	0	work spent in the heuristic without relaxation before the root
ImproveStartGap	0	when the gap reaches this value, switch to improving the incumbent only

- A heuristic solution computed outside the solver — by any of the methods of this chapter — can be passed to it as a **MIP start** (the attribute **Start** of the variables): the solver starts from it as incumbent, uses it for pruning and for RINS, and never returns anything worse.
- The numbers depend on the instance, on the version of the solver and on the machine; a single instance proves nothing. The lesson is qualitative: when the bound is strong, the time to solve is the time to find a good solution, and the heuristics of the solver — or a good MIP start — decide it.

10.9 What MIP-based heuristics give

- A heuristic gives only one side of the gap, and without a dual bound nothing can be said about the quality of its solution. 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. The heuristic **knows how far it is from the optimum** while it runs: the optimality gap

$$\text{optimality gap} = \frac{\text{UB} - \text{LB}}{|\text{LB}|}$$

is computed at every moment, and when it is 0 the solution is *proved* optimal.

instance	solution	LB	UB = $\lfloor z(\text{LP}) \rfloor$	gap	certified by
car, rounding down	(3, 2)	13	15	15.4%	—
car, diving	(4, 1)	14	15	7.1%	proximity search from (3, 2)
P7, rounding down	(2, 0)	6	7	16.7%	—
P7, local branching	(2, 1)	7	7	0%	the bound
P42, RINS	(2, 2, 0)	10	10	0%	the bound
P46, relax-and-fix	(0, 2, 0)	12	19	58.3%	—
P46, fix-and-optimize	(0, 1, 1)	16	19	18.8%	proximity search
P72, feasibility pump	(0, 0, 1, 3, 1)	2	8	300%	—
P72, local branching	(0, 1, 1, 2, 0)	7	8	14.3%	proximity search

- On P7 and P42 the integer rounding of the LP value is already the optimal value: the difficulty is to find a solution that attains the bound, not to prove it. On the other instances the bound $\lfloor z(\text{LP}) \rfloor$ is not tight, and the certificate comes from proximity search (Proposition 10.12) or from the branch-and-bound of the solver.

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.

- The methods of this chapter, in summary:

method	needs LP	needs incumbent	fixing	idea
rounding	yes	no	—	round the LP solution
diving	yes	guided: yes	one at a time	fix a variable, re-solve the LP, repeat
feasibility pump	yes	no	—	alternate rounding and projection until they meet
RINS	yes	yes	hard	fix where LP and incumbent agree, solve the sub-MIP
local branching	no	yes	soft	sub-MIP with $\Delta(\mathbf{x}, \bar{\mathbf{x}}) \leq k$
proximity search	no	yes	soft	minimize $\Delta(\mathbf{x}, \bar{\mathbf{x}})$ subject to the cutoff
relax-and-fix	yes	no	by blocks	integer now, relaxed later, then fix
fix-and-optimize	no	yes	by windows	re-optimize a window, the rest fixed

10.10 Python implementation

- The codes use Python 3 and `gurobipy`. The variables of a model are accessed by their position j in `m.getVars()`, and the list I of the positions of the integer variables is the set $\mathcal{I}$; indices start from 0.
- The module `common.py` contains the environment of the solver and the operations used by all the methods: the LP relaxation, the rounding $[\cdot]$, the completion of a point, the distance Δ and the cutoff constraint.

```

1 import math
2 import gurobipy as gp
3 from gurobipy import GRB
4
5 EPS = 1e-6                                # integrality and
    comparison tolerance
6
7 ENV = gp.Env(empty=True)                  # one silent, single-
    thread (deterministic) environment
8 ENV.setParam("OutputFlag", 0)
9 ENV.setParam("Threads", 1)
10 ENV.start()
11
12
13 def int_index(m):
14     """positions j (in m.getVars()) of the integer and binary variables
        : the set I"""
15     return [j for j, v in enumerate(m.getVars()) if v.VType != GRB.
        CONTINUOUS]
16
17
18 def nearest(v):
19     """[v] = floor(v + 1/2): rounding to the nearest integer, 1/2
        rounded up"""
20     return math.floor(v + 0.5 + EPS)
21
22
23 def is_integral(v):
24     return abs(v - round(v)) <= EPS
25
26
27 def solve_lp(m):
28     """optimal solution of the LP relaxation of m (with its current
        bounds), None if infeasible"""
29     m.update()
30     r = m.relax()
31     r.optimize()
32     if r.Status != GRB.OPTIMAL:
33         return None
34     return [v.X for v in r.getVars()]

```

```

35
36
37 def obj_of(m, x):
38     """value of the objective function of m at the point x"""
39     e = m.getObjective()
40     return e.getConstant() + sum(e.getCoeff(k) * x[e.getVar(k).index]
41                                   for k in range(e.size()))
42
43 def round_point(x_star, I):
44     """x_tilde = [x*]: round the integer components, copy the
45       continuous ones"""
46     x_tilde = list(x_star)
47     for j in I:
48         x_tilde[j] = nearest(x_star[j])
49     return x_tilde
50
51 def evaluate(m, x_int):
52     """fix the integer variables to x_int and optimize the continuous
53       ones:
54       returns (value, x), or None if the point cannot be completed to
55       a feasible solution"""
56     f = m.copy()
57     V = f.getVars()
58     for j in int_index(f):
59         V[j].LB = V[j].UB = x_int[j]
60     f.optimize()
61     if f.Status != GRB.OPTIMAL:
62         return None
63     return f.ObjVal, [v.X for v in V]
64
65 def solve_mip(m):
66     """optimal value and solution of the MIP m, None if infeasible"""
67     m.optimize()
68     if m.SolCount == 0:
69         return None
70     return m.ObjVal, [v.X for v in m.getVars()]
71
72 def distance(m, x_tilde, I):
73     """the linear expression Delta(x, x_tilde) = sum_{j in I} |x_j -
74       x_tilde_j| in the variables of m:
75       x_j - l_j if x_tilde_j is at its lower bound, u_j - x_j if at
76       its upper bound,
77       otherwise an auxiliary variable d_j >= |x_j - x_tilde_j|"""
78     V = m.getVars()
79     expr = gp.LinExpr()
80     for j in I:
81         if x_tilde[j] == V[j].LB:
82             expr += V[j] - x_tilde[j]
83         elif x_tilde[j] == V[j].UB:

```

```

82         expr += x_tilde[j] - V[j]
83     else:
84         d = m.addVar(name=f"d[{j}]")
85         m.addConstr(d >= V[j] - x_tilde[j])
86         m.addConstr(d >= x_tilde[j] - V[j])
87         expr += d
88     return expr
89
90
91 def delta(x, x_tilde, I):
92     """value of Delta(x, x_tilde)"""
93     return sum(abs(x[j] - x_tilde[j]) for j in I)
94
95
96 def add_cutoff(m, z, theta):
97     """require a solution better than z by at least theta"""
98     obj = m.getObjective()
99     if m.ModelSense == GRB.MINIMIZE:
100         m.addConstr(obj <= z - theta, name="cutoff")
101     else:
102         m.addConstr(obj >= z + theta, name="cutoff")
103
104
105 def note(trace, *items):
106     """record a step of the algorithm (used to print the examples of
107         the notes)"""
108     if trace is not None:
109         trace.append(items)

```

- The model of the examples (models.py): $\max c x$ subject to $A x \leq b$, with integer variables in $\mathbb{Z}_{\geq 0}$ and optional upper bounds.

```

1 from common import *
2
3
4 def build_mip(c, A, b, ub=None, name="MIP"):
5     """max c x subject to A x <= b, 0 <= x <= ub, x integer"""
6     m = gp.Model(name, env=ENV)
7     n = len(c)
8     x = m.addVars(n, lb=0, ub=GRB.INFINITY if ub is None else ub, vtype
9                 =GRB.INTEGER, name="x")
10    m.setObjective(gp.quicksum(c[j] * x[j] for j in range(n)), GRB.
11                MAXIMIZE)
12    for i in range(len(b)):
13        m.addConstr(gp.quicksum(A[i][j] * x[j] for j in range(n)) <= b[
14            i], name=f"c{i}")
15    m.update()
16    return m

```

- Rounding and diving:

```

1 def rounding(m, trace=None):
2     I = int_index(m)
3     x_star = solve_lp(m)                                # optimal solution of
4     the LP relaxation
5     note(trace, "LP", x_star)
6     x_tilde = round_point(x_star, I)                    # x_tilde_j = [x*_j] for
7     every integer variable
8     note(trace, "rounded", x_tilde)
9     return evaluate(m, x_tilde)                          # (value, x) if feasible
10    , None otherwise

```

```

1 def diving(m, rule="fractional", x_inc=None, trace=None):
2     d = m.copy()                                         # the bounds of the copy
3     are changed, not those of m
4     V = d.getVars()
5     I = int_index(d)
6     while True:
7         x_star = solve_lp(d)                             # LP relaxation with the
8         current bounds
9         if x_star is None:
10            note(trace, "LP infeasible")
11            return None                                   # the dive fails
12        F = [j for j in I if not is_integral(x_star[j])]
13        note(trace, "LP", obj_of(d, x_star), x_star)
14        if not F:
15            return evaluate(m, round_point(x_star, I))
16        if rule == "fractional":                          # the least fractional
17            variable, to its nearest integer
18            j = min(F, key=lambda k: round(abs(x_star[k] - nearest(
19                x_star[k])), 9))
20            v = nearest(x_star[j])
21        else:                                              # guided: the variable
22            closest to the incumbent, to its value
23            j = min(F, key=lambda k: round(abs(x_star[k] - x_inc[k]),
24                9))
25            v = x_inc[j]
26        note(trace, "fix", V[j].VarName, v)
27        V[j].LB = V[j].UB = v                            # fix x_j := v

```

- The feasibility pump:

```

1 def projection(m, x_tilde, I):
2     ""x* = argmin { Delta(x, x_tilde) : x in P }, P the LP relaxation
3     of m""
4     p = m.relax()
5     p.setObjective(distance(p, x_tilde, I), GRB.MINIMIZE)
6     p.optimize()
7     return [v.X for v in p.getVars()][:m.NumVars]

```

```

1 def feasibility_pump(m, max_iter=100, seed=1, trace=None):
2     I = int_index(m)

```

```

10 x_star = solve_lp(m) # x* = optimal solution
    of the LP relaxation
11 note(trace, "LP", x_star)
12 if all(is_integral(x_star[j]) for j in I):
13     return evaluate(m, x_star)
14 x_tilde = round_point(x_star, I) # x_tilde = [x*]
15 note(trace, "round", x_tilde)
16 history = [x_tilde] # the points x_tilde
    visited so far
17 for it in range(1, max_iter + 1):
18     x_star = projection(m, x_tilde, I) # the point of P closest
    to x_tilde
19     dist = delta(x_star, x_tilde, I)
20     note(trace, "project", x_star, dist)
21     if dist <= EPS: # x* = x_tilde is
    integer and feasible
22         return evaluate(m, x_tilde)
23     x_new = round_point(x_star, I)
24     if all(x_new[j] == x_tilde[j] for j in I): # short
    cycle: [x*] = x_tilde
25         j = max(I, key=lambda k: (round(abs(x_star[k] - x_tilde[k])
    , 9), -k))
26         x_new[j] += 1 if x_star[j] > x_tilde[j] else -1 # flip the
    most uncertain component
27         note(trace, "flip", x_new, j)
28     else:
29         note(trace, "round", x_new)
30         if any(all(x_new[j] == y[j] for j in I) for y in history[-3:]):
    # long cycle
31             for j in I: # restart: random flips
    (binary variables)
32                 seed = (16807 * seed) % 2147483647
33                 rho = seed / 2147483647 - 0.3 # rho_j uniform in
    [-0.3, 0.7]
34                 if abs(x_star[j] - x_new[j]) + max(rho, 0) > 0.5:
35                     x_new[j] = 1 - x_new[j]
36                 note(trace, "restart", x_new)
37             x_tilde = x_new
38             history.append(x_tilde)
39 return None # no feasible solution
    found

```

- RINS, local branching and proximity search:

```

1 def rins(m, x_tilde, theta=1, trace=None):
2     I = int_index(m)
3     z_tilde = obj_of(m, x_tilde) # value of the incumbent
4     x_star = solve_lp(m) # optimal solution of
    the LP relaxation
5     s = m.copy()
6     V = s.getVars()
7     fixed = []

```

```

8   for j in I:
9       if abs(x_star[j] - x_tilde[j]) <= EPS:           # LP and incumbent
               agree on x_j
10          V[j].LB = V[j].UB = x_tilde[j]
11          fixed.append(j)
12      note(trace, "LP", x_star, "fixed", [V[j].VarName for j in fixed])
13      add_cutoff(s, z_tilde, theta)                     # look only for better
               solutions
14      res = solve_mip(s)                                # the sub-MIP on the
               free variables
15      if res is None:
16          note(trace, "sub-MIP infeasible")
17          return z_tilde, x_tilde
18      note(trace, "sub-MIP", res[0])
19      return res

```

```

1  def local_branching(m, x_tilde, k, k_max, k_step, theta=1, trace=None):
2      I = int_index(m)
3      w = m.copy()                                     # the master MIP, which
               receives the reverse constraints
4      z_tilde = obj_of(m, x_tilde)
5      while k <= k_max:
6          s = w.copy()
7          s.addConstr(distance(s, x_tilde, I) <= k, name="local_branch")
               # Delta(x, x_tilde) <= k
8          add_cutoff(s, z_tilde, theta)
9          res = solve_mip(s)
10         w.addConstr(distance(w, x_tilde, I) >= k + 1)
               # never explore it again
11         if res is not None:                             # better solution: re-
               center the neighbourhood
12             note(trace, "improved", k, res[0], res[1])
13             z_tilde, x_tilde = res
14         else:                                           # no better solution
               within distance k: enlarge it
15             note(trace, "no improvement", k)
16             k += k_step
17     return z_tilde, x_tilde

```

```

1  def proximity_search(m, x_tilde, theta=1, trace=None):
2      I = int_index(m)
3      z_tilde = obj_of(m, x_tilde)
4      while True:
5          p = m.copy()
6          add_cutoff(p, z_tilde, theta)                 # better than the
               incumbent by at least theta
7          p.setObjective(distance(p, x_tilde, I), GRB.MINIMIZE) # as
               close as possible to it
8          res = solve_mip(p)
9          if res is None:                                 # no solution better
               than z~ - theta exists
10             note(trace, "infeasible")
11             return z_tilde, x_tilde

```

```

12     z_new, x_new = evaluate(m, res[1])          # value of the new
13         solution, continuous part re-optimized
14     note(trace, "found", res[0], z_new, x_new)
15     z_tilde, x_tilde = z_new, x_new

```

- Relax-and-fix and fix-and-optimize:

```

1 def relax_and_fix(m, blocks, trace=None):
2     r = m.copy()
3     V = {v.VarName: v for v in r.getVars()}
4     vtype = {name: V[name].VType for B in blocks for name in B}
5     for B in blocks:                                # at first, all the
6         blocks are relaxed
7         for name in B:
8             V[name].VType = GRB.CONTINUOUS
9     for B in blocks:                                # the current block is
10        for name in B:                                integer again
11            V[name].VType = vtype[name]
12        res = solve_mip(r)
13        if res is None:
14            note(trace, "infeasible")
15            return None
16        note(trace, "block", B, res[0], {n: V[n].X for BB in blocks for
17            n in BB})
18        for name in B:                                # fix the current block
19            to its value
20            V[name].LB = V[name].UB = round(V[name].X)
21    return res

```

```

1 def fix_and_optimize(m, x_tilde, windows, theta=1, trace=None):
2     I = int_index(m)
3     names = [v.VarName for v in m.getVars()]
4     z_tilde = obj_of(m, x_tilde)
5     improved = True
6     while improved:                                # one pass over all the
7         windows
8         improved = False
9         for W in windows:
10            s = m.copy()
11            V = s.getVars()
12            for j in I:
13                if names[j] not in W:                # outside the window:
14                    fixed to the incumbent
15                    V[j].LB = V[j].UB = x_tilde[j]
16            add_cutoff(s, z_tilde, theta)
17            res = solve_mip(s)
18            if res is not None:
19                z_tilde, x_tilde = res
20                improved = True
21            note(trace, "window", W, z_tilde)
22    return z_tilde, x_tilde

```

- All the functions of this section are in the folder `MIP_HEUR_CODE`, together with `instances.py`, which contains 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.

10.11 Typical mistakes

Warning

- **Rounding a problem with equality constraints.** Independent roundings of the variables of an equality constraint almost never satisfy it: rounding fails, and a dive is needed.
- **Believing that a dive can undo a decision.** A dive is one path of the tree, without backtracking: a wrong fixing is never revised, and a fixing that makes the LP infeasible ends the dive.
- **Using the feasibility pump to get a good solution.** After the first LP the pump ignores the objective: on *P72* it returns a solution of value 2, against the optimal value 7.
- **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 (*P7*).
- **Forgetting the cutoff constraint.** Without $f(\mathbf{x}) \leq f(\bar{\mathbf{x}}) - \theta$ the sub-MIP returns the incumbent solution again; with it, an infeasible sub-MIP proves a local optimum.
- **Claiming optimality without the conditions.** Proposition 10.12 needs $\theta = 1$ and integer objective values on all the feasible solutions.
- **Ignoring the myopia of relax-and-fix.** The relaxed blocks are optimistic: a decision that looks cheap can force a bad one later (*P46*).
- **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.

10.12 In summary

In summary

- MIP-based heuristics use the LP relaxation to decide where to look and a MIP solver to explore a large neighbourhood exactly.
- 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 block by block, 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.

Dashboard of the car example. The lower and upper bounds on $z(\text{ILP})$ found so far in this module:

Chapter	Lower bound	Upper bound	From
Integer linear programming	12, 14	–	heuristic solutions $(4, 0)$ and $(4, 1)$
LP relaxation and dual bounds	–	15	$\lfloor z(\text{LP}) \rfloor$, with $z(\text{LP}) = \frac{31}{2}$
Cutting plane algorithm	14	14	5 Gomory fractional cuts
LP-based branch-and-bound	14	14	6 LP problems solved
This chapter, rounding down	13	15	$(3, 2)$
This chapter, diving	14	15	$(4, 1)$, certified optimal by proximity search

Bibliography

- [1] T. Achterberg, T. Berthold, Improving the feasibility pump, *Discrete Optimization* 4 (2007) 77–86.
- [2] L. Bertacco, M. Fischetti, A. Lodi, A feasibility pump heuristic for general mixed-integer problems, *Discrete Optimization* 4 (2007) 63–76.
- [3] E. Danna, E. Rothberg, C. Le Pape, Exploring relaxation induced neighborhoods to improve MIP solutions, *Mathematical Programming* 102 (2005) 71–90.
- [4] M. Fischetti, F. Glover, A. Lodi, The feasibility pump, *Mathematical Programming* 104 (2005) 91–104.
- [5] M. Fischetti, A. Lodi, Local branching, *Mathematical Programming* 98 (2003) 23–47.
- [6] M. Fischetti, M. Monaci, Proximity search for 0-1 mixed-integer convex programming, *Journal of Heuristics* 20 (2014) 709–731.