Politecnico di Torino

Master's Degree in Management Engineering

Vacuum Furnace Load Optimisation

Master's Thesis

Supervisor:
Prof. Fabio Salassa

Candidate:
Alberto Ricca

Academic Year 2025/2026

Abstract

This thesis formalises the vacuum-furnace load-planning problem arising at Tesio Cooling Systems S.p.A. as a level-based two-stage two-dimensional bin packing problem, a structure shared with the classical two-dimensional cutting-stock problem. A binary mixed-integer linear programme is developed and solved exactly per compatibility family by decomposition, and is compared against a fast two-phase constructive heuristic representative of current planning practice. On the real-world order book the exact model removes 12 of the heuristic's 214 furnace loads, a saving traceable to a specific cross-code consolidation that the heuristic's greedy first phase forecloses.

A controlled synthetic benchmark then characterises when exact optimisation is worth its cost: families whose codes share a common length are solved optimally by the heuristic, whereas heterogeneous families yield a 15–20% reduction in load count under the MILP. This separation proves invariant to the geometric regime and to persist on uniformly random instances. A final experiment establishes a negative result of methodological interest: the magnitude of the advantage cannot be predicted by any a priori geometric index of length complementarity ($R^2 < 0.04$ across every variant tested), because it arises from the interaction between structure and demand rather than from length geometry alone. The outcome is a two-tier selection rule that screens families in $O(n)$ by structural homogeneity and reserves the exact model for the regime where it delivers measurable value.

Contents

List of Figures

List of Tables

Introduction

This thesis addresses the furnace loading optimisation problem arising during a curricular internship at Tesio Cooling Systems S.p.A., an Italian manufacturer of aluminium radiators for railway and agricultural applications. These radiators are produced by *brazing*: a joining process in which the assembled aluminium components are heated until a filler alloy melts and bonds them permanently, without melting the components themselves. The operation is carried out inside a vacuum furnace, where the absence of air prevents oxidation of the aluminium and ensures a uniform temperature distribution across the assembly.

Brazing is the primary capacity-constrained operation of the production process. It is the most time- and energy-intensive phase, and it processes radiators in batches: each furnace cycle, or *load*, brazes a group of radiators simultaneously and incurs a fixed energy and operating cost largely independent of how many radiators it contains. The furnace chamber is therefore the binding physical resource, and the number of loads required to satisfy demand is the quantity that production planning seeks to minimise.

The radiators are arranged inside the chamber along its length, so that the way they are combined within each load directly determines how many loads are needed. From September 2026 a new furnace with a substantially longer chamber will enter service at Tesio. With the current short chamber only a few radiators fit per load and the arrangement is decided by operator experience; the increased length of the new chamber multiplies the number of feasible combinations to the point where manual planning is no longer adequate. This furnace transition motivates the optimisation approach developed in this thesis.

The present work pursues three objectives. The first is to formalise the furnace loading problem as a mixed-integer linear programme (MILP), an optimisation framework in which an objective is minimised (or maximised) subject to linear constraints over continuous and integer decision variables. The available theory of two-dimensional cutting and packing models provides the theoretical foundation for this objective. The second is the development of a fast constructive heuristic, based on the same

geometric assumptions and intended to represent the type of approach that operators would most likely adopt in practice. The third objective is to quantify the reduction in required loads achievable by exact optimisation over greedy construction. While the Tesio order book provides an observed industrial case study, a controlled set of synthetic instances is used to investigate the behaviour of the two approaches under a wider range of operating conditions.

The principal contribution is an *applicability map* establishing when the MILP recovers significant value and when a near-instant heuristic already suffices. A secondary contribution is negative but instructive: the applicability of the exact model is shown *not* to be predictable from an a priori geometric index of length complementarity. The decisive factor is the structural homogeneity of the family, which separates the two regimes cleanly in $O(n)$, whereas the magnitude of the advantage within the heterogeneous regime resists reduction to any cheap precomputable scalar.

The thesis is organised as follows. Chapter 1 defines the problem and locates it within the theory of two-dimensional cutting and packing. Chapters 2 and 3 develop and implement the MILP formulation; Chapter 4 introduces the constructive heuristic. Chapter 5 reports the results on a toy instance, on the real-world order book, and on the synthetic benchmark. Chapter 6 draws the conclusions, and Chapter 7 sets out the directions for future work.

Chapter 1

Problem Definition

1.1 Industrial Context

The unit analysed in the present work is the radiator brazed *core*: the central section of the heat exchanger. It is formed by an alternating stack of fins and turbulators, both responsibles of the thermal performance of the radiator, separated by thin aluminium plates. The core is geometrically a parallelepiped, defined by three dimensions: its two in-plane dimensions and its height.

The loading model represents each unit by its two in-plane dimensions only. This is a direct consequence of the loading practice: radiators are placed side by side on a single horizontal tier inside the chamber, so the height does not participate in the arrangement and introduces no further packing freedom. The doubled chamber height of the new furnace would in principle allow a second stacked tier; this possibility is deliberately excluded from the initial formulation and is treated as a future extension in Chapter 7. Within a single tier, each unit is therefore modelled as a rectangular item characterised by its two in-plane dimensions.

The current furnace comprises two independent chambers, each 440 $\times$ 1320 mm (width $\times$ length). The new furnace retains both the 440 mm width and the two-chamber configuration, but extends the usable chamber length to 2700 mm. An illustrative comparison between the two chamber geometries is shown in Figure 1.2. From September 2026, the new furnace enters in service and will absorb most of the production. The current unit is not decommissioned: it is retained as a complementary resource for absorbing delays and for processing very small residual batches, for which starting a full cycle in the much larger new chamber would be inefficient. The loading model developed in this thesis targets the new furnace, where the bulk of the brazing workload is planned.

Figure 1.1 illustrates the production flow of a radiator, showing the context in which the new furnace will operate.

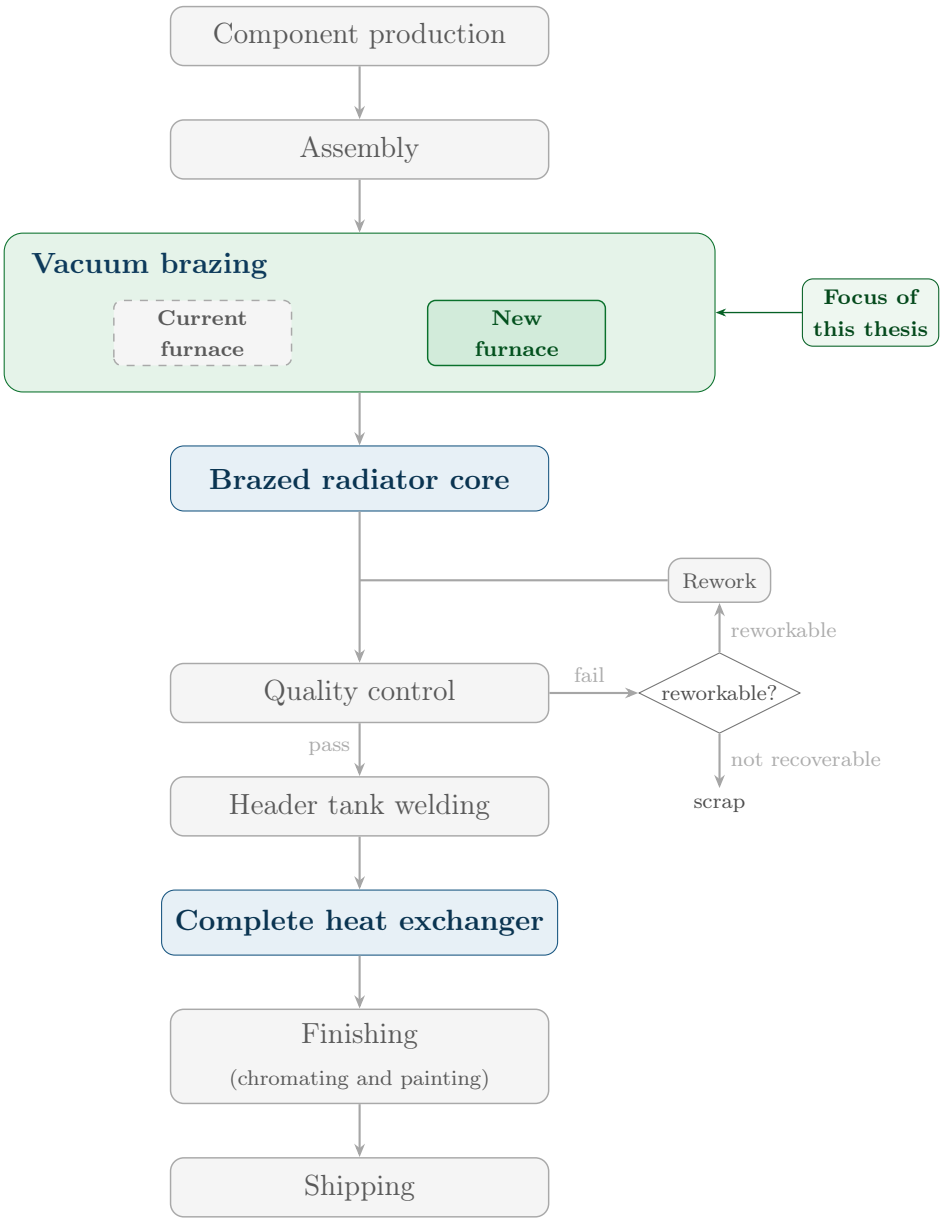


Figure 1.1: **Manufacturing process flow.**The two furnaces are alternatives.

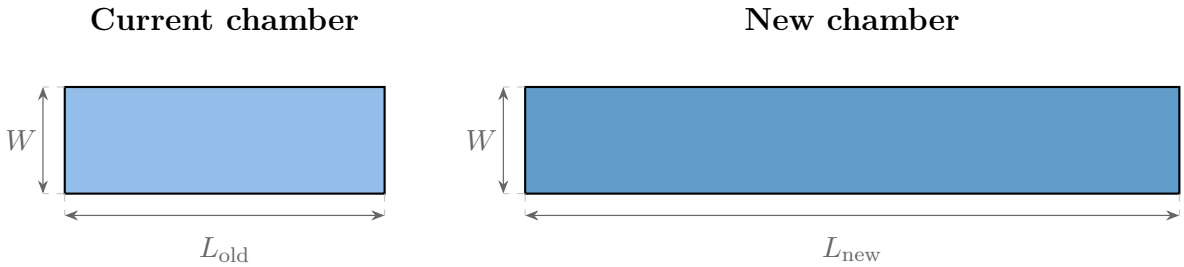


Figure 1.2: **Current and incoming furnace chambers.** The usable chamber increases from 1320 mm to 2700 mm while the usable width remains unchanged.

1.2 Compatibility Families

On the current furnace the loading problem barely existed. Its short chamber held only one or two radiators per chamber (there were only two exceptional codes with three and four loadable units per chamber). With so little room there was nothing to combine, and each load was filled with units of a single code. The new furnace changes this. Its far longer chamber accommodates many more radiators per load, and filling it efficiently with single-code loads alone is no longer possible. Leftover space must be recovered by placing different codes in the same load. This need to mix codes raises the question of which codes may be combined, and it is the origin of the compatibility families described in this section.

Mixing codes is not unrestricted. A furnace load subjects every unit it contains to a single temperature–time profile, the *thermal cycle*, which must melt the filler alloy without damaging the base material of the components. The furnace supports four such cycles, numbered 1, 2, 3 and 10, and the cycle assigned to a given radiator is governed by its core width: wider cores need to be driven harder and held longer before the filler alloy melts uniformly through the mass. Cycle 3 serves the narrowest cores, up to roughly 65 mm; Cycle 2 cores up to about 95 mm; Cycle 1 cores up to about 125 mm; and Cycle 10 the widest cores, spanning a much broader band from about 125 mm up to 300 mm. The cycle numbering follows Tesio's internal convention and does not run in order of core width, but each radiator is mapped to exactly one cycle on the basis of its core size.

However,the cycle number alone does not fully determine which radiators may be brazed together. Each cycle is a sequence of timed phases: a staged ramp-up, intermediate holds, the approach to the brazing temperature, and a controlled cooling. The rate at which a given radiator follows those phases is not identical across all units nominally assigned to the same cycle. Both the core width and the fin insert type affect how quickly a radiator absorbs heat and reaches the target temperature, so two units sharing a cycle in principle may in practice reach brazing temperature a few minutes apart. On a profile lasting roughly two hours, a gap of even five minutes is enough to matter. Consider two radiators both assigned to Cycle 10, one with a core width of 150 mm and another of 300 mm: the narrower unit reaches brazing temperature first. Holding the cycle long enough to braze the wider unit completely then risks melting not only the filler alloy but a thin layer of the base material of the narrower one, whereas shortening the cycle leaves the wider unit insufficiently brazed. Either outcome results in a defective part.

For Cycles 1 to 3 this effect is contained. Each of these cycles spans a narrow band of core widths, so two radiators sharing the same cycle and the same fin insert are

necessarily close in size and respond to the profile similarly enough to be brazed together without further qualification. Cycle 10 behaves differently: its core-width band is far wider, from about 125 mm to 300 mm, and across that range the thermal response of different radiators diverges enough to require an additional split by core-width class. Sharing cycle and fin insert is therefore not sufficient under Cycle 10, and only radiators of comparable core width may be grouped together.

Compatibility is consequently determined by three factors: the thermal cycle, the fin insert and, within Cycle 10, the thickness class of the core. Two radiators may share a load only if they agree on all applicable factors; those that do are grouped into *compatibility families*.

The thirteen families recognised at Tesio are listed in Table 1.1.

Family	Thermal cycle	Fin insert	Core width [mm]
F1	Cycle 1	Fin–A	95–125
F2	Cycle 1	Fin–B	95–125
F3	Cycle 1	Fin–C	95–125
F4	Cycle 2	Fin–D	65–95
F5	Cycle 2	Fin–B	65–95
F6	Cycle 2	Fin–C	65–95
F7	Cycle 3	Fin–B	≤ 65
F8	Cycle 10	Fin–D	125–210
F9	Cycle 10	Fin–E	170–210
F10	Cycle 10	Fin–A	130–220
F11	Cycle 10	Fin–A	270–330
F12	Cycle 10	Fin–B	130–220
F13	Cycle 10	Fin–C	270–330

Table 1.1: **Compatibility families.** Classification adopted throughout the thesis.

A furnace load may contain radiators of one family only: all units sharing a single load must belong to the same family. This is the first and most important property that any feasible load must satisfy, and it partitions the planning problem into independent sub-problems, one per family, a structure later exploited algorithmically in Section 3.1. Throughout this thesis a *load* denotes one filling of the single modelled chamber of length L and width W; the two-chamber configuration of the physical furnace is treated as a separate extension in Chapter 7, where the two chambers may run different families within the same thermal cycle. Family F4 is absent from the current order book and does not appear in the planning instances; the remaining twelve are

those effectively solved in Chapter 3.

1.3 Problem Modelling

What remains to be fixed is how a compatible set is physically arranged inside the chamber and what features of that arrangement the model represents. This section draws the boundary explicitly, presenting each modelling choice and structural restriction with its physical motivation. A summary table at the end collects all choices for quick reference.

The starting point is that radiators are not loaded bare. During brazing each core is enclosed in a supporting frame whose geometry varies from code to code. In the actual conformation of the frames in service at Tesio, the frame extends beyond the core laterally only, while it stays flush with the core along the length. The supporting feet of these frames, positioned perpendicularly to the frame plane, can partially interlock when two adjacent frames are placed side by side, so that the effective space occupied by two radiators may fall below the sum of their frame footprints. This interlocking is routinely exploited by experienced operators and allows transversal loading capacity that a purely rectangular model cannot predict. This mechanism is governed by code-specific foot geometry that the present work does not measure, and representing it would require pairwise compatibility rules between frames that lie outside the scope of this work. It is excluded here and treated as a future modelling extension in Chapter 7. Figure 1.3 illustrates the interlocking mechanism.

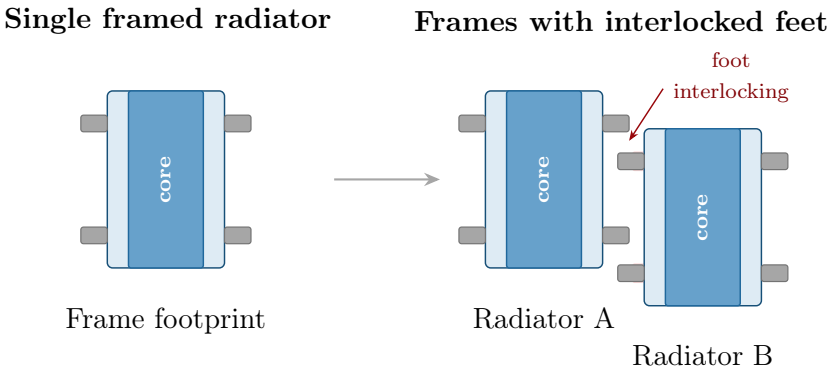


Figure 1.3: **Frame-foot interlocking concept.** This effect is excluded from the model and treated in Chapter 7.

The framed units are not placed into the chamber directly. They are arranged on a *loading guide*: a guided carrier inserted into the chamber for each cycle and withdrawn once brazing is complete. The guide is slightly smaller than the chamber and sits inside it with a margin of empty space on every side. It defines the region actually

available for packing, a rectangle of usable length L and usable width W. The chamber dimensions never bind the arrangement. Every dimension in the model, including the furnace figures quoted in Section 1.1, refers to the loading guide and not to the chamber. This margin also shapes how units sit across the width. A radiator at the lateral edge of the guide faces empty space rather than a wall, and its frame may reach the guide boundary with no clearance beside it. Clearance is needed only between two adjacent radiators: the lateral gap σ of Section 2.3.

Figure 1.4 builds the map from the bare radiator core to its position inside the furnace chamber.

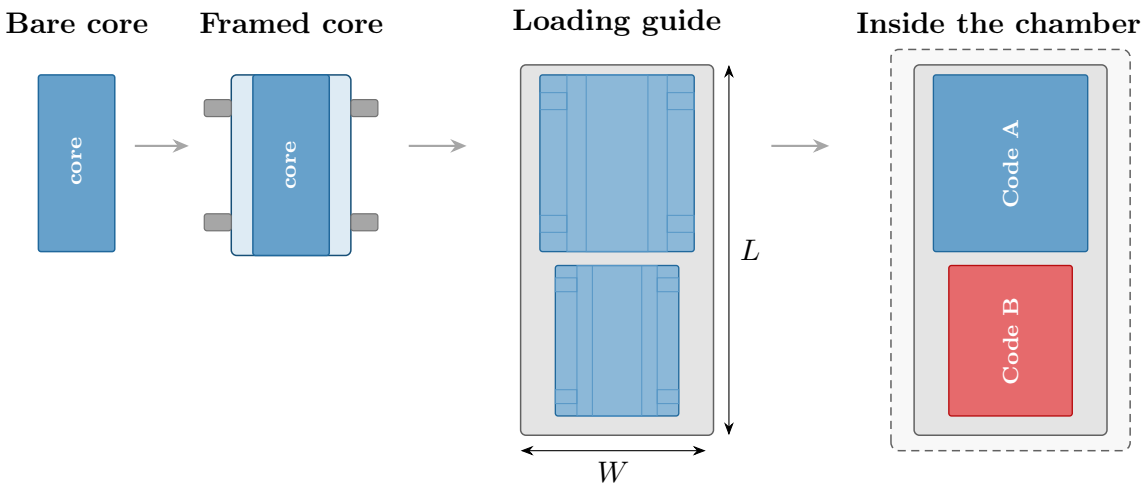


Figure 1.4: **From a bare core to a loaded guide inside the chamber.**

The model further considers a single chamber and a single loading tier: the second physical chamber and the second stacked tier enabled by the new furnace's doubled height are both deferred to Chapter 7. The formulation developed here therefore concerns one chamber filled on a single level. Under these restrictions, the radiator is represented as a rectangle whose footprint is the bare core enlarged by constant frame clearances, fixed at the values measured on the frames currently in service at Tesio. The geometric parameters formalising this representation are defined in Section 2.3.

All radiators are positioned parallel to the furnace axes. Rotations, diagonal and fishbone arrangements, and configurations exploiting foot interlocking are excluded from the present formulation, as each would require a geometric complexity that lies outside the scope of a first analysis. Each loading level contains units of a single radiator code, replicated side by side across the chamber width. The chamber is partitioned longitudinally into a sequence of such levels, each filled transversely. This structure, illustrated in Figure 1.5, places the problem within the class of cutting-and-packing models identified in Section 1.4.

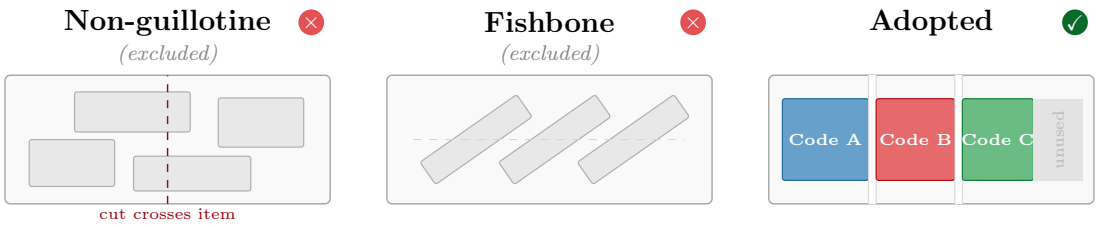


Figure 1.5: **Physical loading configurations.**

Production demand d_i for each code is taken as the aggregate monthly demand over the order book from May to September of the current year and is treated as known and deterministic. The family-compatibility constraint introduced in Section 1.2 is a physical process requirement rather than a modelling choice: it constrains which codes may share a load regardless of any geometric consideration.

Table 1.2 collects the choices above for quick reference.

Aspect	**Choice**	**Reference**
Item geometry	Rectangular footprint	
Frame clearances	Fixed across codes	Section 2.3
Item orientation	Parallel to furnace axes	
Loading structure	Two-stage level-based	Section 1.4
Foot interlocking	Excluded	Chapter 7
Loading dimensions	Loading guide, not chamber	Section 2.3
Chamber count	Single chamber	Chapter 7
Loading tiers	Single tier	Chapter 7
Production demand	Known and deterministic	
Family compatibility	Physical requirement	Section 1.2

Table 1.2: **Modelling choices.**

1.4 2D Cutting-Stock Analogy

The modelling choices of Section 1.3 translate the furnace loading problem into the language of two-dimensional cutting and packing (2D C&P), a family of problems extensively studied in the operations research literature [4, 3, 10, 6]. This section works through the main classes in that taxonomy, discards those that do not fit the furnace context, and arrives at the single formulation used throughout the rest of the thesis.

Strip packing. In strip packing one dimension of the container is left free and the objective is to minimise the total length used. This does not match the furnace context, where both chamber dimensions are fixed in advance and the quantity to minimise is the number of loads rather than a length. Strip packing is excluded.

Non-guillotine bin packing. Two-dimensional bin packing treats each bin as a fixed rectangle and seeks the minimum number of bins needed to pack all items. Non-guillotine formulations within this class allow arbitrary rectangular placement, without requiring that items be separable by straight cuts running across the full width or length. They are more flexible but require explicit two-dimensional coordinate variables for every item and are considerably harder to solve [10]. Since the loading process is orthogonal and level-based, this additional flexibility brings no benefit and is not adopted.

One-stage guillotine bin packing. In one-stage guillotine models the bin is divided by a single sequence of parallel cuts running in one direction [4]. This produces strips of uniform orientation but cannot represent a layout in which each strip is then further subdivided in the orthogonal direction, precisely the structure that the furnace loading process exhibits. One-stage guillotine models are therefore also excluded.

Figure 1.6 summarises these exclusions and the path to the adopted class.

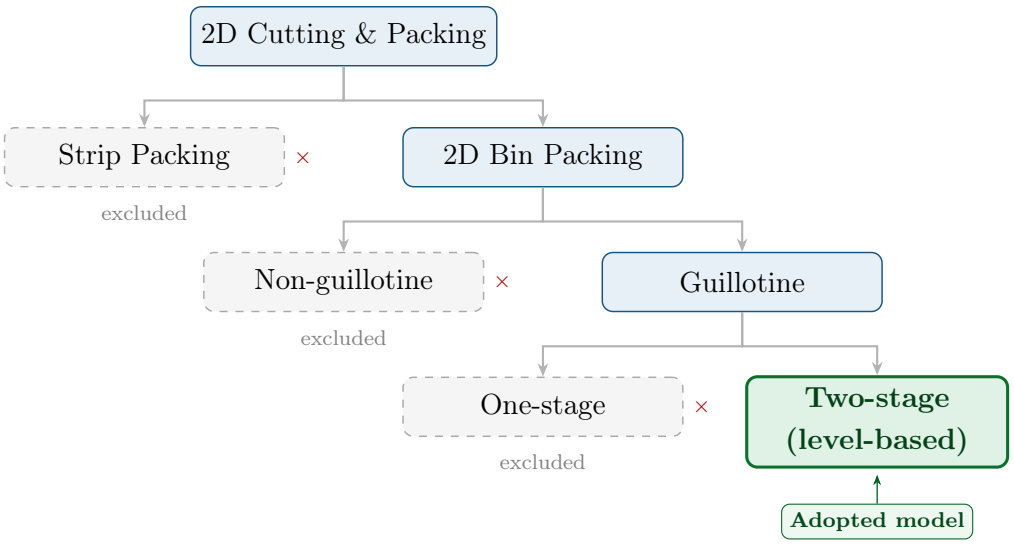


Figure 1.6: **Taxonomy of two-dimensional cutting and packing models.**

Two-stage guillotine bin packing. Two-stage models produce strips with a first set of cuts and then subdivide each strip along the orthogonal direction with a second [4]. This structure maps directly onto the furnace loading process: the first stage corresponds to the longitudinal partitioning of the chamber into loading levels, and the second stage to the side-by-side placement of radiators within each level.

Figure 1.7 illustrates the two stages on a concrete chamber configuration.

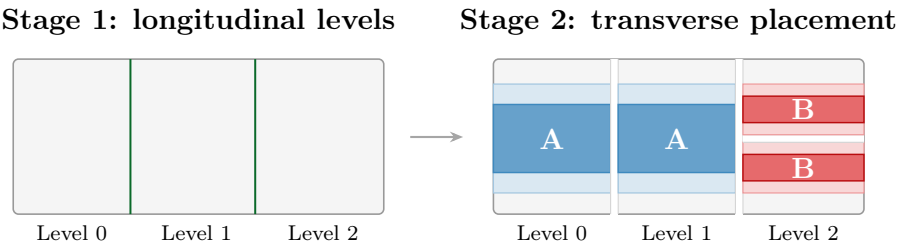


Figure 1.7: **Two-stage guillotine structure**

The adopted model is therefore a *level-based two-stage two-dimensional bin packing problem*: each furnace load is a bin; the first stage partitions the bin into longitudinal levels; the second stage fills each level with copies of a single radiator code placed side by side. The same structure is shared by the classical Two-Dimensional Two-Stage Cutting Stock Problem (2SCSP), and the parallel between the two problems is close enough to be made explicit.

Table 1.3 maps each element of the classical problem onto its furnace counterpart.

2D Cutting Stock (classical)	Furnace loading
Raw-material sheet	Furnace chamber
Rectangular item to cut	Radiator core plus frame clearances
Number of sheets used	Number of furnace loads
Horizontal strip (1st stage)	Longitudinal loading level
Transverse cut (2nd stage)	Side-by-side placement in width
Minimise number of sheets	Minimise number of furnace loads

Table 1.3: **Correspondence between the furnace loading problem and the 2SCSP.**

The level-based structure has one acknowledged limitation. Since radiators are cut to customer specification, the sum of level lengths and inter-level gaps will not equal L exactly, leaving a strip of unusable space at the end of every chamber. A large residual would signal that the excluded arrangements—foot interlocking and fishbone placement chief among them—carry density the model has discarded. The saturations reported in Chapter 3 show this is not the case: the chamber is filled almost completely on the great majority of loads, confirming that the excluded arrangements would refine the result at the margin rather than transform it.

Chapter 2

MILP Formulation

Symbol	Kind	Description
I	Set	Set of radiator codes.
$\mathcal{F}$	Set	Set of compatibility families.
I_f	Set	Radiator codes belonging to family f.
J_f	Set	Candidate loading levels for family f.
K_f	Set	Candidate furnace loads for family f.
i	Index	Radiator-code index.
f	Index	Compatibility-family index.
j	Index	Loading-level index.
k	Index	Furnace-load index.
L	Parameter	Usable chamber length.
W	Parameter	Usable chamber width.
τ	Parameter	Minimum longitudinal gap between levels.
δ	Parameter	Lateral frame overhang per side.
σ	Parameter	Minimum lateral clearance between adjacent items.
HEC_i	Parameter	Core length of radiator i [mm].
SP_i	Parameter	Core width of radiator i [mm].
d_i	Parameter	Production demand of code i.
f_i	Parameter	Compatibility family of code i.
η_i	Parameter	Maximum units of code i placeable side by side.
m_i	Parameter	Maximum levels of code i in a single-code load.
cap_i	Parameter	Full single-code load capacity: $\eta_i m_i$.
$\bar{j}_f$	Parameter	Maximum candidate levels per load for family f.
$\bar{j}_f^{\min}$	Parameter	Minimum level bound for family f.
n_k	Derived quantity	Number of active levels in load k.
ρ_k	Derived KPI	Longitudinal saturation of load k.
c_k	Decision variable	1 if load k is used, 0 otherwise.
z_{ijk}	Decision variable	1 if level j of load k is assigned to code i, 0 otherwise.

Table 2.1: **Model notation.**

2.1 Loading geometry

Chapter 1 defined the furnace loading problem and placed it within the class of level-based two-stage two-dimensional bin packing problems. This chapter translates that structure into a binary mixed-integer linear programme (MILP). The model operates on a single chamber of usable length L and usable width W, fixed by the specification of the new furnace (Section 2.3).

Recall from Chapter 1 that a furnace *load* consists of *loading levels* arranged along the chamber length, each containing units of a single radiator code placed side by side across the width. Consecutive levels are separated by a minimum longitudinal gap τ, so a load with n_k active levels occupies n_k core lengths and exactly $n_k - 1$ inter-level gaps.

The maximum number of units of code i that fit side by side within one level is:

$$\eta_i = \max\left(1,\ \left\lfloor \frac{W + \sigma}{\mathrm{SP}_i + 2\delta + \sigma} \right\rfloor\right)$$

where SP_i is the core width, δ the lateral frame overhang per side, and σ the minimum lateral clearance between adjacent items. The quantity η_i is computed once from the geometric parameters before the optimisation begins and enters the model as a fixed parameter.

Figure 2.1 illustrates the side-by-side arrangement that defines η_i.

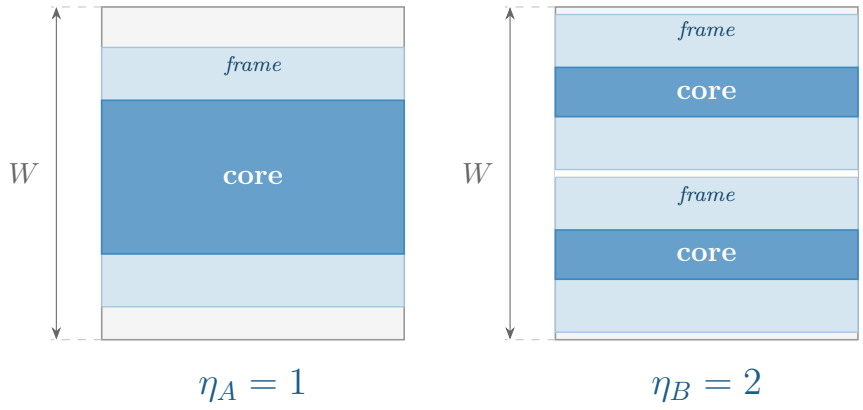


Figure 2.1: **Lateral capacity of a loading level.** Cross-sectional interpretation of parameter η_i.

The chamber length also limits the number of levels a single load can contain. For family f, the largest feasible number is:

$$\bar{j}_f = \left\lfloor \frac{L + \tau}{\min_{i \in I_f} \mathrm{HEC}_i + \tau} \right\rfloor,$$

obtained when every level is occupied by the shortest code in the family. Like η_i, the bound $\bar{j}_f$ is a fixed input determined by the geometry alone, and it sets the number of candidate levels per load (Section 2.2).

2.2 Sets and Indices

The following sets and indices are used throughout the formulation. All sets are instantiated separately for each compatibility family f, consistently with the decomposition strategy described in Section 3.1.

- I: set of radiator codes, indexed by i.
- $\mathcal{F}$: set of compatibility families, indexed by f.
- $I_f \subseteq I$: subset of codes belonging to family f.
- $J_f = \{0, 1, \ldots, \bar{j}_f - 1\}$: candidate loading levels per load for family f, with the bound $\bar{j}_f$ defined in Section 2.1.
- K_f: set of candidate furnace loads for family f; the bound $|K_f|$ is derived in Section 3.1.

2.3 Parameters

- $L = 2700$ mm: usable chamber length.
- $W = 440$ mm: usable chamber width.
- HEC_i: core length of radiator i [mm], frame excluded.
- SP_i: core width of radiator i [mm], frame excluded.
- $\delta = 70$ mm: lateral frame overhang per side, uniform across all codes; measured from the frame geometries in use at Tesio.
- $\tau = 40$ mm: minimum longitudinal gap between consecutive levels; measured from the frame geometries in use at Tesio.
- $\sigma = 10$ mm: lateral clearance between adjacent items; measured from the frame geometries in use at Tesio.
- d_i: production demand for code i [units].
- η_i: maximum units of code i per level, as defined in Section 2.1.

Figure 2.2 collects the geometric parameters on a sample load.

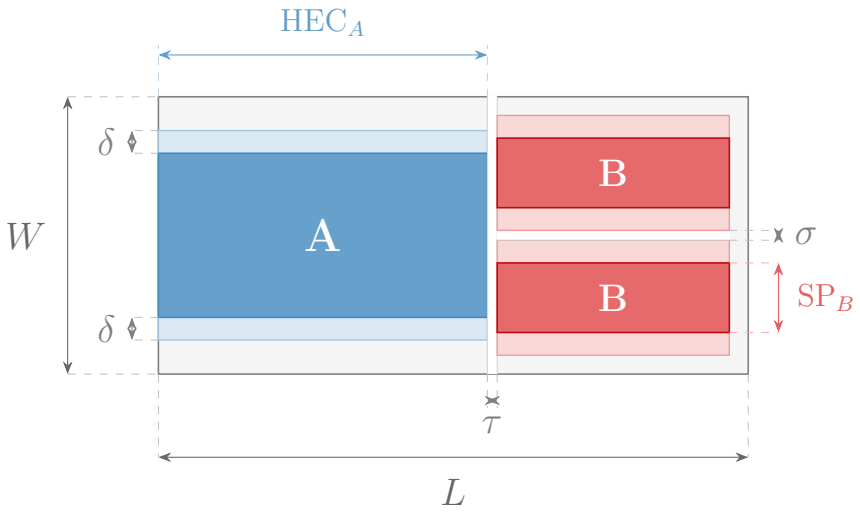


Figure 2.2: **Geometric parameters of a furnace load.**

2.4 Decision Variables

Two binary variables are sufficient to represent the assignment of radiators to loads and levels.

- $c_k \in \{0, 1\}$: equals 1 if furnace load k is used, 0 otherwise.
- $z_{ijk} \in \{0, 1\}$: equals 1 if level j of load k is dedicated to code i, 0 otherwise.

No family assignment variable is required: since the model is solved separately for each compatibility family (Section 3.1), every sub-problem already operates over a single family and no inter-family assignment needs to be represented.

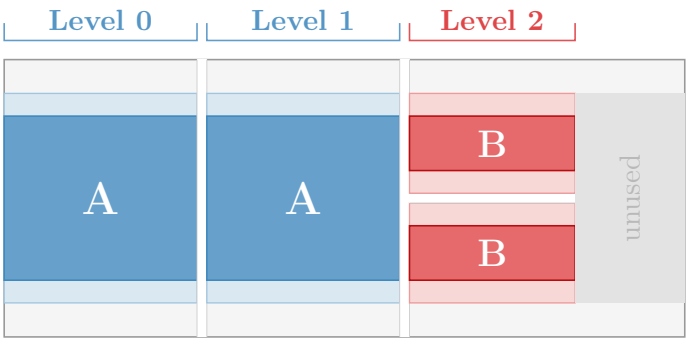
2.5 Objective Function

The objective is to minimise the total number of furnace loads used:

$$\min \sum_{k \in K_f} c_k$$

Minimising load count directly minimises energy consumption and operating cost, since each load incurs a fixed cost largely independent of how many radiators it contains (as established in Chapter 1).

Figure 2.3 shows a single active load with three loading levels and the corresponding binary variable values.



$c_0 = 1$

$z_{A,0,0} = 1, \quad z_{A,1,0} = 1, \quad z_{B,2,0} = 1$
all other $z_{ijk} = 0$

Produced: code A ($2 \cdot \eta_A = 2$), code B ($\eta_B = 2$)

Figure 2.3: **Worked example of the MILP decision variables.**

2.6 Constraints

The model enforces six constraint groups. The first satisfies demand; the second enforces the physical chamber capacity; the third and fourth govern the relationship between level and load activation; the fifth and sixth break symmetry to accelerate the search.

(1) Demand satisfaction. For each code i, the total units produced across all levels and loads must meet the production demand:

$$\sum_{j \in J_f} \sum_{k \in K_f} \eta_i \cdot z_{ijk} \geq d_i, \qquad \forall\, i \in I_f$$

The inequality permits the produced quantity to exceed d_i; the mechanism and its magnitude are discussed in Section 2.7.

(2) Chamber length capacity. The total length occupied by all active levels in load k must not exceed L:

$$\sum_{j \in J_f} \sum_{i \in I_f} (\mathrm{HEC}_i + \tau) \cdot z_{ijk} \;\leq\; (L + \tau) \cdot c_k, \qquad \forall\, k \in K_f$$

This is an equivalent reformulation of the physical gap-counting condition; the derivation is given in Section 2.7.

(3) Single radiator code per level. Each level may be assigned to at most one radiator code:

$$\sum_{i \in I_f} z_{ijk} \leq 1, \qquad \forall\, j \in J_f,\ \forall\, k \in K_f$$

This constraint captures the extra constraint from Chapter 1 requiring each level to contain a single code, with units replicated side by side.

(4) Level activation implies load activation. A level can only be active within an active load:

$$\sum_{i \in I_f} z_{ijk} \leq c_k, \qquad \forall\, j \in J_f,\ \forall\, k \in K_f$$

Since $c_k \in \{0, 1\}$, this bound subsumes constraint (3): when $c_k = 1$ it reduces to $\sum_i z_{ijk} \leq 1$, and when $c_k = 0$ it forces all levels in load k to be inactive. Constraint (3) is nonetheless retained explicitly because it expresses a distinct modelling requirement that would remain meaningful even in a reformulation where constraint (4) took a different form.

(5) Symmetry breaking — load ordering. Active loads precede inactive ones in index order:

$$c_k \geq c_{k+1}, \qquad \forall\, k \in \{0, \ldots, |K_f| - 2\}$$

(6) Symmetry breaking — level ordering. Within each load, levels are ordered by non-increasing HEC length:

$$\sum_{i \in I_f} \mathrm{HEC}_i \cdot z_{ijk} \ \geq \ \sum_{i \in I_f} \mathrm{HEC}_i \cdot z_{i,j+1,k}, \qquad \forall\, j \in \{0, \ldots, \bar{j}_f - 2\},\ \forall\, k \in K_f$$

The index range ends at $\bar{j}_f - 2$ so that the successor level $j + 1$ always lies within $J_f = \{0, \ldots, \bar{j}_f - 1\}$.

Both symmetry-breaking constraints reduce the number of equivalent solutions explored during branch-and-bound search without restricting the optimal value, a standard technique in bin packing formulations [7].

Figure 2.4 summarises the complete MILP formulation.

Sets I_f: codes in family f J_f: candidate levels K_f: candidate loads

Objective $\min \sum_{k \in K_f} c_k$

Subject to

(1) $\sum_{j \in J_f} \sum_{k \in K_f} \eta_i \cdot z_{ijk} \geq d_i, \quad \forall\, i \in I_f$ — demand

(2) $\sum_{j \in J_f} \sum_{i \in I_f} (\mathrm{HEC}_i + \tau) \cdot z_{ijk} \leq (L + \tau) \cdot c_k, \quad \forall\, k \in K_f$ — length capacity

(3) $\sum_{i \in I_f} z_{ijk} \leq 1, \quad \forall\, j \in J_f,\ \forall\, k \in K_f$ — one code per level

(4) $\sum_{i \in I_f} z_{ijk} \leq c_k, \quad \forall\, j \in J_f,\ \forall\, k \in K_f$ — level→load linking

(5) $c_k \geq c_{k+1}, \quad \forall\, k \in \{0, \ldots, |K_f| - 2\}$ — symmetry (loads)

(6) $\sum_{i \in I_f} \mathrm{HEC}_i \cdot z_{ijk} \geq \sum_{i \in I_f} \mathrm{HEC}_i \cdot z_{i,j+1,k}, \quad \forall\, j \in \{0, \ldots, \bar{j}_f - 2\},\ \forall\, k \in K_f$ — symmetry (levels)

Domains $c_k \in \{0, 1\}$ $z_{ijk} \in \{0, 1\}$

Figure 2.4: **Compact summary of the MILP model.** Objective function and constraints solved independently for each compatibility family.

2.7 Remarks on the Model

This section collects three clarifications on modelling choices that require justification beyond the constraint statements.

Derivation of constraint (2). The written form of constraint (2) differs from the physically intuitive gap-counting condition; this paragraph shows the two are equivalent. The physical condition for load k is that the sum of core lengths plus the $(n_k - 1)$ inter-level gaps does not exceed L:

$$\sum_{j \in J_f} \sum_{i \in I_f} \mathrm{HEC}_i \cdot z_{ijk} \;+\; \tau\,(n_k - 1) \;\leq\; L$$

where $n_k = \sum_j \sum_i z_{ijk}$ is the number of active levels. Adding $\tau \cdot c_k$ to both sides and noting that $n_k = 0$ whenever $c_k = 0$:

$$\sum_j \sum_i (\text{HEC}_i + \tau) \cdot z_{ijk} \;\leq\; (L + \tau) \cdot c_k$$

This is constraint (2). The reformulation is linear and avoids introducing n_k as an explicit auxiliary variable, keeping the model compact.

Over-production. Constraint (1) is a weak inequality ($\geq d_i$), so a code may be produced in excess of its demand. Two effects combine. Each level assigned to code i carries η_i units, so meeting d_i needs $\lceil d_i/\eta_i \rceil$ levels and forces a surplus of up to $\eta_i - 1$ units whenever d_i is not a multiple of η_i. Beyond this floor, the objective counts loads and not units, so the solver may fill spare level capacity in an already active load at no cost to the objective, producing a further surplus that the weak inequality permits. Both effects are small and industrially harmless: the extra units can be released to stock or held against forecast uncertainty. The surplus observed on the real instance is quantified in Chapter 5.

Symmetry breaking and load saturation. Constraints (5) and (6) remove permutation symmetries that would otherwise force the solver to explore many equivalent solutions. Constraint (6) does not break symmetry among levels assigned to codes with identical HEC values; on the current instance this residual symmetry has negligible impact, as CBC solves every family sub-problem to optimality within seconds.

No lower bound on load saturation is imposed. Minimising the number of loads already drives the solver towards compact, well-utilised configurations: an unnecessarily empty load forces additional loads elsewhere to cover the same demand, which directly worsens the objective. An explicit saturation floor would only restrict the feasible set without improving the objective, and could in fact require extra loads when residual demand falls below the threshold.

Chapter 3

Computational Implementation

3.1 Family Decomposition

The MILP formulation of Chapter 2 operates on a single compatibility family at a time. A global formulation covering all families at once only enlarges the search space, since the added freedom of assigning families to loads creates no further packing opportunities. The adopted solution is therefore a **decomposition by compatibility family**: a separate MILP sub-problem is solved for each family f, restricted to the codes in I_f and a family-specific candidate set K_f. The family-level solutions are ultimately combined into the final loading plan.

Figure 3.1 illustrates the adopted decomposition strategy.

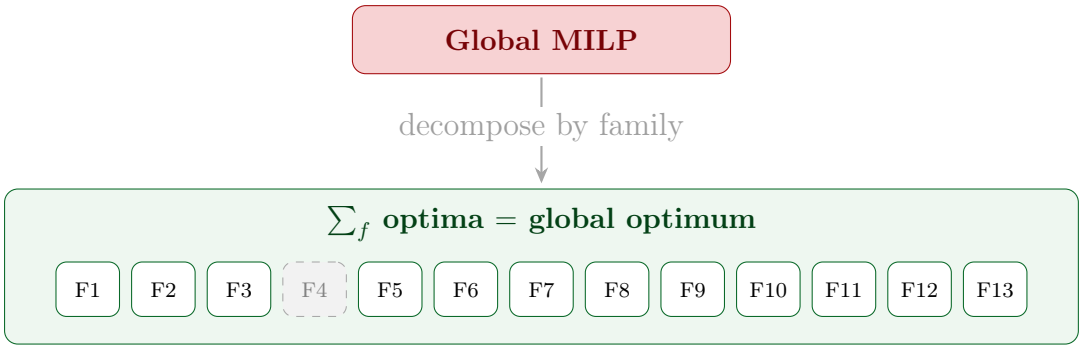


Figure 3.1: **Family decomposition strategy.**

Combining the family-level solutions in this way incurs no loss of optimality. The family-compatibility constraint already prevents radiators of different families from sharing a load, so each family's decisions are independent of the others, and solving the sub-problems separately leads to the same optimum as a single global model. This is a standard separability result for block-structured integer programmes [11].

The number of candidate loads for family f requires one quantity not needed in the formulation itself: the minimum number of loading levels that fit in a single load, denoted $\bar{j}_f^{\min}$ and defined as

$$\bar{j}_f^{\min} = \left\lfloor \frac{L + \tau}{\max_{i \in I_f} \mathrm{HEC}_i + \tau} \right\rfloor .$$

This is the level count achieved when every level is occupied by the longest code in the family; any load with shorter codes admits at least as many levels, so $\bar{j}_f^{\min}$ is a conservative lower bound on the levels available per load. The $(L + \tau)$ numerator follows the same gap-counting convention as $\bar{j}_f$ (Section 2.2): a stack of n levels contains $n - 1$ inter-level gaps, so the usable space is $L + \tau$ when expressed in units of $(\mathrm{HEC}_i + \tau)$.

The number of candidate loads $|K_f|$ is set to

$$|K_f| = \left\lceil \frac{\sum_{i \in I_f} \lceil d_i / \eta_i \rceil}{\bar{j}_f^{\min}} \right\rceil .$$

The numerator $\sum_{i \in I_f} \lceil d_i / \eta_i \rceil$ counts the total level slots needed to cover all demand in family f, taking one dedicated level per η_i units of code i. Dividing by $\bar{j}_f^{\min}$, the minimum number of levels any load can accommodate, gives an upper bound on the loads required. Any optimal solution uses at most this many loads, so restricting the candidate set to K_f is without loss of optimality. The bound is conservative: it ignores the extra capacity gained by mixing shorter codes within a load, so optimal solutions typically use strictly fewer than $|K_f|$ loads.

3.2 Python Implementation

The decomposition is implemented in Python 3 using PuLP [9], a modelling library for linear and mixed-integer programmes, with CBC (COIN-OR Branch and Cut) as the underlying solver [2]. Three standard packages are required: `pulp`, `pandas`, and `openpyxl`. The input is a single Excel workbook (`Orders.xlsx`), shared by both the MILP and the heuristic, with one row per radiator code and the six columns listed in Table 3.1. Demand is aggregated by summing quantities over all rows sharing the same part code before the optimisation begins.

Table 3.1: **Input structure of the order workbook.**

Column	Description
Part Code	Unique code identifying the radiator model
Family	Compatibility family (F1–F13)
Cycle	Thermal cycle, e.g. “Cycle 1”, “Cycle 10”
HEC	Core length in the furnace [mm], frame excluded
SP	Core width [mm], frame excluded
Quantity	Ordered quantity

The implementation is organised into five functional blocks, executed in sequence each time the script is invoked.

1. **Global parameters.** Five scalar constants define the furnace geometry and clearance values: L, W, τ, σ, and δ.
2. **Data ingestion** (`load_from_excel`). Reads `Orders.xlsx`, maps family codes to F1–F13, aggregates duplicate rows by summing quantities, and constructs a list of `Radiator` objects. The value of η_i is computed as a property of each object from the geometric parameters of step 1.
3. **Family sub-problem solver** (`build_and_solve_family`). Receives the list of codes belonging to one family, builds the MILP sub-problem for that family as described in Section 3.1, and calls CBC. Returns the list of furnace loads with their level assignments.
4. **Decomposition coordinator** (`build_and_solve`). Iterates over all families present in the input, calls `build_and_solve_family` for each, and assembles the results into a single global solution.
5. **Output generation** (`export_to_excel`). Writes `load_results.xlsx` with two worksheets: *Loads*, with one row per loading level reporting load index, family, radiator code, HEC, quantity, occupied length and saturation; and *Demand check*, with one row per code reporting ordered and produced quantities and a fulfilment status indicator.

The script is invoked as:

```
python MILP.py --excel Orders.xlsx
```

The overall data flow is summarised in Figure 3.2.

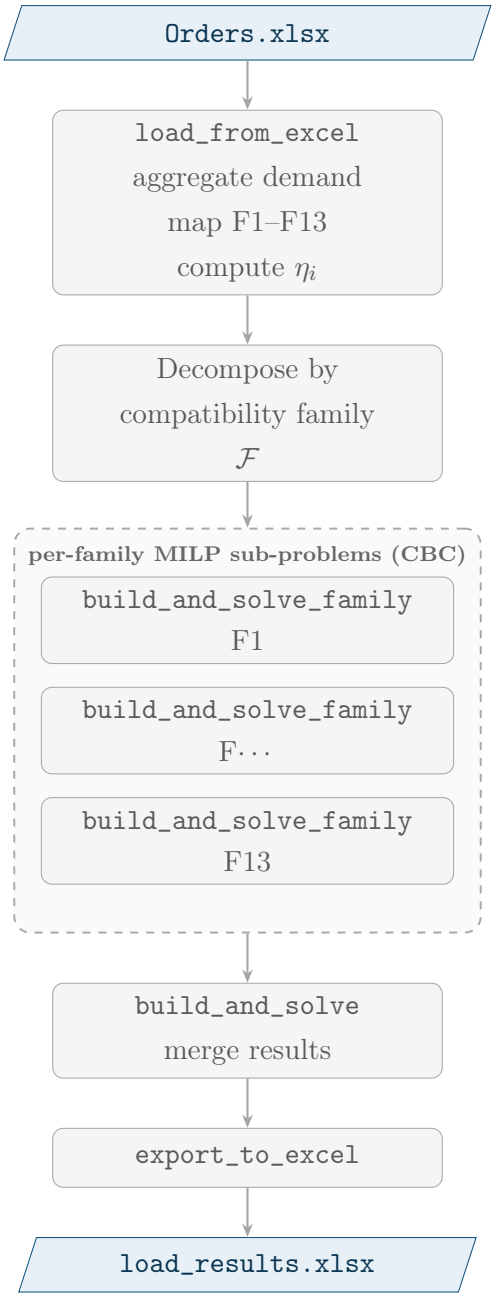


Figure 3.2: **MILP computational pipeline.** The input workbook is decomposed by compatibility family, solved through independent CBC sub-problems, and merged into a single result workbook.

3.3 Computational Performance

Each family sub-problem is solved to optimality by CBC. On the real-world instance of Chapter 5, all 12 active family sub-problems converge within a few seconds each, for a total computation time well under one minute on a standard laptop. The per-family decomposition is therefore compatible with routine daily production planning use. A detailed analysis of computation times across the synthetic benchmark is reported in Chapter 5.

3.4 Load Saturation

The performance of a loading plan is measured by how completely each furnace load fills the chamber. The chamber has two dimensions, but only one of them is a decision: the lateral direction is saturated by construction, since each level holds η_i units of a single code placed side by side, the maximum the width admits for that code (Section 2.1). Lateral occupancy is therefore fixed once the code is chosen, and the only quantity the optimisation can influence is how the levels are composed along the chamber length. The relevant indicator is consequently the longitudinal saturation of the load.

For load k with n_k active levels, it is defined as:

$$\rho_k = \frac{\displaystyle\sum_{j \in J_f} \sum_{i \in I_f} \mathrm{HEC}_i \cdot z_{ijk} \;+\; \tau\,(n_k - 1)}{L}$$

where $n_k = \sum_{j,i} z_{ijk}$ is the number of active levels. The numerator is the total longitudinal space physically occupied: the sum of core lengths plus the $n_k - 1$ inter-level gaps. The denominator is the usable chamber length L. A value of ρ_k close to 1 indicates a nearly full chamber; values significantly below 1 signal recoverable longitudinal waste. The distribution of ρ_k across all loads is reported in Chapter 5.

Chapter 4

Constructive Heuristic

4.1 Motivation

The previous chapters introduced an exact MILP formulation for the furnace loading problem. In order to evaluate the effective contribution of exact optimisation, this chapter presents a constructive heuristic based on the same geometric assumptions and compatibility constraints. The heuristic provides a reference solution generated through sequential greedy decisions, allowing the performance improvement achieved by the MILP to be quantified on both real and synthetic instances. This approach does not guarantee optimality, but it is computationally instantaneous (typical runtime: 0.02 seconds) and, as shown in the benchmark of Chapter 5.3, produces solutions that are often close to the MILP optimum.

4.2 Capacity Parameters

The heuristic operates under the same orthogonal geometric capacity model as the MILP. This ensures that any difference in the number of loads produced by the two methods is attributable entirely to the difference in solution strategy, and not to differences in how capacity is computed.

Lateral capacity η_i. The maximum number of radiators of code i that can be placed side by side across the chamber width is:

$$\eta_i = \max\left(1,\ \left\lfloor \frac{W + \sigma}{\mathrm{SP}_i + 2\delta + \sigma} \right\rfloor\right)$$

Longitudinal capacity. The maximum number of loading levels for code i that fit along the chamber length is:

$$m_i = \max\left(1,\ \left\lfloor \frac{L+\tau}{\mathrm{HEC}_i+\tau} \right\rfloor\right)$$

This expression coincides with the single-code level bound implied by constraint (2): when load k is dedicated entirely to code i, constraint (2) reduces to $\mathrm{HEC}_i+\tau \le L+\tau$, yielding exactly this formula. In multi-code families, the MILP manages the mixed-code bound implicitly. The total capacity of one full load for code i is therefore:

$$\mathrm{cap}_i = \eta_i \cdot m_i$$

Saturation. The loading saturation of a furnace load is computed consistently with the MILP definition of Section 3.4:

$$\rho_k = \frac{\sum_j \mathrm{HEC}_j \cdot l_j \ +\ \tau\,(n_k - 1)}{L}$$

where the sum runs over all codes j assigned to the load, l_j is the number of levels occupied by code j, and $n_k = \sum_j l_j$ is the total number of active levels. The numerator is the total physical longitudinal space occupied: the sum of core lengths plus the n_k-1 inter-level gaps.

4.3 Heuristic Algorithm

The heuristic proceeds in two phases.

4.3.1 Phase 1: Full Loads

For each radiator code i, independently of all others, the algorithm computes the number of full loads that can be filled at maximum capacity:

$$n_i^{\mathrm{full}} = \left\lfloor \frac{d_i}{\mathrm{cap}_i} \right\rfloor, \qquad r_i = d_i - n_i^{\mathrm{full}}\,\mathrm{cap}_i$$

Each of these n_i^{full} loads is recorded as a complete load dedicated entirely to code i, consisting of m_i levels each loaded with η_i units side by side. The remaining r_i units form the residual for Phase 2. The saturation of a full load for code i follows directly from the definition above:

$$\rho_i^{\mathrm{full}} = \frac{m_i \cdot \mathrm{HEC}_i + \tau\,(m_i - 1)}{L}$$

4.3.2 Phase 2: Residual Combination

After Phase 1, each code i with $r_i > 0$ contributes a residual demand of r_i units. Codes belonging to the same compatibility family can share a load; since family membership already encodes all three compatibility criteria (thermal cycle, fin insert and thickness class), grouping by family alone is sufficient. Phase 2 exploits this by combining residuals from the same family into shared loads using a **First-Fit Decreasing** (FFD) bin-packing procedure. FFD is one of the best-known constructive heuristics for bin packing problems because of its simplicity and good empirical performance [5]; its level-based two-dimensional extension is analysed in [1].

Within each family group:

1. Sort the residual codes by HEC_i in non-increasing order. Ties in HEC_i are broken arbitrarily (e.g. by code index); the resulting solution quality is insensitive to the tie-breaking rule in the current instance.

2. For each code (in sorted order), scan the open bins from first to last. Compute the remaining longitudinal space in each bin as:

$$s_b^{\mathrm{free}} = (L + \tau) - \sum_{e \in b} (\mathrm{HEC}_e + \tau) \cdot n_e$$

   where n_e is the number of levels already occupied by code e in bin b. The number of additional level slots available for code i is $\lfloor s_b^{\mathrm{free}} / (\mathrm{HEC}_i + \tau) \rfloor$, which accommodates $\eta_i \cdot \lfloor s_b^{\mathrm{free}} / (\mathrm{HEC}_i + \tau) \rfloor$ further units. Place as many units as fit (up to the remaining residual demand) in the first bin with available space.

3. If any quantity remains unallocated after scanning all open bins, open a new bin and place as many units as fit.

4. Repeat until all residual demand is allocated.

Each closed bin corresponds to a furnace load. Its saturation ρ^{res} is computed with the same formula as above, applied to the actual contents of the bin.

Figure 4.1 summarises the described two-phases heuristic.

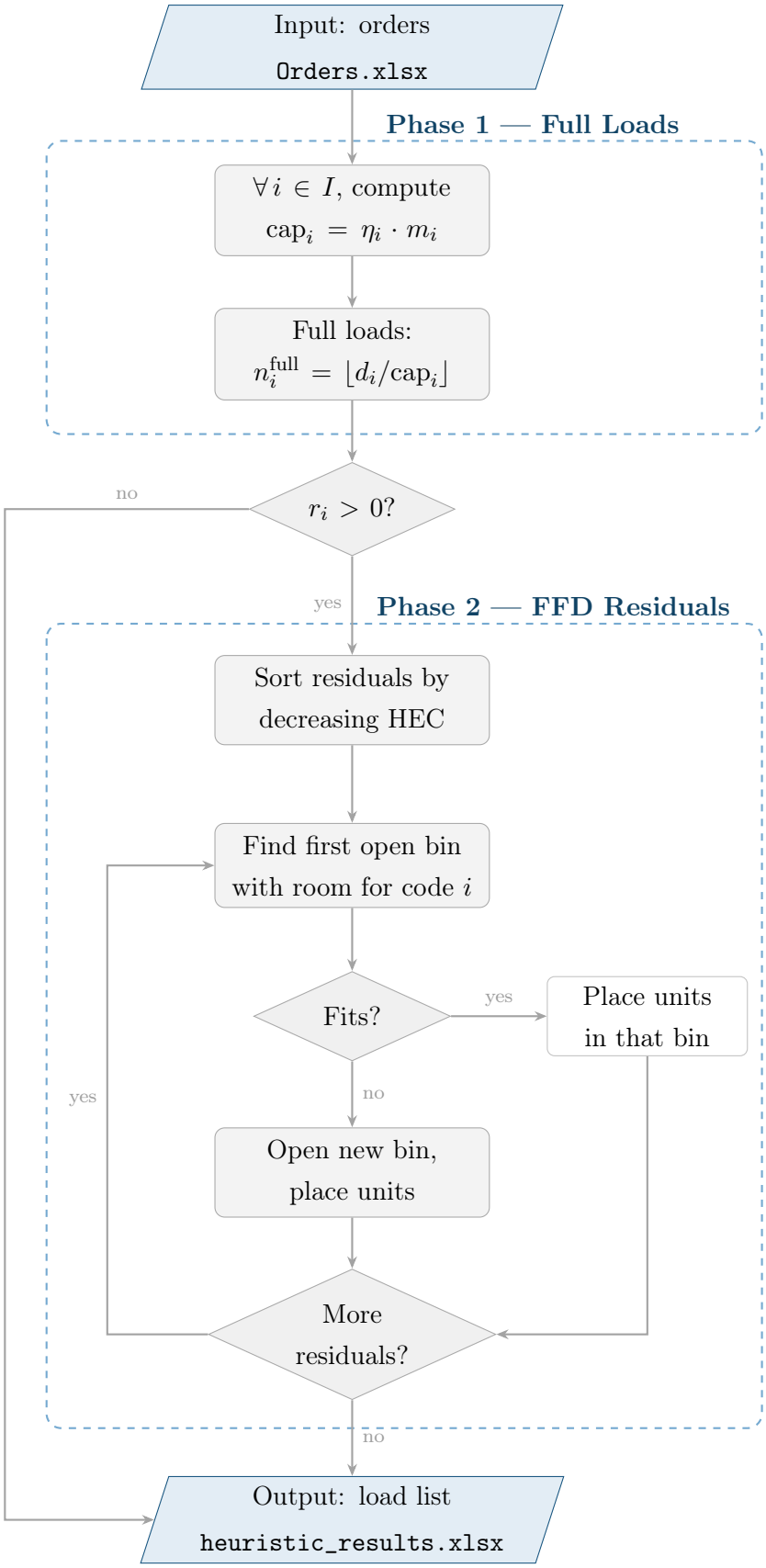


Figure 4.1: **Two-phase constructive heuristic.** Phase 1 generates full mono-code loads, while Phase 2 combines residual demand through a First-Fit Decreasing procedure.

4.4 Relationship to the MILP

The heuristic and the MILP share the same objective (minimise furnace loads), the same family-compatibility constraint, and the same orthogonal geometric capacity model. They differ only in solution strategy.

The MILP solves a global optimisation problem: it simultaneously assigns all codes to loads, searching the entire combinatorial space for the minimum number of loads. The heuristic is greedy: it commits irrevocably to decisions in a fixed order. Phase 1 always fills the easiest loads first (one code at maximum capacity), which is locally optimal but globally suboptimal — it does not consider whether a different demand partition might leave residuals that combine more efficiently in Phase 2. Phase 2 applies FFD, which is a good but not optimal bin-packing heuristic.

As a consequence, the heuristic uses more loads than the MILP in the majority of instances. The MILP's advantage is most pronounced when residuals from different codes within the same family have lengths that would pack well together but are not combined by the greedy Phase 1 partition. The systematic quantification of this gap is the subject of Chapter 5.3.

Chapter 5

Solution Approaches and Computational Results

5.1 Model Validation on a Toy Instance

To verify the model and illustrate the mechanism by which the MILP outperforms the heuristic, this section constructs a minimal toy instance and traces the complete solution. The instance is deliberately small, so that constraint satisfaction and optimality can be checked by hand. Its codes are drawn from the narrow regime (Cycle 3, core widths below 65 mm), where the lateral capacity reaches $\eta_i = 2$ and each level carries two radiators side by side. The figures therefore exhibit side-by-side lateral loading explicitly, complementing the wide single-radiator families ($\eta_i = 1$) examined in Section 5.2.3.

5.1.1 Instance Definition

The toy instance comprises three radiator codes of the same compatibility family in the narrow regime (Cycle 3). The furnace parameters match the full model: $L = 2700$ mm, $W = 440$ mm, $\tau = 40$ mm, $\delta = 70$ mm, $\sigma = 10$ mm.

Table 5.1: Toy instance data

Code	HEC [mm]	SP [mm]	η_i	m_i	cap_i	d_i
A	750	63	2	3	6	4
B	680	63	2	3	6	6
C	560	63	2	4	8	4

Derived parameters. The three codes share the same core width, SP = 63 mm, so the lateral capacity is identical for all of them, following the formula of Section 2.1:

$$\eta_A = \eta_B = \eta_C = \max\left(1,\ \left\lfloor \frac{440+10}{63+2\times 70+10} \right\rfloor\right) = \left\lfloor \frac{450}{213} \right\rfloor = 2.$$

Two radiators therefore occupy each level across the chamber width. The maximum number of levels per load follows from the height clearance of Section 4.2:

$$m_A = \left\lfloor \frac{2740}{790} \right\rfloor = 3, \qquad m_B = \left\lfloor \frac{2740}{720} \right\rfloor = 3, \qquad m_C = \left\lfloor \frac{2740}{600} \right\rfloor = 4,$$

giving $\text{cap}_A = \text{cap}_B = 6$ and $\text{cap}_C = 8$ units.

Key geometric observation. Codes B and C are longitudinally complementary: two levels of each fill the chamber almost exactly,

$$2 \times 680 + 2 \times 560 + 3 \times 40 = 2600 \text{ mm} \leq L = 2700 \text{ mm},$$

whereas a single-code B load tiles the chamber wastefully. Three B levels occupy $3 \times 680 + 2 \times 40 = 2120$ mm, and the 580 mm left idle cannot hold a fourth level, which would require $680 + 40 = 720$ mm. This contrast is the source of the gap analysed below.

5.1.2 MILP Optimal Solution

The MILP produces **2 furnace loads**, shown in Table 5.2.

Table 5.2: Toy instance MILP solution

Load k	Level j	Variable	Code	HEC [mm]	Occupied length [mm]
1	0	$z_{B,0,1} = 1$	B	680	$2 \cdot 680 + 2 \cdot 560 + 3 \cdot 40 = 2600$
1	1	$z_{B,1,1} = 1$	B	680	
1	2	$z_{C,2,1} = 1$	C	560	
1	3	$z_{C,3,1} = 1$	C	560	
2	0	$z_{A,0,2} = 1$	A	750	$2 \cdot 750 + 680 + 2 \cdot 40 = 2260$
2	1	$z_{A,1,2} = 1$	A	750	
2	2	$z_{B,2,2} = 1$	B	680	

Saturation: load 1 achieves 2600/2700 = 96.3%; load 2 achieves 2260/2700 = 83.7%. Each level carries $\eta_i = 2$ radiators, so the two loads hold 14 units in total (4 of code A, 6 of code B, 4 of code C).

5.1.3 Manual Verification of Constraints

We verify constraints (1)–(4) on the solution of Table 5.2.

Constraint (1) — Demand satisfaction. The lateral capacity $\eta_i = 2$ multiplies every assigned level:

$$\text{Code A:} \quad \eta_A(z_{A,0,2} + z_{A,1,2}) = 2 \cdot (1+1) = 4 \;\geq\; d_A = 4 \quad \checkmark$$

$$\text{Code B:} \quad \eta_B(z_{B,0,1} + z_{B,1,1} + z_{B,2,2}) = 2 \cdot (1+1+1) = 6 \;\geq\; d_B = 6 \quad \checkmark$$

$$\text{Code C:} \quad \eta_C(z_{C,2,1} + z_{C,3,1}) = 2 \cdot (1+1) = 4 \;\geq\; d_C = 4 \quad \checkmark$$

Constraint (2) — Chamber length capacity. For each active load ($c_1 = c_2 = 1$):

$$k = 1: \quad 2\,(680 + 40) + 2\,(560 + 40) = 1440 + 1200 = 2640 \;\leq\; 2740 \cdot 1 \quad \checkmark$$

$$k = 2: \quad 2\,(750 + 40) + 1\,(680 + 40) = 1580 + 720 = 2300 \;\leq\; 2740 \cdot 1 \quad \checkmark$$

Constraint (3) — Single code per level. For every active level in both loads, $\sum_i z_{ijk} = 1 \leq 1$. For all inactive levels, $\sum_i z_{ijk} = 0 \leq 1$. ✓

Constraint (4) — Level activation implies load activation. All active levels belong to loads with $c_k = 1$, so $\sum_i z_{ijk} \leq c_k = 1$ is satisfied throughout. ✓

5.1.4 Heuristic Solution

For comparison, the two-phase heuristic of Chapter 4 proceeds as follows.

Phase 1 (full loads).

$$\begin{aligned} n_A^{\text{full}} &= \lfloor d_A / \text{cap}_A \rfloor = \lfloor 4/6 \rfloor = 0, \qquad & r_A &= 4 \\ n_B^{\text{full}} &= \lfloor d_B / \text{cap}_B \rfloor = \lfloor 6/6 \rfloor = 1, \qquad & r_B &= 0 \\ n_C^{\text{full}} &= \lfloor d_C / \text{cap}_C \rfloor = \lfloor 4/8 \rfloor = 0, \qquad & r_C &= 4 \end{aligned}$$

Phase 1 produces one full B-load: three levels of code B (six units), occupying $3 \times 680 + 2 \times 40 = 2120$ mm at 78.5% saturation.

Phase 2 (FFD on residuals). Residual demand is $r_A = 4$ units and $r_C = 4$ units. The codes are processed in non-increasing HEC order (A before C).

- Code A: no bin is open. Open bin 1; the free space $s_1^{\text{free}} = 2740$ mm admits $\lfloor 2740/790 \rfloor = 3$ level slots (6 units). Place the four A units as two A levels. Bin 1: $\{A, A\}$, $r_A = 0$.
- Code C: scan bin 1. $s_1^{\text{free}} = 2740 - 2\,(750 + 40) = 1160$ mm admits $\lfloor 1160/600 \rfloor = 1$ slot (2 units). Place two C units as one C level. Bin 1: $\{A, A, C\}$; two C units remain.
- Two C units remain after the scan. Open bin 2 and place them as one C level. Bin 2: $\{C\}$, $r_C = 0$.

Bin 1 (A, A, C) reaches 2140/2700 = 79.3%; bin 2 (C alone) reaches 560/2700 = 20.7%. Phase 2 adds two loads, for a heuristic total of **3**.

Gap. The MILP uses two loads against the heuristic's three, a saving of one load (33%). The cause is the Phase 1 partition. By filling code B to capacity in a single homogeneous load, the heuristic consumes all six B units at 78.5% saturation, and the 580 mm left idle in that load cannot accommodate a further level. The residual A and C units then pack into one shared load and one load that is almost empty (20.7%), because Phase 2 never reopens the committed B-load. The MILP declines the full B-load: it pairs two B levels with two C levels into a load at 96.3% and places the third B level alongside the two A levels, recovering the longitudinal complementarity between codes B and C. This is the Phase 1 over-commitment analysed for families F10 and F11 in Section 5.2.3. The gap is driven by longitudinal (HEC) complementarity and is independent of the lateral capacity η: the narrow codes used here ($\eta_i = 2$) reproduce the mechanism that the wide single-radiator codes ($\eta_i = 1$) of F10 and F11 exhibit in the order book.

5.1.5 Visual Representation

Figures 5.1 and 5.2 illustrate the two-load MILP solution and the three-load heuristic solution, drawn to scale.

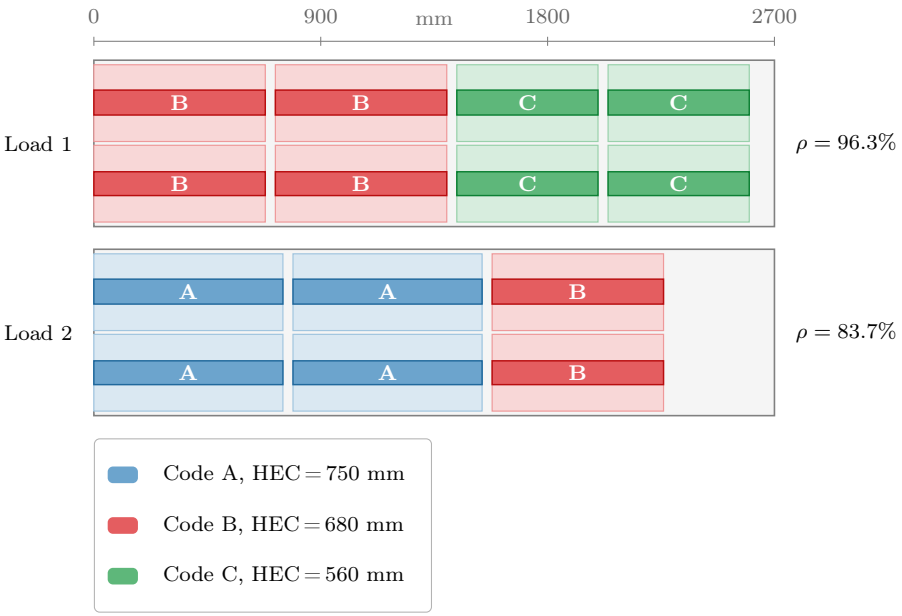


Figure 5.1: **Optimal MILP solution for the toy instance.**

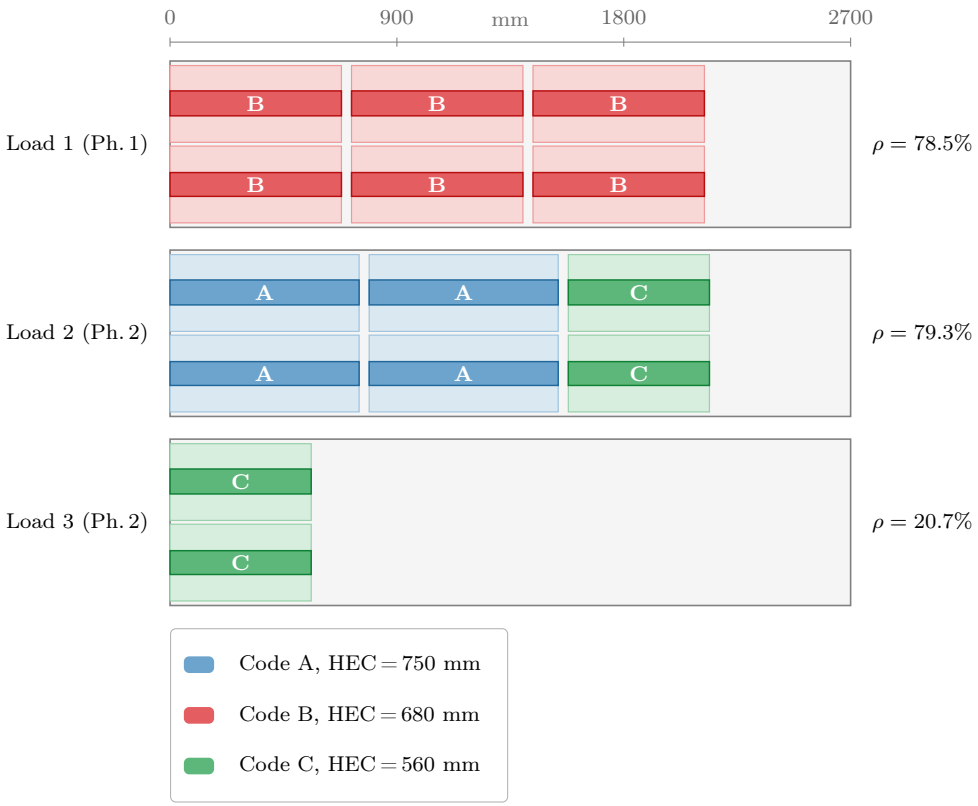


Figure 5.2: **Heuristic solution for the toy instance.**

5.2 Results on the Real-World Instance

5.2.1 Test Instance

Section 5.1 validated the model on a controlled toy instance and traced the mechanism behind the MILP advantage by hand. This section applies both methods to the real planning problem: 20 radiator codes drawn from the current order book at Tesio Cooling Systems S.p.A., with demand aggregated over the planning horizon. The instance activates 11 of the 13 compatibility families of Table 1.1. Family F4 is absent from the order book, and the current horizon carries no demand for family F2; the four thermal cycles (1, 2, 3 and 10) are all represented. Table 5.3 reports the full parametric data.

Table 5.3: Real-world instance data

Code	Family	Cycle	HEC [mm]	SP [mm]	η_i	m_i	cap_i	Demand [units]
C1	F13	10	1145	300	1	2	2	18
C2	F1	1	1002	113	1	2	2	66
C3	F6	2	1091	82	1	2	2	27
C4	F11	10	800	300	1	3	3	118
C5	F12	10	915	150	1	2	2	4
C6	F8	10	976	164	1	2	2	2
C7	F6	2	670	92	1	3	3	23
C8	F3	1	661	100	1	3	3	5
C9	F12	10	615	200	1	4	4	2
C10	F10	10	696	152	1	3	3	38
C11	F11	10	903	300	1	2	2	20
C12	F3	1	670	124	1	3	3	5
C13	F10	10	722	200	1	3	3	36
C14	F5	2	701	85	1	3	3	16
C15	F10	10	1160	203	1	2	2	60
C16	F12	10	1051	185	1	2	2	22
C17	F8	10	498	182	1	5	5	37
C18	F9	10	1015	190	1	2	2	5
C19	F8	10	1137	193	1	2	2	11
C20	F7	3	750	63	2	3	6	30

Every code has $\eta_i = 1$ except C20 (family F7, SP = 63 mm), which takes two units side by side. The loading problem on this instance is therefore essentially one-dimensional along the chamber length, with lateral packing entering only through that single code.

Total demand is 545 units.

5.2.2 MILP Results

Aggregate Performance

The MILP produces **202 furnace loads**, covering all 20 codes with demand fully satisfied, at a global average saturation of **87.8%**. Table 5.4 gives the breakdown by family.

Table 5.4: **MILP results on the real-world instance.**

Family	Thermal cycle	Furnace loads	Avg. saturation (%)
F1	Cycle 1	33	75.7
F3	Cycle 1	4	77.0
F5	Cycle 2	6	71.7
F6	Cycle 2	20	88.7
F7	Cycle 3	5	86.3
F8	Cycle 10	14	90.8
F9	Cycle 10	3	76.7
F10	Cycle 10	49	94.8
F11	Cycle 10	46	93.6
F12	Cycle 10	13	81.5
F13	Cycle 10	9	86.3
Total		**202**	**87.8**

Saturation Distribution

The average alone does not describe how the loads are spread. Table 5.5 shows that the MILP concentrates them in the high-utilisation band: 52% of loads exceed 90% saturation, and only three fall below 60%.

Table 5.5: MILP saturation distribution

Saturation range	Loads	Share (%)
$< 50\%$	1	0.5
50–60%	2	1.0
60–70%	1	0.5
70–80%	52	25.7
80–90%	41	20.3
90–100%	105	52.0
Total	**202**	**100**

These three loads are worth examining, because they mark the floor imposed by the demand structure rather than any solver weakness.

- **Family F8, 42.1%** (one level, 1137 mm). Code C19 has $\text{cap}_i = 2$ and $d_i = 11$, so after five full two-level loads a single unit remains. That unit occupies one level of 1137 mm, i.e. $1137/2700 = 42.1\%$ of the chamber. With $\eta_i = 1$ no packing can improve on a one-unit residual, so the floor is structural and independent of the method.
- **Family F5, two loads at 53.4%.** Code C14 has $\text{cap}_i = 3$ and $d_i = 16$, which requires six loads and cannot fill all of them. CBC returns four full three-level loads at 80.9% and two two-level loads at $(2 \times 701 + 40)/2700 = 53.4\%$. The same residual capacity could instead be held in a single one-level load without changing the load count. Either way, at least six loads are needed for 16 units at $\text{cap}_i = 3$.

Over-Production

Constraint (1) satisfies demand as a weak inequality ($\geq d_i$), and the objective minimises load count rather than units produced. Where the minimum-load packing leaves a spare level slot inside a load, the solver may fill it beyond a code's demand at no cost to the objective. Four codes carry such a surplus in the returned solution.

Table 5.6: **Over-production in the MILP solution.** Surplus = Produced − Required.

Code	Family	Required	Produced	Surplus
C3	F6	27	28	1
C7	F6	23	24	1
C12	F3	5	7	2
C18	F9	5	6	1

The total surplus is 5 units against 545 demanded, or 0.9%. It is a property of the particular optimal solution CBC returns, not a forced effect: a solution with the same 202 loads and no surplus also exists, since the extra levels sit in loads that stay active for other codes and could be left empty without opening a load. The heuristic, which allocates levels by exact residual count, produces no surplus on this instance. Industrially the surplus is immaterial and can be released to stock or held against forecast uncertainty.

5.2.3 Heuristic Results

Aggregate Comparison

Applied under the same orthogonal geometric capacity model, the heuristic uses 214 loads. Table 5.7 sets the two methods side by side.

Table 5.7: **Aggregate comparison on the real-world instance.**

Method	Furnace loads	Avg. saturation (%)
MILP	**202**	**87.8**
Heuristic	214	82.0
Gap (Δ)	12 (5.9%)	−5.8 pp

The MILP saves 12 loads, a 5.9% reduction. Since both methods share the capacity model, this gap isolates the value of exact global optimisation over greedy two-phase construction.

Family-Level Breakdown and Phase Decomposition

Table 5.8 locates the gap by family and splits the heuristic solution into its Phase 1 and Phase 2 contributions.

Table 5.8: **Family-level comparison between MILP and heuristic.**

Family	Cycle	MILP		Heuristic			
		Loads	Sat. (%)	Ph. 1	Ph. 2	Total	Sat. (%)
F1	1	33	75.7	33	0	33	75.7
F3	1	4	77.0	2	2	4	63.8
F5	2	6	71.7	5	1	6	71.8
F6	2	20	88.7	20	1	21	81.2
F7	3	5	86.3	5	0	5	86.3
F8	10	14	90.8	13	1	14	90.8
F9	10	3	76.7	2	1	3	63.7
F10	10	49	94.8	54	1	55	84.3
F11	10	46	93.6	49	1	50	86.0
F12	10	13	81.5	11	3	14	75.6
F13	10	9	86.3	9	0	9	86.3
Total		**202**	**87.8**	193	21	**214**	**82.0**

The gap sits almost entirely in two families. F10 (+6) and F11 (+4) account for 10 of the 12 saved loads, and F6 and F12 supply the other two. The remaining seven families match exactly.

Mechanistic Analysis of the F10 and F11 Gaps

The MILP advantage in F10 and F11 is not global search in the abstract. It rests on a specific property of the demand data that the heuristic's Phase 1 partition cannot use.

Family F10. The family holds three codes: C15 (HEC = 1160, $d = 60$), C13 (HEC = 722, $d = 36$) and C10 (HEC = 696, $d = 38$). All three fit in a single load,

$$1160 + 722 + 696 + 2 \times 40 = 2658 \text{ mm} \leq L = 2700 \text{ mm},$$

and every pair fits with room to spare ($1160+722+40 = 1922$; $1160+696+40 = 1896$; $722+696+40 = 1458$). The MILP uses these combinations: 38 of its 49 F10 loads are multi-code, saturated between 97.5% and 99.4%. Phase 1 instead fills $\lfloor 60/2 \rfloor = 30$ mono-code loads for C15, $\lfloor 36/3 \rfloor = 12$ for C13 and $\lfloor 38/3 \rfloor = 12$ for C10, then clears the residuals in Phase 2. FFD cannot recover the mixing that Phase 1 gave away, and the family ends six loads heavier.

Family F11. The family holds two codes: C4 (HEC = 800, $d = 118$) and C11 (HEC = 903, $d = 20$). The two lengths share a load comfortably,

$$903 + 800 + 40 = 1743 \text{ mm} \leq L,$$

and three levels remain feasible ($903+800+800+2\times40 = 2583$; $903+903+800+2\times40 = 2686$). The MILP places 14 multi-code loads out of 46, reaching up to 99.5%; Figure 5.4 shows one at 95.7%. Phase 1 instead dedicates $\lfloor 118/3 \rfloor = 39$ loads to C4 and $\lfloor 20/2 \rfloor = 10$ to C11, leaving one residual load for Phase 2.

The pattern is identical in both families. Rather than partition demand by code and then combine the leftovers, the MILP assigns cross-code levels together and recovers longitudinal space that the greedy partition gives up permanently.

Figure 5.3 shows a three-code F10 load filled to 98.4%, and Figure 5.4 the matching F11 load, one level of C11 above two of C4. Figure 5.5 shows a structural-floor load, low not through suboptimality but through an indivisible residual, while Figure 5.6 sets the heuristic's mono-code Phase 1 load against the MILP's consolidation for F10, an 11.0 percentage-point gain per load.

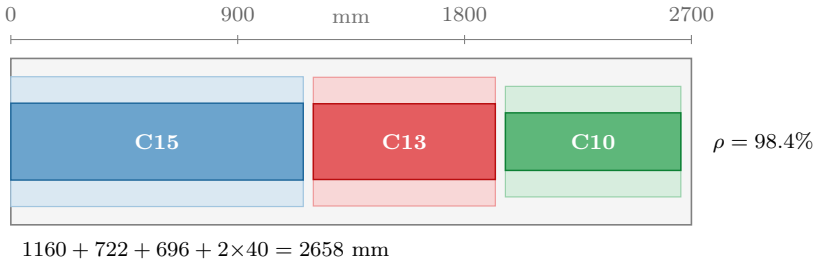


Figure 5.3: **F10 multi-code load.** The MILP combines three radiator codes in a single load, reaching 98.4% longitudinal saturation. Each radiator is shown as core plus frame, to scale: the frame overhangs the core by δ in the width direction, so the narrower code C10 (SP = 152 mm) leaves more lateral clearance than C15.

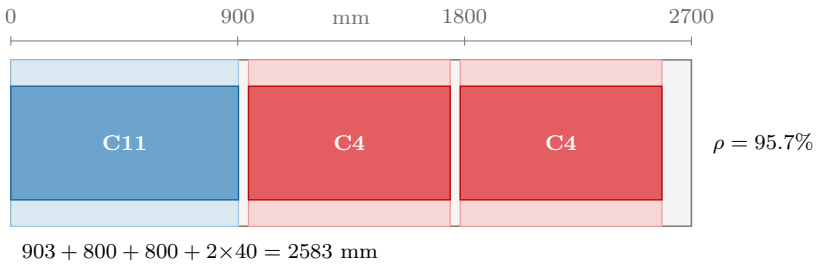


Figure 5.4: **F11 multi-code load.** The MILP combines one level of code C11 with two levels of code C4, reaching 95.7% longitudinal saturation. With SP = 300 mm these codes fill the guide edge to edge, so the frames span the full chamber width.

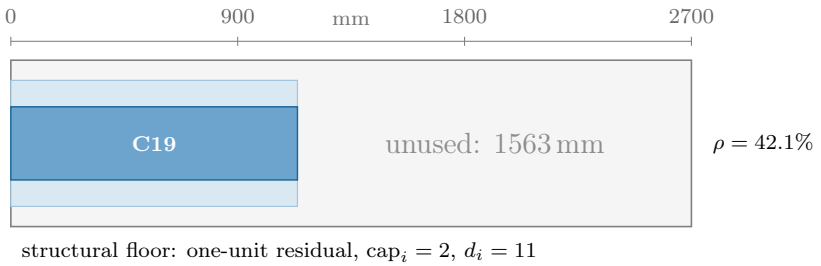


Figure 5.5: **F8 structural low-saturation load.** The low utilisation is caused by an indivisible demand residual rather than by solver suboptimality.

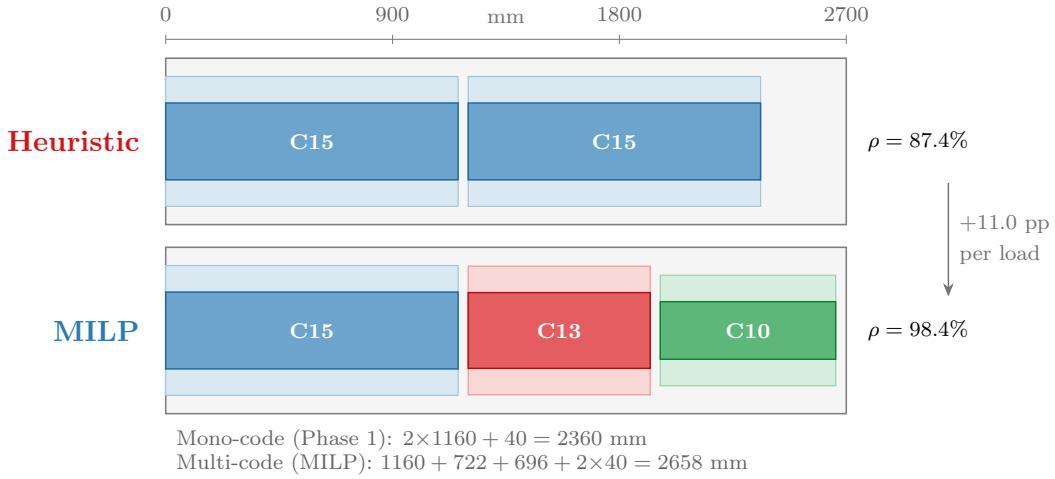


Figure 5.6: **F10 loading comparison.** The heuristic produces a mono-code load at 87.4% saturation, while the MILP exploits cross-code consolidation and reaches 98.4%.

5.2.4 Saturation Comparison and Phase Structure

Table 5.9 and Figure 5.7 compare the two saturation distributions, and the difference is structural rather than marginal.

Table 5.9: MILP–heuristic saturation comparison

Saturation range	MILP		Heuristic	
	Loads	%	Loads	%
< 50%	1	0.5	5	2.3
50–70%	3	1.5	13	6.1
70–80%	52	25.7	57	26.6
80–90%	41	20.3	92	43.0
90–100%	105	52.0	47	22.0
Total	**202**	**100**	**214**	**100**

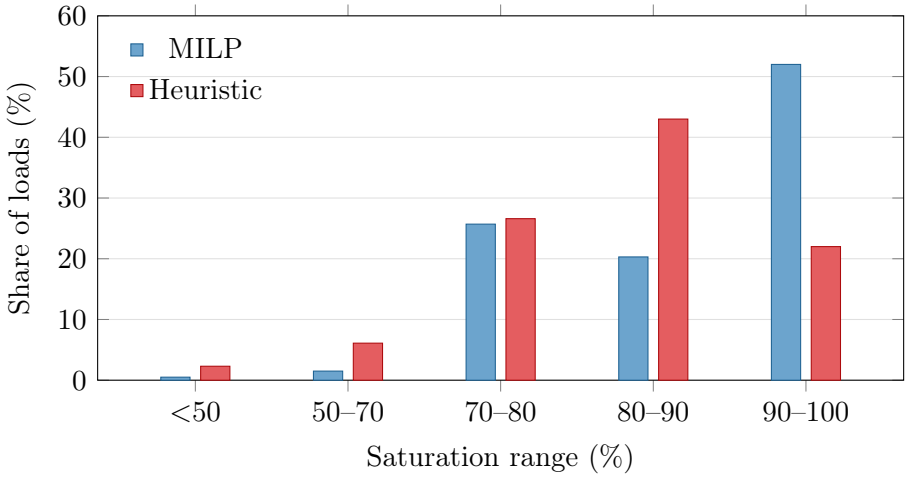


Figure 5.7: **Saturation distribution on the real-world instance.** The MILP concentrates a larger share of loads above 90% saturation, while the heuristic generates more medium- and low-saturation residual loads; exact counts are reported in Table 5.9.

The MILP holds 52% of loads above 90% saturation against the heuristic's 22%. The heuristic's Phase 1 produces 193 mono-code loads, clustered in the 70–90% band, while its 21 Phase 2 residual loads average 64% and pull the overall mean down by 5.8 percentage points.

5.2.5 Industrial Impact Assessment

The 12-load gap can be priced against the current furnace. A reference brazing session at Tesio (23 March 2026, nine-hour recording) measured a mean active draw of about 65 kW on the existing Abar-Ipsen unit. With a typical two-hour cycle and an all-in operating cost of about €102/hour (energy, labour, overhead), the cost of one load is

$$c_{\text{load}} = 2\ \text{h} \times €\,102/\text{h} = €\,204.$$

The 12-load advantage is therefore worth

$$\Delta C = 12 \times €\,204 = €\,2{,}448 \quad \text{per planning horizon,}$$

with the MILP costing €41,208 ($202 \times €\,204$) against the heuristic's €43,656 ($214 \times €\,204$), as summarised in Figure 5.8.

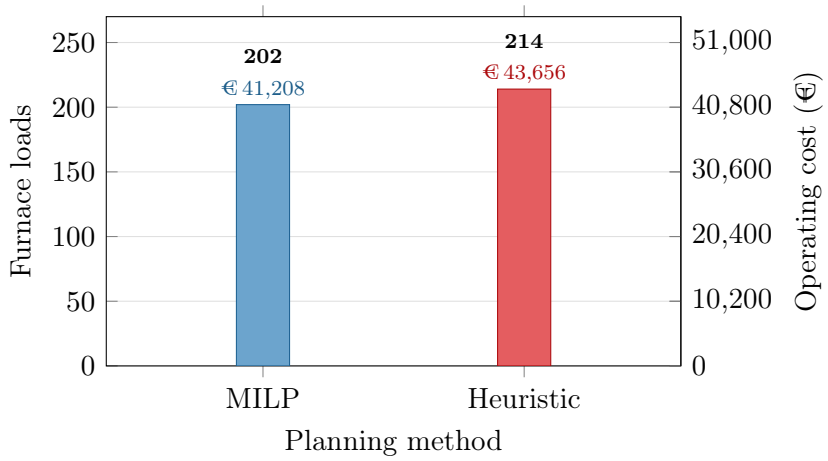


Figure 5.8: **Furnace-load reduction and operating cost.** On the real-world instance, the MILP reduces the plan by 12 loads, corresponding to a direct saving of € 2,448 at € 204 per load.

Over a year of roughly 250 working days at 2 loads per day, the 202-load horizon corresponds to about 101 working days, so consistently applying the MILP rather than the heuristic saves on the order of

$$\Delta C_{\text{year}} \approx 12 \times \frac{500}{202} \times € 204 \approx € 6{,}000 \text{ per year.}$$

This is a lower bound tied to the current furnace and its measured profile. The incoming JVAB301020 furnace (BJVT, specification TS20241128) is rated at 690 kW, about ten times the mean draw of the current unit, with two chambers of 2700 mm usable length each. A larger usable volume packs more radiators per load and needs fewer loads for the same demand, and the value of each saved load rises with the new unit's operating cost. A formal cost model must wait for commissioning (September 2026), but the direction is clear: any per-load saving on the current furnace carries over to the new one, amplified.

Limitations of the cost estimate. The €204 figure rests on a single measurement session and assumes that cycle duration and energy use are independent of the load. In practice heavily filled loads may need longer soaking, and energy use scales non-linearly with load mass. A tighter estimate would require systematic energy logging across cycle types and loading configurations, which is identified as a direction for future data collection.

5.2.6 Discussion

Four conclusions follow from the real-world results.

Both methods are industrially viable. The MILP solves the eleven family subproblems to optimality in under a minute on a standard laptop, and the heuristic completes in about 0.02 seconds. Both are fast enough for production planning, and both satisfy demand for all 20 codes with no constraint violation.

The MILP advantage is measurable and explainable. The 12-load reduction is not a generic benefit of exact optimisation. It comes from the cross-code consolidation in families F10 and F11 analysed in Section 5.2.3, where codes of complementary length pack together but are separated by the heuristic's Phase 1 mono-code partition. The MILP recovers this space; the greedy construction cannot.

The heuristic is the right tool when speed dominates. In 7 of the 11 active families it matches the MILP exactly. The gap appears only where a family holds codes of heterogeneous HEC whose lengths admit cross-code consolidation; homogeneous or single-code families give no gap. A planner can therefore run the heuristic as a fast baseline and reserve the MILP for families whose HEC distribution signals the consolidation opportunity, a condition that can be screened a priori.

The low-saturation loads are unavoidable. Every MILP load below 60% saturation traces to a demand remainder that the per-load capacity cannot divide evenly, not to solver suboptimality. The floor is set by the modular arithmetic of d_i and cap_i; lowering it would require relaxing the single-code-per-level constraint (Section 7.4) or introducing foot interlocking (Section 7.6).

5.3 Benchmark: Systematic Comparison on Synthetic Instances

5.3.1 Scope and Experimental Design

The real-world instance of Section 5.2 and the toy instance of Section 5.1 are single data points. This section turns that evidence into a systematic account: when the exact MILP improves on the constructive heuristic, when the two coincide, and by how much, over a controlled synthetic benchmark.

The benchmark does not study the partner's specific furnace. It studies the abstract problem behind it, the two-stage guillotine level packing introduced in Chapter 1, of which that furnace is one instance. To keep the conclusion a property of the packing problem rather than of a particular framed part, the synthetic instances are generated frame-free ($\delta = 0$): each item is a plain rectangle of length HEC and width SP in

a chamber of length L and width W. The instances then span the full geometric spectrum, from the quasi-one-dimensional regime (one item across the width, two or three levels) to the fully two-dimensional regime (up to four items side by side and up to nine levels per load).

The question that fixes the design. Both solvers share the geometric capacity model and differ only in strategy. The heuristic commits in Phase 1 to dedicated mono-code full loads before it considers any cross-code combination, whereas the MILP assigns all codes globally. A gap therefore arises exactly when a better solution exists in which a code yields part of a load, so that the residual demands of several codes combine more efficiently than the greedy Phase 1 partition allows. The benchmark is built to switch this mechanism on and off.

Choice of instance source. Historical planning horizons were set aside. Consecutive monthly windows share most of their demand and are strongly serially correlated, and the orthogonal level-based model does not reproduce the manual interlocking of the historical records (Section 7.6), which would make a direct comparison structurally asymmetric. Frame-free synthetic instances, generated under the exact geometric assumptions read by both solvers, are the correct basis for a controlled comparison, as is standard in the bin-packing literature [6, 8].

Candidate Drivers and the Selected Factors

Three candidate drivers of the gap were tested one at a time on both solvers, with every MILP solution CBC-certified optimal. The findings below are quantified in Section 5.3.2.

- **Part size and two-dimensional richness**, the number of items side by side (η) and the number of levels per load (m). Moving the geometry from the quasi-1D regime ($\eta = 1$, $m \approx 3$) to the 2D-rich regime ($\eta \geq 2$, m up to 9) leaves the gap unchanged (Section 5.3.2). Part size is not a driver and is excluded as an axis.
- **Demand level**, the number of units requested per code. At fixed homogeneous geometry the gap stays at zero for every demand level, since more demand only multiplies identical loads without creating recombination. Demand alone is not a driver, but it amplifies the gap when structure is present, so it is kept as a secondary factor.
- **Complementary-length structure**, the presence within a family of one long code that fills about half a load together with two short codes whose combined length completes the chamber. This is the structure of the toy instance

(Section 5.1) and the primary driver, producing a robust gap that grows with demand.

The benchmark is therefore a 2×2 factorial over **structure** (homogeneous versus heterogeneous) and **demand** (low versus high). Part size is left out on purpose: Section 5.3.2 shows that it does not move the gap, so varying it would spend half the quadrants on geometrically distinct but statistically indistinguishable regimes.

Instance Classes and Generator Design

A dedicated Python script (`generate_instance.py`) produces instances that share the input format and the geometric parameters of both solvers. All instances are frame-free ($\delta = 0$) in the 2D-rich regime, with an explicit `Delta` column so that both solvers apply $\delta = 0$ exactly. Each instance is a single compatibility family, since the gap mechanism is intra-family.

Structure factor. A heterogeneous instance of n codes is built as $\lfloor n/3 \rfloor$ complementary triples: one long code (HEC $\in [1320, 1360]$ mm, filling about half the chamber on its own) and two short codes (HEC $\in [640, 680]$ mm) whose combined length completes the load. A mixed load of long, short and short is feasible and highly saturating, whereas a mono-code full load of the long code wastes the unused half of its length. The heuristic, committed to mono-code loads in Phase 1, cannot recover this; the MILP does. A homogeneous instance instead draws all n codes from one narrow band (HEC $\in [880, 920]$ mm), so every load holds equal-length levels, no complementary triple can form, and Phase 1 is already optimal.

Demand factor. Per-code demand is drawn from $[4, 8]$ units (low) or $[28, 32]$ units (high). Demand scales the residual volume left by Phase 1 and so amplifies the gap on heterogeneous instances, while leaving homogeneous instances at zero gap by construction.

Factorial design. The four quadrants HOM-LO, HOM-HI, HET-LO and HET-HI each receive 40 independently seeded instances of $n = 9$ codes (a multiple of three, so the complementary-triple construction is exact), for 160 instances in total. A disjoint deterministic seed block per quadrant guarantees reproducibility:

```
python generate_instance.py --factorial --n_per_quadrant 40 --n
   9 --seed 0
```

Benchmark Execution

A second script (`benchmark_runner.py`) runs both solvers on every instance and records the MILP and heuristic load counts, the gap $\Delta = n^{\mathrm{EUR}} - n^{\mathrm{MILP}}$, the mean load saturation of each method, and the CBC optimality status of every family sub-problem, so that any instance not certified optimal is flagged rather than silently averaged in. Each sub-problem runs with a CBC time limit of 180 seconds:

```
python benchmark_runner.py --instances instances_factorial --
    timelimit 180
```

5.3.2 Empirical Justification of the Axis Choice

Before the results, this subsection settles empirically the two claims the design rests on: that complementary structure drives the gap, and that part size and 2D-richness do not. The evidence is a controlled invariance experiment that measures the het/hom contrast in two geometric regimes, quasi-1D ($\eta = 1$, $m \approx 3$) and 2D-rich ($\eta \geq 2$, m up to 9), at fixed structure and demand. It uses 40 independently seeded instances per cell (8 cells, 320 total), each solved to CBC-certified optimality or to the 180-second per-family limit, with anomalous instances documented separately (Section 5.3.6). The generator command is:

```
python benchmark_runner.py --instances instances_invariance --
    invariance --timelimit 180
```

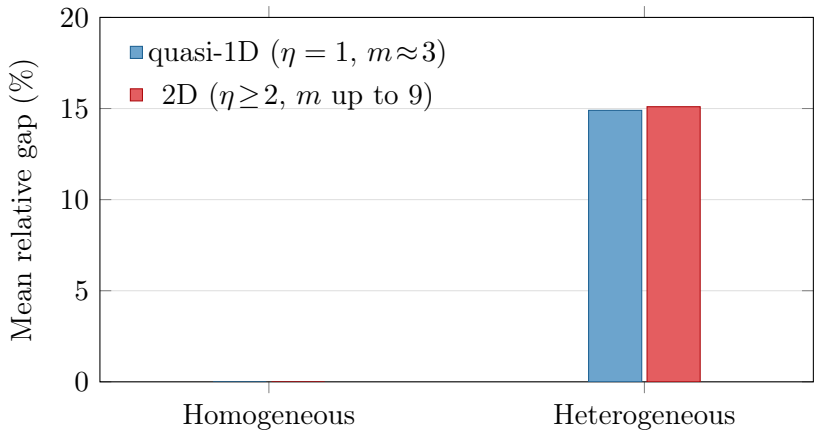


Figure 5.9: **Invariance of the MILP–heuristic gap to the geometric regime.** Homogeneous instances yield a zero gap in both regimes; heterogeneous instances yield 14.9% (quasi-1D) and 15.1% (2D-rich). The geometric regime does not drive the gap: complementary-length structure does.

Figure 5.9 shows the outcome. Structure governs the mean relative gap entirely: homogeneous instances give a zero gap in *both* regimes, and heterogeneous instances

give a positive gap of comparable size in both (14.9% quasi-1D against 15.1% 2D-rich, a difference of 0.2 percentage points). Enlarging the number of side-by-side units and the number of levels per load, that is moving from a quasi-1D to a fully 2D geometry, neither creates a MILP advantage where the complementary structure is absent nor removes one where it is present. The residual difference between the two regimes (0.2 pp) is negligible and carries no consistent sign.

This is the empirical basis for excluding part size as an axis: a factor that does not move the response in a controlled experiment does not earn a place in the design. The same experiment confirms that structure is the driver, since it is the only factor across which the gap moves. The benchmark therefore varies structure and demand and holds the geometric regime at the 2D-rich setting, in the knowledge, not the assumption, that the conclusions carry over to the quasi-1D regime.

5.3.3 Results

Per-Quadrant Summary

Table 5.10 reports the per-quadrant aggregates over the factorial benchmark. Every MILP solution behind the table is CBC-certified optimal (Section 5.3.6 covers the four Het-Hi instances excluded from it), so the gaps are exact.

Table 5.10: Benchmark summary by quadrant (frame-free 2D geometry, $n = 9$ codes, 40 instances per quadrant; Het-Hi reported on $N = 36$ certified instances, see Section 5.3.6). Gap $\Delta = n^{\mathrm{EUR}} - n^{\mathrm{MILP}}$ in furnace loads; $\bar{\rho}$ is the mean load saturation. All MILP solutions certified optimal.

Quadrant		N	Gap Δ (loads)			$\bar{\rho}$ (%)	
Structure	Demand		Mean	±CI95	Median	MILP	Heur.
Homogeneous	Low	40	0.00	0.00	0.0	66.7	66.7
Homogeneous	High	40	0.00	0.00	0.0	67.8	67.8
Heterogeneous	Low	40	2.17	0.29	2.0	79.8	67.1
Heterogeneous	High	36*	8.11	1.29	9.0	80.1	68.6
Total		**156**					

*Four Het-Hi instances excluded: three with negative gap ($\bar{k}_f$ underestimate) and one not certified within the time limit. See Section 5.3.6.

The outcome matches the analytical prediction of Section 5.3.1. On homogeneous instances the heuristic equals the MILP on every instance at both demand levels: the mean gap is exactly zero and the saturation columns coincide. On heterogeneous instances the MILP is strictly better, by a mean of 2.17 loads at low demand (18.5%

relative gap) and 8.11 loads at high demand (14.9% relative gap). The absolute gap grows with demand, as expected, since higher demand leaves more residual units for Phase 1 to strand and more room for the MILP to recombine.

Saturation

The saturation columns isolate the mechanism. On heterogeneous instances the MILP reaches a mean saturation near 80% against roughly 67–68% for the heuristic, a difference of about 12 percentage points, whereas on homogeneous instances the two coincide. The load-count gap and the saturation gap share one origin: the MILP fills each load with a long code and the short residuals that complete it, while the heuristic leaves the long codes in half-empty mono-code loads.

Scatter Plot

Figure 5.10 plots heuristic loads against MILP loads for all 307 valid instances of the invariance benchmark, which spans the same four structural quadrants as the factorial, pooled across geometric regimes.

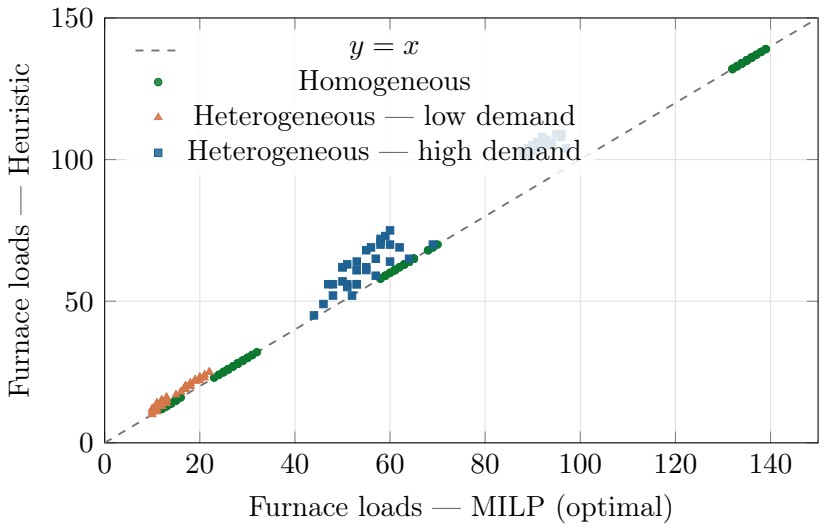


Figure 5.10: **Benchmark scatter plot: MILP vs heuristic.** Furnace loads: MILP (x-axis, optimal or best bound) versus heuristic (y-axis), across 307 valid instances of the invariance benchmark (156 homogeneous, 79 heterogeneous low demand, 72 heterogeneous high demand; 13 anomalous instances excluded — see Section 5.3.6). Homogeneous instances cluster on the diagonal ($y = x$): the heuristic is optimal. Heterogeneous instances lie strictly above, the gap growing with demand level. No valid instance falls below the diagonal.

Every point lies on or above the diagonal $y = x$, so the MILP is never worse than the heuristic. The homogeneous points sit exactly on the diagonal, the zero-gap control;

the heterogeneous points lie strictly above it, their vertical distance growing with instance size, that is with demand. No point falls below the diagonal.

Gap by Quadrant

Figure 5.11 shows the mean relative gap per quadrant from the factorial benchmark.

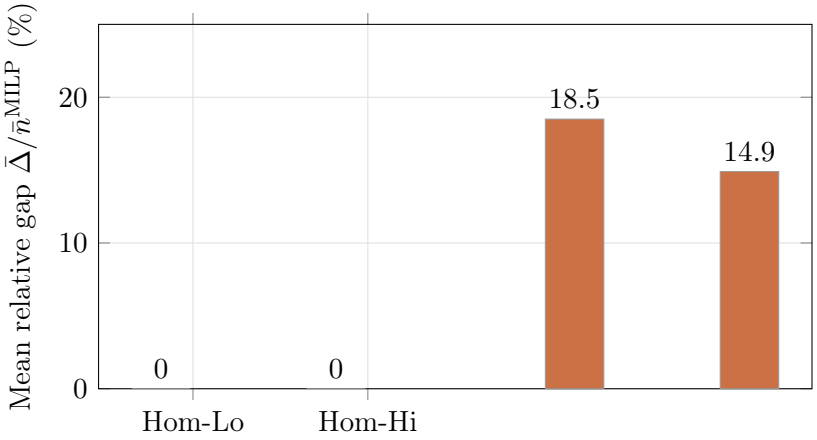


Figure 5.11: **Mean relative gap per quadrant, factorial benchmark.** $n = 9$, $\delta = 0$, 40 instances per quadrant; Het-Hi reported on $N = 36$ certified instances (see Section 5.3.6). Homogeneous quadrants yield zero gap at both demand levels: the heuristic is optimal whenever all codes share the same length class. Heterogeneous quadrants yield a gap of 18.5% (low demand) and 14.9% (high demand): the gap is driven by the complementary-length structure and persists at both demand levels.

The two homogeneous quadrants are flat at zero. The heterogeneous quadrants read 18.5% at low demand and 14.9% at high demand. Demand moves the absolute gap substantially (from 2.17 to 8.11 loads) while the relative gap stays broadly stable: structure switches the gap on, and demand scales it, without changing its relative magnitude.

5.3.4 Gap Distribution on Random Instances

The factorial instances are constructed to carry complementary structure, which raises a fair question: does the MILP advantage disappear on instances that are *not* built to be complementary? The random arm answers it, running 80 instances whose HEC values are drawn uniformly from $[300, 1360]$ mm (the full plausible range for the new furnace), with per-code demand from $[4, 32]$ units and $n = 9$ codes, under a fixed seed:

```
python benchmark_runner.py --random --n_random 80 --n_codes 9
  --random_seed 42 --timelimit 90
```

Three of the 80 produced a negative gap, from the same $\bar{k}_f$ underestimation documented in Section 5.3.6, and are excluded, leaving 77 valid instances.

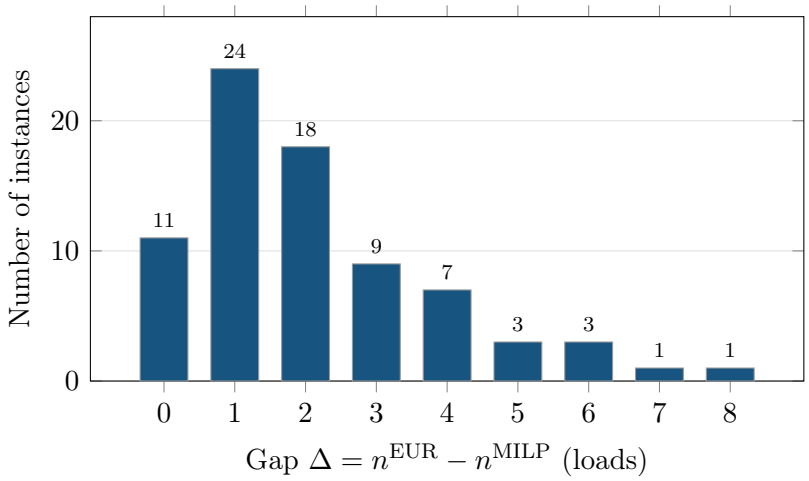


Figure 5.12: **Gap distribution on random instances.** 77 valid instances with HEC values drawn uniformly from $[300, 1360]$ mm ($n = 9$ codes, seed 42; three instances with negative gap excluded — see Section 5.3.6). The gap is positive on 85.7% of instances, with mean 2.12 loads and median 2. The distribution confirms that the MILP advantage is not an artefact of the factorial construction but persists on uniformly random families.

Figure 5.12 shows the gap distribution on those 77 instances. The gap is positive on 85.7% of them, with a mean of 2.12 loads and a median of 2. The distribution is right-skewed: most instances show a small gap of one or two loads, while eight (10%) reach $\Delta \geq 5$.

Two points follow. First, the MILP advantage is not an artefact of the factorial construction, since it persists on random instances where no complementary structure was engineered. The gap appears wherever the demand distribution happens to leave Phase 1 residuals that recombine, which occurs naturally even in uniformly random families. Second, the mean gap on random instances (2.12 loads) is well below the constructed heterogeneous range (2.17 to 8.11 loads), which confirms that engineered complementarity amplifies the gap but is not its only precondition.

5.3.5 A Predictive Index of Complementarity?

The random arm shows that the gap arises wherever Phase 1 residuals admit recombination, but not yet whether that condition can be detected before the solvers run. A natural candidate is an a priori index of how complementary a family's lengths are. Define

$$\chi \;=\; \frac{\left|\{(a,b) : a \neq b,\; 0.90\,(L-\tau) \leq \mathrm{HEC}_a + \mathrm{HEC}_b \leq (L-\tau)\}\right|}{n\,(n-1)}, \qquad (5.1)$$

the fraction of ordered code pairs whose combined length falls in the high-utilisation window used as the $O(n^2)$ screen in Section 5.3.9. By construction χ depends only on the family geometry and needs neither solver. If the gap were a function of complementary structure alone, χ would predict it.

It does not. Figure 5.13 plots the per-instance gap against χ across the 77 valid random instances. The least-squares fit is essentially flat, at $R^2 = 0.01$, so χ explains under one percent of the variance. The qualitative picture agrees: several instances with the largest χ show a zero gap, while the largest gaps occur at low χ.

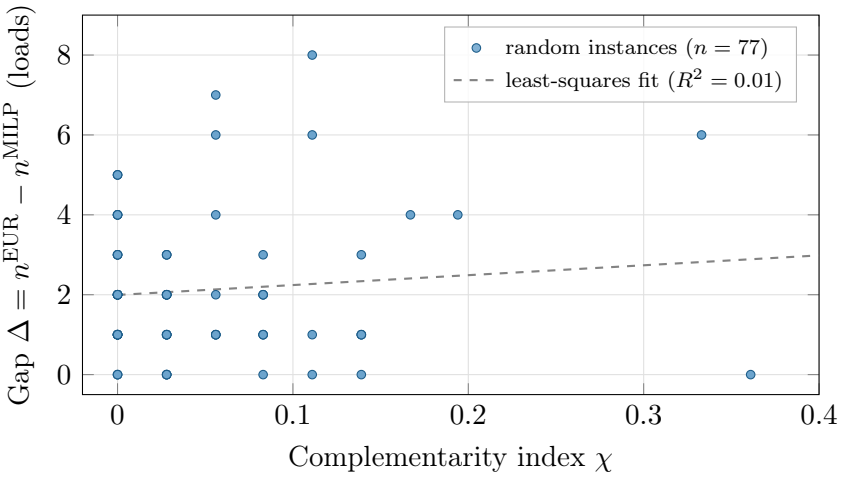


Figure 5.13: **Gap versus complementarity index.** Per-instance gap Δ against the a priori complementarity index χ for the 77 valid random instances of Section 5.3.4 ($n = 9$ codes, seed 42). The least-squares fit is essentially flat ($R^2 = 0.01$): χ, as a purely geometric index computed before any solver is invoked, carries almost no predictive information about the gap. Instances attaining the highest χ include several with zero gap, while the largest gaps occur at low χ.

This is a finding, not a failed measurement. Pairwise length complementarity is abundant in uniformly random families, yet its abundance does not translate into a larger gap. Two refinements give the same verdict: counting complementary *triples* rather than pairs, which aligns the index with the long-plus-fillers mechanism of Section 5.3.7, yields $R^2 = 0.003$, and weighting complementary configurations by the demand of the codes involved never raises R^2 above 0.03.

The failure extends to the weaker task of a binary screen. Even without forecasting how many loads are saved, one might hope χ could flag whether the MILP is worth invoking at all. It cannot. Treating the gap as a two-class label ($\Delta > 0$ versus $\Delta = 0$) and scoring it with χ gives an area under the ROC curve of 0.43, below the 0.50 of a random classifier. The class means explain why: the instances the MILP does help carry a *lower* mean χ (0.045) than those it does not (0.073). Under any rule of the form "invoke the MILP when χ exceeds a cut-off", the screen misses much of

what would benefit, since 27 of the 66 positive instances carry $\chi = 0$, containing no complementary pair at all, yet still show a strictly positive gap. The exact model recovers savings by recombining Phase 1 residuals in ways that do not pass through pairwise complementarity, so an index built on that complementarity is blind to them.

The gap is therefore not a property of length geometry alone. Complementary lengths create the *opportunity* for recombination, but whether it is realised depends on how demand distributes the codes across loads and on whether the greedy Phase 1 commitment happens to forgo a recombination the exact model recovers. These are structure-demand interactions that no scalar computed a priori from lengths can capture. A genuinely predictive index would have to weight complementary configurations by the co-requested quantities that bring them into the same load, at which point it stops being a cheap screen and becomes a surrogate for the packing problem itself.

The predictive signal does not vanish, though; it moves. The factorial benchmark shows that the cleanest separator of the two regimes is not a derived index but the structural character of the family: homogeneous families give a zero gap without exception, heterogeneous ones a gap of 15–20%. This is the rule a planner should apply, and it is readable in $O(n)$ by checking whether a family mixes codes of markedly different length, without computing χ and without running either solver. The section therefore contributes twice: it shows that the magnitude of the gap does not reduce to an a priori index of length complementarity, and it redirects the selection rule to the structural feature that does separate the regimes.

5.3.6 Anomalous Instances

A small number of instances are excluded from the reported statistics under two criteria, both recorded automatically by the runner: a negative gap ($n^{\mathrm{MILP}} > n^{\mathrm{EUR}}$), and failure to certify optimality within the CBC time limit. The exclusions, by arm, are four of the 160 factorial instances (three negative-gap, one not certified), three of the 80 random instances (all negative-gap), and thirteen of the 320 invariance instances, which leaves the 156, 77 and 307 valid instances used in the tables and figures above.

Every negative-gap case has the same root cause. The upper bound $\bar{k}_f$ on the number of candidate loads is a conservative estimate derived from the total demand and the minimum feasible levels per load. On instances with very high demand and heterogeneous HEC values, this bound occasionally underestimates the loads actually needed. With the feasible search space thus restricted, CBC may settle on a solution that is optimal within the truncated space but uses more loads than the heuristic, which the bound does not constrain. This is not a defect in the model: the objective

and constraints are correct, and the bound is deliberately conservative to keep the number of binary variables tractable.

The bound $\bar{k}_f$ was deliberately left uncorrected in the solver code, so that the thesis formulation and the implementation stay mutually consistent. The qualitative conclusions of the benchmark do not depend on these exclusions.

5.3.7 Mechanistic Interpretation

At scale, and with certified-optimal solutions, the benchmark confirms the mechanism read off the toy instance (Section 5.1). The gap is the deterministic consequence of one structural property, complementary code lengths within a family, and not a diffuse statistical effect. Four consequences follow.

The heuristic is exact on the homogeneous regime. Without complementary structure the heuristic returns the optimal load count on every instance tested, at every demand level and in both geometric regimes. Here it is not a good approximation but provably optimal, and its sub-second runtime makes it the natural choice.

The MILP recovers a substantial margin on the heterogeneous regime. With complementary structure the MILP removes 18–19% of the heuristic's loads at low demand and 15% at high demand, the absolute saving rising with demand (from a mean of 2.17 to 8.11 loads per instance) as more residual units are stranded and more combinations open up. This is where the exact model earns its extra computational cost.

The driver is generic, not geometry-specific. The complementary-length mechanism is a property of how code lengths relate to the chamber length, shown invariant to the number of side-by-side units and levels per load (Section 5.3.2: 14.9% versus 15.1%, difference 0.2 pp). The benchmark therefore describes a general property of two-stage level-based packing, instantiated on a particular geometry but not tied to it.

The gap persists on random instances. The random arm confirms that the advantage is not an artefact of the factorial construction: 85.7% of uniformly random instances show a positive gap, at a mean of 2.12 loads. The factorial benchmark fixes the conditions under which the gap is large and predictable; the random arm shows it survives even when those conditions are not engineered in.

5.3.8 Computation Time

The heuristic finishes well under one second on every instance. The MILP runtime depends strongly on both factors of the design, as Table 5.11 shows.

Table 5.11: CBC solver time per structural quadrant over the invariance benchmark ($n = 9$ codes, 80 instances per quadrant, time limit 180 s per family sub-problem). "At limit" counts instances that exhausted the time budget; these include instances flagged as not certified optimal.

Quadrant		MILP time (s)			At limit
Structure	Demand	Mean	Median	Max	$n/80$ (%)
Homogeneous	Low	1	1	1	0 (0%)
Homogeneous	High	22	5	181	4 (5%)
Heterogeneous	Low	74	6	541	27 (34%)
Heterogeneous	High	156	180	182	61 (76%)

Four points follow from the table.

Structure is the primary driver of runtime. Homogeneous instances solve quickly at both demand levels: none reaches the time limit at low demand, and only 5% do at high demand. Heterogeneous instances are qualitatively harder, with 76% of Het-Hi instances exhausting the 180-second budget. The explanation mirrors the gap result: on homogeneous instances Phase 1 is already optimal and the MILP confirms it quickly, whereas on heterogeneous instances the solver must search a much larger space to verify that cross-code consolidation is fully exploited.

Demand amplifies runtime within the heterogeneous quadrant. At low demand the heterogeneous runtime is bimodal: the median is only 6 seconds, for instances where the optimum is found quickly, yet 34% reach the limit. At high demand the distribution collapses towards the limit, with the median itself at 180 seconds and 76% of instances exhausting the budget.

The heuristic is always fast. Across all 320 instances the heuristic completes in under one millisecond. Its speed-up over the MILP ranges from three orders of magnitude on easy homogeneous instances to five or more on the hardest heterogeneous ones.

Practical implication. For the real-world instance of Chapter 5.2 ($n = 20$ codes, 11 families) the MILP finishes in under a minute. The invariance benchmark shows

that this speed is specific to instances without strong complementary structure. A planner facing a horizon with codes of complementary HEC lengths should expect the MILP to approach the per-family time limit on at least some families. The heuristic stays the tool of choice whenever a solution is needed in seconds; the MILP is reserved for horizons where the screen of Section 5.3.9 detects the structural condition under which it delivers a measurable quality advantage.

5.3.9 Discussion

A clean separation of regimes. The two methods are provably equivalent on homogeneous instances and differ by a substantial margin on heterogeneous ones, with no grey zone in the factor that matters: structure switches the gap on or off, demand scales it once on, and geometry leaves it unchanged. This is sharper and more defensible than an undifferentiated average gap over a mixed set, which would conflate the regimes and report a mean describing neither.

A practical selection rule. The driver is structural, and the benchmark identifies which structural feature governs it. The separation runs along the homogeneous–heterogeneous axis: a family whose codes share a common length admits no recombination and is solved optimally by the heuristic, whereas a family mixing codes of markedly different length leaves Phase 1 residuals the exact model can consolidate. A planner can screen each horizon in $O(n)$ by checking whether the family is length-homogeneous, invoking the MILP only on heterogeneous families and using the heuristic elsewhere. What does not serve as this screen is the a priori pairwise index χ of Section 5.3.5, which neither predicts the magnitude of the gap nor reliably classifies which instances show one, because the exact model recovers savings through residual recombinations that pairwise complementarity does not capture. The dependable signal is the structural character of the family, not a derived index of its length geometry. This hybrid strategy captures the full quality advantage where it exists at no cost where it does not.

A direction for heuristic improvement. The gap on heterogeneous instances is due entirely to the irrevocable mono-code commitment of Phase 1. A Phase 1 variant that delays this commitment when complementary codes are detected, or a post-Phase 2 local search that merges under-saturated loads within a family, would close most of the gap while preserving the heuristic's speed. Both directions are taken up in Chapter 7.

Chapter 6

Conclusions

This thesis set out to determine how the radiators of a vacuum brazing furnace should be combined into loads, and whether exact optimisation earns its cost over the greedy planning that current practice would adopt. The furnace-loading problem was formalised as a level-based two-stage two-dimensional bin packing problem, solved exactly by a per-family mixed-integer linear programme and measured against a two-phase constructive heuristic built on the same geometric model. The work yields three main results, and a practical rule that follows from the first two.

The first result is a clean separation of regimes. On length-homogeneous families the constructive heuristic is provably optimal and runs in milliseconds; on heterogeneous families the exact MILP recovers a 15–20% reduction in load count, traceable to a single mechanism, the cross-code consolidation that the heuristic's irrevocable first-phase commitment forecloses. The same mechanism operates on the real order book, where only a minority of families are heterogeneous: there the MILP removes 12 of the heuristic's 214 loads, with a directly estimated operating saving that the incoming higher-power furnace will only amplify. The separation is not a statistical average over a mixed population but a structural law. It held invariant across the geometric regime, equally in the quasi-one-dimensional and fully two-dimensional settings, and it persisted on uniformly random families where no complementary structure was engineered in.

The second result marks how far this characterisation can be pushed. An attempt to compress the structural condition into an a priori index of length complementarity failed, and the failure is itself informative. No geometric scalar computed before the solvers, whether defined over code pairs or triples and whether or not weighted by demand, predicts the magnitude of the gap, with R^2 below 0.04 in every variant tested, and none reliably classifies which instances exhibit a gap at all. The advantage the exact model recovers is therefore not a property of length geometry in isolation but of

its interaction with the demand distribution, which cannot be precomputed cheaply without effectively re-solving the packing problem. What does separate the regimes is the structural homogeneity of the family, readable directly in $O(n)$.

These two results combine into a practical rule. A planner need not run the exact model on every family. It is enough to screen each family in $O(n)$ by whether it mixes codes of markedly different length, applying the instantaneous heuristic to the homogeneous families, on which it is optimal, and reserving the MILP for the heterogeneous ones, on which it earns its additional computation time. This hybrid strategy captures the full quality advantage where it exists at no cost where it does not, and it is robust precisely because it rests on a structural feature rather than a fragile numerical predictor.

The third result concerns the modelling scope. The level-based formulation deliberately excludes the denser arrangements that operators exploit in practice, foot interlocking and fishbone placement chief among them, and it leaves a strip of unusable space at the end of every chamber that no single-code level can fill. Had these choices been too restrictive, the loads would have settled at low saturation, which would have signalled that the excluded arrangements carried density the model had discarded. The saturations attained show the opposite: the chamber is filled almost completely on the great majority of loads, so the orthogonal level-based scope captures nearly all the density physically available. This does not make the excluded arrangements worthless, but it places them as refinements that trim the result at the margin rather than reshape it, and it justifies in retrospect the scope fixed in Section 1.3 as a deliberate and well-calibrated decision rather than a simplification of convenience.

The work carries limitations, stated openly rather than concealed. The candidate-load bound $\bar{k}_f$ is deliberately conservative, producing a small number of documented anomalous instances that were excluded rather than corrected, so that the formulation and its implementation remain mutually consistent. The real-world validation rests on a single order book and one furnace, and the cost estimate derives from a single measurement session. The orthogonal level-based model also sets aside the foot-interlocking behaviour that experienced operators exploit, a choice that keeps the formulation faithful to a well-defined packing class at the price of underestimating attainable density. Each of these is a deliberate scoping decision, and each marks a direction in which the present results can be extended. Those directions are taken up in the chapter that follows.

Chapter 7

Future Developments

7.1 Extension to Two Chambers

The furnace has two separate chambers. A two-chamber formulation would add an index for the chamber and require deciding, for each load, which codes and configurations to assign to each. The two chambers share the same thermal programme, so they can run different families within one physical furnace load, while their loading configurations may differ. This extension roughly doubles the number of z variables but does not increase the number of candidate loads.

7.2 Extension to Two Loading Tiers

The new furnace supports loading on two stacked tiers. A two-tier formulation would add a tier index and allow the effective capacity of each load to double. The added complexity lies in the height constraints and in allowing the two tiers to hold different loading configurations.

7.3 Heuristic Improvement: Local Search and Multistart

The benchmark shows that the heuristic's Phase 2 FFD strategy falls well below the MILP optimum on heterogeneous instances. The root cause is the irrevocable mono-code commitment of Phase 1: once full single-code loads are recorded, cross-code consolidation is permanently foreclosed. Two complementary directions can address this.

Post-Phase-2 local search. Once Phase 2 completes, a local search can try to reduce the load count within each family. The natural move is a pairwise merge: scan all pairs of loads in the same family, test whether their combined levels fit in one chamber, and merge them when they do, recording the freed load as a saving. The move is applied until no improving merge remains. Since the feasibility test is $O(\bar{j}_f)$ per pair and the number of loads per family is small in practice, the overhead is negligible against Phase 1 and Phase 2.

Phase 1 variant with complementarity lookahead. A more targeted change is to suppress the Phase 1 full-load commitment for codes whose length leaves a recoverable residual. Rather than filling all $\lfloor d_i/\text{cap}_i \rfloor$ mono-code loads for such a code, the variant holds back a controlled fraction of its demand for Phase 2 to combine. This is an internal trigger for where to defer demand, not a predictor of the final gap, so it is not bound by the negative result of Section 5.3.5: the pairwise complementarity condition fails to predict whether an instance exhibits a gap, but it can still flag individual codes worth deferring. Whether the deferral actually lowers the load count must be settled empirically, by benchmarking the variant against the baseline, since Section 5.3.5 shows that the savings the MILP recovers do not in general pass through pairwise length complementarity. The lookahead is one candidate trigger among several, and its value is an open question rather than a settled improvement.

Both variants keep the heuristic's sub-second runtime and its exact optimality on homogeneous instances, where neither merge nor lookahead is triggered. Benchmarking them against the baseline heuristic and the MILP is a direct next step.

7.4 Multi-Code-per-Level Constraint

Constraint (3) requires each level to hold a single radiator code. When two codes in the same family have similar lengths, they could instead share a level in adjacent positions across the width. Relaxing the constraint would require modelling item placement within each level explicitly, which raises the combinatorial complexity but could improve chamber utilisation appreciably, especially for families with many codes of similar but not identical HEC.

7.5 Non-Guillotine Loading Patterns

The current model restricts loading to a two-stage guillotine structure: partition the chamber longitudinally into levels, then fill each level across the width. Allowing non-guillotine patterns would permit arbitrary rectangular placement and capture

configurations such as alternating short and long codes on the same transverse row, an arrangement operators sometimes use with similarly sized frames. It would require explicit coordinate variables for each item and a substantial increase in model complexity.

7.6 Incorporation of Foot Interlocking

The most significant element excluded from the current model is the interlocking of frame feet, which lets operators reach loading densities beyond those the rectangular packing model predicts.

Radiator frames sit on supporting feet mounted perpendicular to the frame plane. When two frames occupy adjacent longitudinal positions, their feet may overlap partly across the width, reducing the effective longitudinal pitch between consecutive radiators. For codes loaded two per chamber in the old furnace, the operator uses a mirrored, staggered configuration: two radiators face each other with interlocked feet and take up less total length than two independent rectangular footprints would. In the new furnace, whose chamber is roughly twice as long, two such mirrored pairs may leave enough residual central space to insert a fifth radiator that no rectangular packing assumption would admit.

Formalising this would take two steps. First, systematic collection of foot dimensions, length and lateral offset, for each radiator code in production. Second, a geometric compatibility rule between adjacent frames: given the foot profiles of two codes i and i', the rule would decide whether interlocking is feasible and, if so, the longitudinal pitch reduction $\Delta\tau_{ii'}$ it yields. This reduction would replace the fixed longitudinal gap τ with a code-pair-dependent value $\tau_{ii'} = \tau - \Delta\tau_{ii'}$, modifying constraint (2) of the MILP and the capacity formula of the heuristic directly.

7.7 Rolling Time Horizon

The current model treats demand as deterministic and aggregated over a fixed planning horizon. A rolling-horizon model would plan over a sliding window of T months, updating the solution as new orders arrive and completed ones leave. This requires managing carry-overs, the units not produced in the current month that accumulate into the next, and defining priority policies for codes with more urgent demand.

Appendix A

Python Implementation

This appendix provides the complete Python scripts described in the thesis. All scripts require Python 3.10 or later and the packages listed below, installable via:

```
pip install pulp pandas openpyxl matplotlib
```

A.1 MILP Model: `MILP.py`

The script reads `orders.xlsx`, solves one MILP sub-problem per compatibility family, and writes `load_results.xlsx`.

```
python MILP.py --excel orders.xlsx
```

The key functional blocks are described in Chapter 3. The complete source code is available in the project repository alongside this thesis.

A.2 Heuristic: `Heuristic.py`

The script reads the same `orders.xlsx`, applies the two-phase constructive heuristic described in Chapter 4, and writes `Heuristic_results.xlsx`.

```
python Heuristic.py --excel orders.xlsx
```

The heuristic derives load capacity exclusively from the chamber geometry and the orthogonal loading model, using the same parameters as the MILP. The result is therefore directly comparable with the MILP output, as discussed in Section 5.2.3.

A.3 Instance Generator: `generate_instance.py`

Generates synthetic instances compatible with both solvers. In batch mode, produces N files in the `instances/` subdirectory, each with a different random seed.

```
python generate_instance.py --n 10 --batch 50 --seed 0
```

Parameters:

- `-n`: number of radiator codes per instance (default: 10).
- `-batch`: number of instances to generate (default: 1).
- `-seed`: base random seed for reproducibility.
- `-dmin`, `-dmax`: demand range per code (default: 2–80).

A.4 Benchmark Runner: `benchmark_runner.py`

Runs both solvers on all instances in `instances/` and produces `benchmark_results.xlsx` and `benchmark_scatter.png`.

```
python benchmark_runner.py --n_inst 50 --timelimit 120
```

Parameters:

- `-instances`: path to the instances folder (default: `./instances/`).
- `-n_inst`: number of instances to process (default: 50).
- `-timelimit`: CBC time limit per family sub-problem in seconds (default: 120).

Bibliography

[1] Edward G. Coffman et al. "Performance bounds for level-oriented two-dimensional packing algorithms". In: *SIAM Journal on Computing* 9.4 (1980), pp. 808–826.

[2] COIN-OR Foundation. *COIN-OR Branch and Cut Solver (CBC)*. 2025. URL: https://github.com/coin-or/Cbc (visited on 05/01/2026).

[3] Harald Dyckhoff. "A typology of cutting and packing problems". In: *European Journal of Operational Research* 44.2 (1990), pp. 145–159.

[4] Paul C. Gilmore and Ralph E. Gomory. "Multistage cutting stock problems of two and more dimensions". In: *Operations Research* 13.1 (1965), pp. 94–120.

[5] David S. Johnson et al. "Worst-case performance bounds for simple one-dimensional packing algorithms". In: *SIAM Journal on Computing* 3.4 (1974), pp. 299–325.

[6] Andrea Lodi, Silvano Martello, and Daniele Vigo. "Two-dimensional packing problems: A survey". In: *European Journal of Operational Research* 141.2 (2002), pp. 241–252.

[7] François Margot. "Symmetry in Integer Linear Programming". In: *50 Years of Integer Programming 1958–2008*. Ed. by Michael Jünger et al. Berlin: Springer, 2010, pp. 647–686.

[8] Silvano Martello, David Pisinger, and Daniele Vigo. "The Three-Dimensional Bin Packing Problem". In: *Operations Research* 48.2 (2000), pp. 256–267. DOI: 10.1287/opre.48.2.256.12384.

[9] Stuart Mitchell, Michael O'Sullivan, and Iain Dunning. *PuLP: A Linear Programming Toolkit for Python*. Version used: 2.x, accessed May 2026. 2011. URL: https://coin-or.github.io/pulp/ (visited on 05/01/2026).

[10] Gerhard Wäscher, Heike Haußner, and Holger Schumann. "An improved typology of cutting and packing problems". In: *European Journal of Operational Research* 183.3 (2007), pp. 1109–1130.

[11] Laurence A. Wolsey. *Integer Programming*. Wiley, 1998.

Sitography

- Tesio Cooling Systems S.p.A. official website: https://www.tesiocooling.com
- Python official documentation: https://www.python.org
- Pandas documentation: https://pandas.pydata.org
- OpenPyXL documentation: https://openpyxl.readthedocs.io
- Matplotlib documentation: https://matplotlib.org
- Vacuum brazing technical resources: https://www.vacaero.com
- Nadcap heat treating resources: https://p-r-i.org/nadcap/