

UNIVERSITÀ
DI PAVIA

UNIVERSITY OF PAVIA
FACULTY OF ENGINEERING
DEPARTMENT OF INDUSTRIAL ENGINEERING

MASTER'S DEGREE IN INDUSTRIAL AUTOMATION
ENGINEERING

MASTER THESIS

OPTIMIZATION OF ROBOTIC DOSING SYSTEM: ADVANCED
ILP SCHEDULING FOR BOTTLENECK REDUCTION AND
BUFFER MANAGEMENT

OTTIMIZZAZIONE DI UN SISTEMA DI DOSAGGIO ROBOTICO:
SCHEDULAZIONE ILP AVANZATA PER RIDUZIONE DI COLLI DI
BOTTIGLIA E GESTIONE DEI BUFFER

Supervisor:

Prof. Chiara Toffanin

Candidate:

Zamproni Eleonora

569267

A.Y. 2025/2026

Abstract

This thesis addresses the optimization of an industrial robotic dosing system. Thanks to *Logic S.r.l.*, the case studio relies on real technical specifications and customer requirements. The physical plant is composed of (5) dosing machines and a limited number of parking spots (6). The process consists of several recipes, that must be dynamically routed to different machines to receive their ingredients, hence they are divided into sub-recipes. The primary objective of the developed algorithm is to minimize the overall completion time, while respecting physical constraints, like limited parking spots capacity and mutual exclusivity within barrels and machines, and time constraints for the completion of individual sub-recipes.

To manage this scheduling problem, Integer Linear Programming (ILP) was utilized to develop the mathematical model to represent the industrial environment. The ILP formulation relies on a set of optimization variables, representing the solver's decisions, and on the allocation matrix and capacity vector, which allow to constrain the feasible solutions region. The implemented model successfully optimizes the scheduling, providing a minimization of the completion time and a maximization of the machines utilization.

Contents

Abstract	i
List of Tables	vii
List of Figures	ix
1 Introduction	1
2 Theoretical background	5
2.1 Introduction to Job Shop Scheduling Problem	5
2.2 Computational complexity	6
2.3 Limited parking spots and resource contention	7
2.4 Time management	8
2.4.1 Discrete-time	8
2.4.2 Continuous-time	9
2.5 Solution methods	9
2.5.1 Choice of hybrid approach	10
2.6 Automation and dosage process application	11
3 System Description	13
3.1 Problem statement	13
3.2 Physical Layout	14
3.2.1 Porposed configurations	14
3.2.2 Company's choice	16
4 ILP Mathematical formulation	17
4.1 ILP background and assumptions	17

4.2	Solution methodology	18
5	Logic S.r.l. ILP mathematical formulation	21
5.1	Introduction	21
5.2	ILP mathematical formulation	22
5.3	Optimization variables	24
5.4	Cost function	24
5.5	Constraints	25
5.6	New multi-objective cost function	27
5.7	Illustrative example of matrix composition	29
6	Implementation	35
6.1	System setup	35
6.1.1	Data Acquisition and Pre-processing	37
6.1.2	Industrial specifications and Model assumptions	37
6.1.3	Data organization	38
6.1.4	Recipes generation	40
6.1.5	Database generation	41
6.1.6	Temporal flow	42
6.2	Matrices generation	43
6.2.1	Matrix A	43
6.2.2	Vector b	46
6.2.3	Upper bound and Lower bound	47
6.3	Objective function	48
6.3.1	Movement tracking	48
6.3.2	Objective function versions	49
6.4	Solver	50
6.5	Database population	50
6.5.1	System state update	51
7	Simulations and Results	53
7.1	Proof of concept	53
7.2	Extended recipes list	59
7.3	Robustness validation with an alternative dataset	61
7.4	Robustness validation with a different parking spot capacity	62

CONTENTS

v

8 Conclusions and Future work

65

Bibliography

69

List of Tables

3.1	Individual dimensions of system components.	14
5.1	Simplified scenario for matrix population.	29
6.1	First lines of the ingredient information table.	39
6.2	Subdivision of Barrel 1.	42
7.1	Performance indicators for the 10 recipes scenario across the four objective function versions.	58
7.2	Performance indicators for the 30 recipes scenario across the four objective function versions.	60
7.3	Performance indicators with a different set of recipes to vali- date model robustness.	62
7.4	Performance parameters comparison for a 20 recipes subset under varying parking slot capacities.	63

List of Figures

3.1	System Straight line Layout.	14
3.2	System Double line Layout.	15
3.3	System U-Layout.	15
6.1	Flowchart of the optimization loop.	36
7.1	10 recipes scheduling version V0.	55
7.2	Saturated buffer in 10 recipes scenario, version V0.	56
7.3	10 recipes scheduling version V1.	56
7.4	10 recipes scheduling version V2.	57
7.5	10 recipes scheduling version V3.	58

Chapter 1

Introduction

In the new era of Industry 4.0, decisions regarding productivity can no longer be left to human intuition. The overall efficiency of the manufacturing line depends not only on the processing speed of machines as individual tools, but also on the tight synchronization of parallel operations, taking into account collaboration among different workstations. In the current industrial environment, this synchronization is fundamental to eliminating waste, especially in automated material handling, where different strategies can be applied.

When analyzing multiple jobs competing for limited resources, bottlenecks are an ever-present risk. Without a mathematical model to support the decision-making process, it is left to rely on empirical approaches that don't capture the big picture of the plant. As a result it becomes frequent to deal with useless bottlenecks, longer completion times, suboptimal utilization of the buffer zone, and in general, unwanted long makespan and elevated costs.

To address all of these critical points, scheduling optimization has become a cornerstone for researchers, leading to the formulation and wide study of the *Job Shop Scheduling Problem* (JSSP).

The JSSP has been one of the most thriving areas of operational research and management science, and for its broad application backgrounds, it has been extensively studied by academics and researchers [1]. Several tasks, composed of operations, must be processed in a certain order on some ma-

chines, under predetermined constraints, usually due to space or technology limits. The aim is to complete all jobs minimizing the overall completion time [1].

Different formulations of the JSSP exist, as each industrial context requires adaptations of the general model due to space constraints, machine configuration, or job characteristics. Over the decades some solution methods have been developed in the literature, from exact approaches, such as Linear programming, to heuristic algorithms or a combination of both.

In this thesis, the scheduling of an industrial robotic dosing system has been developed. The company *Logic S.r.l* presented an industrial problem: the commissioning of a system composed of 5 machines and 6 intermediate buffers, 80 recipes to be completed in a working day, around 600 ingredients distributed according to frequency and dosing duration in each machine, a robotic arm to transfer barrels, and the aim of minimizing the makespan.

Here, an Integer Linear Programming (ILP) approach is applied to address this static and deterministic JSSP.

The structure of the thesis is as follows:

- *Chapter 2* formally classifies Job Shop Scheduling Problems, exact and heuristic methods, and the mathematical background of ILP;
- *Chapter 3* describes the system proposed by *Logic S.r.l.* and analyses its critical aspects;
- *Chapter 4* presents the mathematical formulation of decision variables, constraints, objective function, resources, jobs, and operations;
- *Chapter 5* shows the implementation in MATLAB software and explains the solver;
- *Chapter 6* collects the fundamental simulations of a subset of recipes in different operative conditions to prove the functioning of the optimization and compares the results;
- *Chapter 7* reports a simulation using a more realistic dataset and comparison with previous results;

- *Chapter 8* concludes the project and presents future work ideas.

Chapter 2

Theoretical background

The first chapter describes the theory related to the problem object of this thesis, including a collection of characteristics of the scheduling model regarding advantages, disadvantages, variations, and how it has been addressed.

2.1 Introduction to Job Shop Scheduling Problem

The production planning problem has been known for decades and is commonly referred to as the *JSSP*, Job Shop Scheduling Problem, a term that first appeared in the 1950s [2]. In the classical *JSSP* a set of jobs must be processed on a set of machines, the resources. Each job is composed of a sequence of operations, with finite processing time and a machine to be elaborated on [1] and the schedule represents the occupancy of machines over time [2]. The scheduling combines operations across machines with the primary objective of minimizing the makespan. Other entities in the production activities of manufacturing systems can be involved such as operators, transportation devices, industrial robots, auxiliary tools, buffers, warehouses, and containers [1]. Specifically, this thesis will address buffer and transportation management.

In general, there exist other two types of Shop Scheduling Problem: the *Flow Shop* and the *Open Shop*. The first one characterizes for having a predefined machine process and it is equal for all jobs, while the second one

does not have a sequence for jobs but they must visit every machine to be completed, and it is quite rare in the real world. Despite some peculiarities, the aim is to find the optimal combination of jobs to minimize some parameters, most commonly the completion time [2].

The *JSSP* formulation involves assumptions or constraints representing the gap between mathematical model and physical reality. In this project, machines are available at time zero and either breakdowns or re-processing of unqualified jobs are considered; there is no fixed priority among jobs and the sequence of operations only varies due to optimality reasons; a machine can only perform an operation at a time and an operation is assigned a machine only once during its completion [1].

A further division in *JSSP* is between dynamic and static models. The former one consists of relaxing assumptions on release time of jobs and managing unexpected events, while the latter regards knowing beforehand all involved attributes and environment information, which remain unchanged during the scheduling process [1]. Having listed the assumptions on the current problem, it reveals to be a static model, therefore any change in the original conditions is not taken into consideration once the optimization has begun.

2.2 Computational complexity

The scheduling problem generally consists of these key components: production recipes, available processing and storage equipment, intermediate storage policy, production requirements, specifications on resources, and a time horizon. The goal is to determine the best schedule under these limitations. JSSPs are inherently combinatorial because of the numerous discrete decisions involved, so they belong to the set of NP-complete problems, meaning that the solution time grows exponentially as the problem size increases [3].

In computational complexity theory, the class NP encompasses decision problems for which, if a solution exists and the answer is "yes", it can be verified in polynomial time. NP-hard problems are Non-Deterministic Polynomial-time Hard problems, and they are at least as difficult as the

hardest problem in NP, meaning they could not generally have a solution verifiable in polynomial time. The difference between NP-complete and NP-hard is that the former is a decision problem, while the latter is tied to optimization. NP-complete problems are only a subset of NP-hard ones, coming from the intersection with NP problems. In general, there is no known polynomial algorithm that solves NP-hard problems. Some real-world categories of NP-hard problems are: combinatorial optimization, scheduling, decision and search, bio informatics, operations research, logistics and transportation, planning, manufacturing and production [4]. Consequently, the problem tackled in this thesis falls into this classification.

Several factors contributing to the complexity of finding an optimal solution are:

- Exponential growth: the scheduling problem is combinatorial; as mentioned, the number of possible solutions can grow exponentially with the size of the input;
- Interdependence between variables: if changing one variable affects others, a complex network of interactions arises, complicating the research of a solution;
- Incompleteness of information, human factor, limitation of resources, and multiple objectives and constraints [4].

2.3 Limited parking spots and resource contention

In classical job shops, the capacity of intermediate parking storage is assumed to be infinite, which means parking spot constraints are neglected. However, this assumption makes the model inapplicable to real industrial systems. For example, in the food industry, the canning operation must come immediately after the cooking operation to ensure freshness, so the *no – wait* constraint occurs. In the chemical industry, the *no – buffer/blocking* constraint happens when a partially processed product must be temporarily kept in the processing machine because of safety reasons [5]. Note that in this thesis

the terms buffer, parking spots, and parking slots are intended as the same entity.

The blocking constraint refers to storage capacity constraints, meaning that when a job completed by a machine must remain on this machine as long as the next machine that will process it is busy with another job. The no-wait constraint occurs when a job must be processed continuously among machines without interruption [6].

In industrial realities, the *limited – buffer* constraint is fundamental to alleviating abrupt changes in fabrication lines since an intermediate ”parking spot” allows started products to rest while waiting for the next equipment unit [5].

In this project, machines are separated by buffers to allow the storage of barrels among operations. These two constraints are not intrinsically a characteristic of the problem analyzed, but one of them may happen: when an operation finishes in a machine, but buffers are saturated, a barrel must occupy a machine until there is a free park (this will be a further discussed topic). Conversely, the no-wait constraint is not an interesting objective for the purpose of this study, since there are no peculiarities regarding the nature of the material dosed.

2.4 Time management

When formulating a mathematical model for any process scheduling problem, the major issue to face is time management. It is possible to classify all existing formulations in two distinct categories: discrete-time and continuous-time models [3].

2.4.1 Discrete-time

Discrete-time models follow the time-discretization approach: the time horizon is divided into time intervals with uniform durations, to which the decisions to be made are associated. This means working with an approximation of time; in fact, a good strategy to obtain a reasonable degree of accuracy is selecting a short length of time for the duration of time intervals. In fact,

if the time horizon is long or the time interval length is small, the number of discretized intervals can be very large, leading to an oversized combinatorial problem that is computationally expensive to solve. To overcome this drawback, continuous-time models have emerged [3].

2.4.2 Continuous-time

The continuous-time approach introduces the concept of variable time intervals: they become continuous variables that can potentially assume any value in the time horizon, hence changes in the activities can take place at any time, modeling time more accurately. On one hand, problems can decrease in size and require less computational effort, on the other, it is more challenging to model the scheduling due to the variable nature of events [3].

Due to the complexity of this kind of modeling, the possibility of applying the discrete-time approach has been considered and chose because the Duration of Planning (*DoP*) and the choice of the length of the time interval as 1 minute allowed to reach the optimal solution anyway.

2.5 Solution methods

The difficulty yet importance of solving scheduling problems, due to their combinatorial and NP-hard nature, has lead to a large number of solution techniques which can be either exact or approximate [7].

Exact algorithms guarantee to find an optimal solution and prove its optimality or that no feasible solution exists, when given enough computation time [7]. Some exact methods are branch-and-bound, branch-and-cut, integer programming and linear programming [8].

The most successful method has been *Integer Programming*. Advantages are that, if the algorithm succeeds, the optimality can be proven; it is possible to know information about lower and upper bounds even if the algorithm is stopped before end; it is mathematically guaranteed to eliminate unpromising regions within the solution space, where it is sure that optimal points cannot exist [7].

Although the exactness, there are disadvantages such as: the size of the

solvable instance is rather limited, the variance of the computation times may be large when applied to instances of the same size, the memory consumption could lead to the abortion of the program and finally, it is usually difficult to add details [7].

For this reason heuristic and metaheuristic methods like genetic algorithms are factored in, since they are more versatile and adaptive. They are iterative approaches that explore solution spaces using patterns from mathematics, nature and social behavior, consequently being reasonable for real-world optimization problems where the objective is to obtain a good solution in limited amount of time [9].

2.5.1 Choice of hybrid approach

Another possibility is to combine the two approaches in order to exploit the speed of the metaheuristic and boost its performance with mathematical programming techniques [7].

In this thesis the fundamental approach is an exact method: *Integer Linear Programming*. Nevertheless, during the formulation of the solution algorithm, it was decided to add the possibility of enriching the cost function with two heuristic-based approaches: *Shortest Processing Time (SPT)* and *Most Work Remaining (MWKR)*. The former one is the most commonly used for job shop scheduling and it is based on giving priority to short operations on machines, minimizing mean flow time and mean tardiness [10]. While the latter gives the highest priority to the waiting operation associated with the job having the most total remaining processing time to be done [11].

It has been decided to keep these two dispatching rules user-selectable during the optimization phase. Although they are not always essential for reaching the optimum, they provide a valuable mechanism to find an alternative scheduling configuration.

2.6 Automation and dosage process application

It is clear how scheduling optimization plays a pivot role in the competitiveness of an industry on the market. The pharmaceutical, manufacturing, food, chemical sectors are all examples of industries that rely on a daily basis on the management of automated systems to maximize efficiency and minimize costs, lead time, cycle time, waste, etc [12].

The context in which this project applies ranges from the chemical and pharmaceutical to the coating and food sectors, since it establishes the basis for the scheduling of an industrial dosage system where barrels must be filled with various ingredients, whether they are fragrances, paints, additives, or flavourings.

Chapter 3

System Description

This chapter shows the structure of the reference system: the layout of the machines and parking spots, the jobs and operations to be performed, the displacement method, and the data.

3.1 Problem statement

This thesis addresses the scheduling problem of an industrial dosage system. The process involves a set of n recipes, each assigned to a barrel and consisting of a specific combination of ingredients. These ingredients are distributed across n_m dosing machines; consequently, each barrel must be routed to the appropriate machines and remain stationed there for the exact time required to complete the dosage, proportional to the quantity to pour. To rest the barrels between operations, there are n_p parking spots, whose number and configuration may vary due to performance or spatial requirements. Besides machines and parking spots, there are an input place, output place and an accommodation for the robot in charge of moving the barrels between stations.

At the beginning of the production, the barrels are staged in the infeed, where they queue before entering the process. When the schedule commands a displacement from the entry point to a machine, the robot arm deals with it. It picks the barrel from the initial position (infeed, machine, parking spot) and takes it to the final position (machine, parking spot) that can ultimately

be the outfeed.

The dimensions of the physical machines and spots proposed by the company are:

Spot	Dimension
Machine	1800mm
Parking spot	500mm
Infeed	1500mm
Outfeed	1500mm
Robot park	500mm

Table 3.1: Individual dimensions of system components.

3.2 Physical Layout

3.2.1 Porposed configurations

Considering a plant composed by 5 machines (A,B,C,D,E) and 6 parking spots ($P_1, \dots, P_6$) the following possible layouts of the system may be considered by the company:

- *Straight line layout* allows to allocate the entire system along a straight line, it saves up space but involves a longer run for the robot in charge of the transportation.

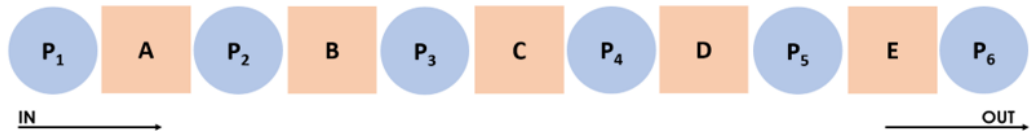


Figure 3.1: System Straight line Layout.

- *Double line layout* takes up more space and clearly separates the machines from the parking spots. It ensures a shorter run but the robot arm would have to operate at a much greater angular range, since passing through the two lines it must rotate of minimum 180° .

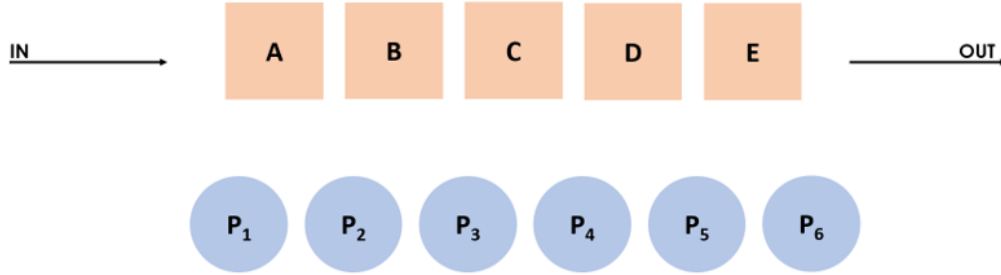


Figure 3.2: System Double line Layout.

- *U-layout* implies to dedicate a wider area to the system and to be equipped with a different route for the robot displacements, because not only does it have to rotate wider, but also its movements must consider a U-shaped track.

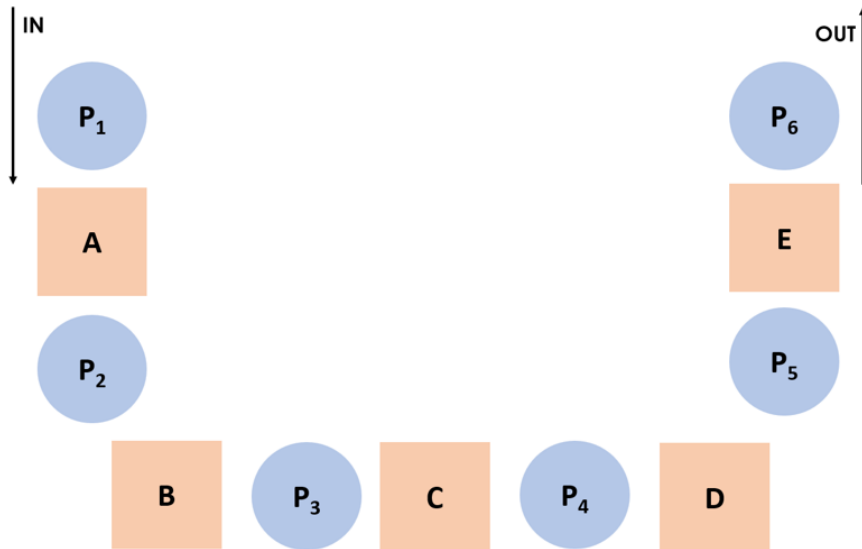


Figure 3.3: System U-Layout.

In these illustrative examples the number of parking spots is set to 6 ($n_p = 6$) following the initial company's requirement. A first objective of this thesis was to analyze the effects of changing this number, to suggest potential modifications, as discussed in *Chapter 7*.

3.2.2 Company's choice

The company confirmed the *straight line layout* for the final configuration, thus the system is arranged in a linear layout, due to space restrictions.

The plant under study consists of $n_m = 5$ machines and $n_p = 6$ parking spots, located between machines and at the edges of the line. As it is shown in Figure 3.1, there are machines A, B, C, D, E , and parking slots P_p , with $p = \{1, \dots, 6\}$.

The robot used to move barrels has a mean speed of $0.5 [m/s]$ and acceleration of $1 [m/s^2]$. Considering that the total length of the line results from the sum of all the stations, the maximum displacement that the robot must perform is $15.5 [m]$, as it can be seen in Table 3.1. Despite the space constraint of having subsequent stations in a straight line, the buffer layout and robot velocity allow to consider the time for displacements almost negligible.

This thesis specifically addresses the need to optimize the system's scheduling through an Integer Linear Programming (ILP) approach. The objective is to sequence the movement of the barrel across various machines handled by a robotic arm, considering spatial distances between stations to complete the entire process in the shortest possible time.

Chapter 4

ILP Mathematical formulation

This chapter presents the general mathematical formulation of the ILP method.

4.1 ILP background and assumptions

Linear programming has a continuous feasible region, meaning that each variable can assume every value in the region. Integer linear programming restricts this region to integer values only. As a result the feasible region is made up of discrete points [13].

The choice of restricting variables to be integer make it easier for some optimization problems to formulate mathematically, but at the expense of computational complexity due to the increase in constraints [14]. However, when dealing with physical units, it is strictly necessary to rely on integer values, for the sake of realism.

The basic elements of the ILP method are:

- Decision variables $x \in \{0, 1\}$: binary variables equal to 1 if the decision is affirmative, 0 otherwise;
- Cost/Objective function: the function to be optimized - either maximized or minimized. In complex scenarios could be multi-objective, in this thesis the aim is to minimize the completion time;
- Constraints: decision variables are subject to mathematical limitations defining the limit of the feasible region and they can be equalities and

inequalities.

The decision variable specifies whether the decision is affirmative or not, under certain constraints and after an evaluation of the decision cost, through the following formulation:

$$\begin{aligned} \min_x \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i \in \{1, 2, \dots, m\} \\ & x_j \geq 0, \quad j \in \{1, 2, \dots, n\} \end{aligned} \tag{4.1}$$

where $f(x) = c^T x$ and $g_i(x) = a_i^T x - b_i$.

The components of the formulation are:

- **optimization variables:** $x = [x_1, \dots, x_n]^T \in \mathbb{R}^n$;
- **feasible region:** $X = \{x \in \mathbb{Z}^n : g_1(x) \leq 0, \dots, g_m(x) \leq 0\}$;
- **feasible points:** $x \in \mathbf{X}$;
- **objective function:** linear function $f : \mathbf{X} \rightarrow \mathbb{R}$;
- **decision cost:** $c = [c_1, \dots, c_n]^T \in \mathbb{R}^n$;
- **constraints:** $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$, $i = \{1, \dots, m\}$;
- **allocation matrix:** $\mathbf{A} = [a_1^T, \dots, a_m^T]^T \in \mathbb{R}^{m \times n}$;
- **capacity vector:** $b = [b_1, \dots, b_m]^T \in \mathbb{R}^m$;

In canonical form, it becomes:

$$\begin{aligned} \min_x \quad & c^T x \\ \text{subject to} \quad & \mathbf{A}x \leq b, \\ & x \in \{0, 1\} \end{aligned} \tag{4.2}$$

4.2 Solution methodology

There exist different solution methodologies to solve the ILP, as mentioned in *Chapter 1*. Among the exact methods there are for example the *Cutting planes*

algorithms, the *Branch and Bound* and *Branch and Cut* methods, and the *Relaxations and decomposition* techniques. The general approach of this methods is to relax the integrality restrictions on the variables and solve the linear constrained program, if the solution is integer, it is the optimal solution of the original problem [15]. The algorithm includes the INTLINPROG function, which uses in particular the *Branch and Bound* method.

This approach is designed for global optimization that maintains provable lower and upper bounds on the optimal objective value. The algorithm explores the solution space by partitioning it into smaller subproblems, that are represented as nodes in a binary tree. The branching step is performed by selecting a binary decision variable and creating two new branches: one with value 0 and one with value 1. To evaluate the lower bound of each subproblem without solving it (which would result in a long process due to the combinatorial nature of the problem) the algorithm uses a continuous relaxation. The integrality constraints are relaxed into continuous intervals, to make the problem easier to solve. As the tree expands, the computational load decreases because any node whose lower bound is greater than the current global upper bound is deleted. This procedure continues until the gap between the global upper and lower bounds falls below a preset tolerance, terminating with the optimal solution [16].

Chapter 5

Logic S.r.l. ILP mathematical formulation

The following chapter investigates how the general ILP formulation has been adapted to the company's system.

5.1 Introduction

The total duration of the planning process DoP is divided into time intervals of 1 minute (sampling time $T_s = 1 [min]$) which corresponds to an iteration. At each iteration the algorithm performs the optimization of the current status of the scheduling, if there are available resources. It is possible to assimilate this system structure to an event-triggered greedy approach, since decisions are taken at regular time intervals, if a resource is released. It is important to note that the DoP is the whole time horizon, but at each iteration, as one minute of operational time elapses, the remaining time available decreases of one minute, shrinking the planning horizon. The residual planning horizon is defined as $RpH = DoP - \bar{t}$, where $\bar{t}$ is the current iteration of the scheduling.

Moreover, the n recipes are broken down based on the machines they have to visit. For example, if recipe **R1** must receive ingredients from 3 different machines (A,B,C), it will generate 3 sub-recipes (**R1_A**, **R1_B**, **R1_C**) related to the same barrel. The optimization variables then will include these

r sub-recipes defining the proper constraints to ensure the same barrel is not collecting ingredients from different machines at the same time.

5.2 ILP mathematical formulation

To adapt the general formulation to the current project, these indexes are introduced:

- $q \in \{1, 2, \dots, n\}$ index of recipes to complete;
- $j \in \{1, 2, \dots, r\}$ index of sub-recipes to schedule;
- $m \in \{1, 2, \dots, n_m\}$ index of machine occupied by sub-recipe j ;
- $p \in \{1, 2, \dots, n_p\}$ index of occupied parking spot;
- $i \in \{1, 2, \dots, n_m + n_p\}$ index of spot occupied (both machines and parking spots);

Therefore the mathematical formulation at a time instant $\bar{t}$ is as follows:

$$\begin{aligned} \min_x \quad & c^T x \\ \text{subject to} \quad & Ax \leq b, \\ & x \in \{0, 1\} \end{aligned} \tag{5.1}$$

where:

- $c = [c_1, \dots, c_r, \tilde{c}_1, \dots, \tilde{c}_r]^T \in \mathbb{R}^{2r}$ is the column vector containing the costs c_j of starting the remaining $j - th$ sub-recipes, and the costs $\tilde{c}_j$ of sub-recipes exceeding the CoT . The CoT is computed at the beginning of the scheduling for the whole barrel (CoT_q , which is a user-defined time parameter), then specialized for each sub-recipe, and iteratively updated based on their minute of starting. In the following, the cutoff associated to sub-recipe j will be referred to as CoT_j .

- $x = [x_1, x_2, \dots, x_r, \delta_1, \dots, \delta_r]^T \in \{0, 1\}^{2r}$ is the optimization variables vector, where its elements are defined as:

$$x_j = \begin{cases} 1 & \text{if sub-recipe } j \text{ is scheduled,} \\ 0 & \text{otherwise.} \end{cases}$$

$$\delta_j = \begin{cases} 1 & \text{if sub-recipe } j \text{ exceeds its } CoT_j, \\ 0 & \text{otherwise.} \end{cases}$$

•

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{\text{machines}} \\ \mathbf{A}_{\text{parks}} \\ \mathbf{A}_{\text{barrels}} \\ \mathbf{A}_{\text{penalty}} \end{bmatrix} \quad (5.2)$$

is the global allocation matrix which is partitioned in sub-matrices according to the constraint they refer to. While they differ in the number of rows, they match the number of columns ($2r$) to map both the r decision variables and r auxiliary penalty variables. The detailed description of the components of $\mathbf{A}$ is reported in Section 5.5;

•

$$b = \begin{bmatrix} b_{\text{machines}} \\ b_{\text{parks}} \\ b_{\text{barrels}} \\ b_{\text{penalty}} \end{bmatrix} \quad (5.3)$$

is the availability vector subdivided into sub-vectors according to the allocation matrix. Additional details are reported in Section 5.5.

5.3 Optimization variables

The decision variables x_j dictate whether a sub-recipe is scheduled during the current iteration. However, failing to schedule the start of sub-recipe j might leave insufficient time for j to finish within its CoT_j , if other sub-recipes in the same machine come first. To penalize this specific condition, r additional variables δ_j have been incorporated into the decision vector and properly constrained. This penalty is necessary because it behaves as a correcting factor, discouraging the solver from selecting a solution that may appear optimal at the current iteration but causes tardiness for other sub-recipes.

Let d_j be the dosage duration of sub-recipe j . Let $\mathcal{M}_j$ be the set of all sub-recipes allocated to the same machine as sub-recipe j . The duration of any competing sub-recipe within the same machine is d_k , where $k \in \mathcal{M}_j \setminus \{j\}$, and let x_k be the binary variable of competing sub-recipes: it takes into account the duration of sub-recipes that may be planned. The activation condition is:

$$CoT_j - d_j \cdot x_j < \sum_{\mathcal{M}_j} d_k \cdot x_k \iff \delta_j = 1 \quad (5.4)$$

With this condition, the penalty δ_j will be activated if and only if the total duration of the selected sub-recipes accessing the same machine does not leave enough time to complete the sub-recipe j . The solver will explore available scheduling paths not to incur in activating δ_j , thereby prioritizing sub-recipes to respect their cutoff times.

5.4 Cost function

The objective function cf represents the function to be minimized:

$$cf = \sum^r c_j x_j + \sum^r \tilde{c}_j \delta_j \quad (5.5)$$

and it is composed of various contributions.

The initial form V0 of the objective function to start sub-recipe j at

minute $\bar{t}$ includes:

$$c_j = \bar{t} + (DoP - CoT_j) \quad (5.6)$$

The cost is strictly dependent on time related factors: the current iteration of the scheduling, and the time window for the sub-recipe to be completed, which progressively decreases as the time horizon approaches.

5.5 Constraints

Matrix $\mathbf{A}$ and vector b represent the core construction of the problem constraints. The allocation matrix gathers information about the current status of the system, while the capacity vector limits the possible choices of the algorithm. The construction of these components tracks the constraints formulation, which consider the different specific aspects of the problem. The components are related to:

- **Machines:** the resources cannot house more than one barrel at once, the relative matrix is

$$\mathbf{A}_{\text{machines}} = \begin{bmatrix} \mathbf{A}_A \\ \mathbf{A}_B \\ \mathbf{A}_C \\ \mathbf{A}_D \\ \mathbf{A}_E \end{bmatrix} \quad (5.7)$$

such that $\mathbf{A}_S \in \mathbb{R}^{RpH \times 2r}$ with $S = \{A, B, C, D, E\}$. Each sub-matrix $\mathbf{A}_S$ represents the constraint of a specific machine. In particular, a generic row of the of the matrix $\mathbf{A}_S$ is related to a time instant $\tilde{t} \in [\bar{t}, DoP]$. If the $j - th$ sub-recipe starts at current iteration $\bar{t}$ and occupies machine S at time $\tilde{t}$, the position $(\tilde{t}, j)$ will be equal to 1.

- **Parking spots:** to ensure physical feasibility, each individual parking slot can accommodate at most one sub-recipe at any given time. This constraint is formalized through the matrix $\mathbf{A}_{\text{parks}} \in \mathbb{R}^{RpH \times 2r}$. This matrix is obtained by summing the machine allocation matrices $\mathbf{A}_{\text{machines}}$ and the specific allocation matrix dedicated to the parking

slots $\mathbf{A}_{\text{park}_{\text{limited}}}$. The latter only refers to the reservation of parking spots beside machines, the elements can assume the following values:

$$\begin{cases} 1 & \text{if sub-recipe } j \text{ needs a parking slot } p \\ 0 & \text{if sub-recipe } j \text{ is already being processed by a machine } m \\ -1 & \text{if sub-recipe } j \text{ must release a parking spot } p \end{cases}$$

(see detailed construction in *Chapter 6*). The combination of these physical places ensures to consider the whole system as a potential waiting area for the barrel. This logic maximizes the utilization of space: if parking spots are saturated and a machine is neither processing a barrel or requested by a sub-recipe, it can be used as a temporary storage space.

- **Barrel integrity:** sub-recipes belonging to the same barrel cannot be scheduled simultaneously and $\mathbf{A}_{\text{barrels}} \in \mathbb{R}^{r \times 2r}$ collects sub-matrices of ones to relate sub-recipes belonging together (see detailed construction in *Chapter 6*). A generic row will contain 1 in correspondence of all the sub-recipes sharing the same barrel.
- **Penalty:** $\mathbf{A}_{\text{penalty}} \in \mathbb{R}^{2r \times 2r}$ doubles the original number of columns of matrix $\mathbf{A}$ and has $2 \cdot r$ rows to add the upper and lower bounds associated to the sub-recipes to implement the constraints reported in Formula (5.4).

The capacity vector b follows the structure of matrix $\mathbf{A}$ and it is composed as follows:

- $b_{\text{machines}}, b_{\text{barrels}} \in \{0, 1\}$: the active sub-recipes in a machine and within a barrel must not be greater than one; this mutual exclusivity constraint states that a machine can process one barrel and a barrel can visit one machine at once. If the resource is occupied or a barrel is being processed the value in b is set to 0, meaning their unavailability.
- $b_{\text{parks}} \in \{0, 1, \dots, n_m + n_p\}$: it represents the maximum number of stations to rest the barrels equals the total number of physical places

in the system, with number of machines n_m and number of parking spots n_p . Whenever a place is occupied, the value decreases of the number of spots taken, reaching 0 if there is no free spot among the stations.

- $b_{penalty}$: this vector gathers lower and upper bounds based on the Formula (5.4).

In particular, to adapt the penalty logic formula (5.4) to a set of valid linear constraints involving the matrix $\mathbf{A}_{penalty}$ and the vector b , it is necessary to specify the lower bound L_j and the upper bound U_j :

$$L_j = -(CoT_j - d_j) \quad (5.8)$$

$$U_j = \sum_{\mathcal{M}_j} d_k \cdot x_k + L_j \quad (5.9)$$

By integrating these bounds, the following constraints are formulated:

$$\sum_{\mathcal{M}_j} d_k \cdot x_k - U_j \cdot \delta_j \leq CoT_j - d_j \quad (5.10)$$

$$\sum_{\mathcal{M}_j} d_k \cdot x_k - (L_j - \epsilon) \cdot \delta_j \leq -(CoT_j - d_j) - L_j \quad (5.11)$$

where ϵ represents a positive infinitesimally small value.

5.6 New multi-objective cost function

The cost function has undergone several additions during the thesis development to address specific scheduling requirements and improve the scheduling performance. These modifications are detailed below.

SPT and MWKR The evolution of the initial form comes from the inclusion of the aforementioned dispatching rules:

- **short processing time SPT**: this term act as a cost reduction. The aim is to prioritize the sub-recipes that occupy the machine for a brief

time, by proportionally lowering the cost based on the operation length:

$$\text{SPT}_j = -0.5 \cdot d_j$$

- **most work remaining MWKR**: this term serves as a strategic weight. For any given sub-recipe, the cumulative processing time of its barrel is computed; the shortest part will have the most remaining work to do and the bigger cost. By prioritizing operations with longer process, it is prevented to generate bottlenecks at later stages.

Let $\mathcal{B}_j$ be the set of all sub-recipes sharing the same barrel as sub-recipe j and let d_w be their individual durations, where $w \in \mathcal{B}_j \setminus \{j\}$. The computation of this cost is:

$$\text{MWKR}_j = 2 \sum_{\mathcal{B}_j} d_w$$

The new cost function V1 including all the components involves:

$$c_j = \bar{t} + (DoP - CoT_j) + \text{SPT}_j + \text{MWKR}_j \quad (5.12)$$

Transport Cost To increase the level of realism, the last adding was considering the transport costs among stations. Every robot movement has been mapped, encompassing the inlet, outlet, machines, and parking slots, with a specific cost assigned based on the distance between these locations. As previously mentioned, the maximum distance the robot must cover is short. Given its operating speed of 0.5 [m/s], the resulting travel costs are minimal and do not significantly influence the overall scheduling.

The most complete version V3 of the cost function includes:

$$c_j = \bar{t} + (DoP - CoT_j) + \text{SPT}_j + \text{MWKR}_j + tc_j \quad (5.13)$$

where tc_j is the transport cost of sub-recipe j .

Another version to be considered is V2, where V0 is extended by the tc_j term, without including the dispatching rules. It will be used in the simulations to compare its individual effect on the scheduling. The term c_j

in version V2 becomes:

$$c_j = \bar{t} + (DoP - CoT_j) + tc_j \quad (5.14)$$

5.7 Illustrative example of matrix composition

Consider a simplified scenario with 2 machines ($n_m = 2$) where $n = 2$ recipes, **R1** and **R2** require both the machines, and must be scheduled in $DoP = 5$ minutes, and the capacity of the parking spots is $n_p = 2$. At first step, the 2 recipes have been divided in 2 sub-recipes, each of them requiring a single machine. In Table 5.1, the first column reports the identification code of the single sub-recipe, the subsequent column clarify the dosage duration in the 2 machines. Then the barrel of belonging is reported, as well as the name of the sub-recipe and its cutoff index. Let's assume the CoT_q of both barrels is equal to the DoP . Considering a sub-recipe j belonging to barrel q , the cutoff index is computed as

$$CoT_j = CoT_q - \sum_{B_j} d_w$$

and when a sub-recipe w starts, the CoT_j is incremented by d_w .

This example is provided solely to facilitate the understanding of the matrix formulation. The next chapter will detail the generation of the following table, which acts as the core database supplying the algorithm with the essential scheduling data.

ID	A	d _A	B	d _B	ID Barrel	Name	CoT idx
1	1	2	0	0	1	R1 _A	4
2	0	0	1	1	1	R1 _B	3
3	1	1	0	0	2	R2 _A	3
4	0	0	1	2	2	R2 _B	4

Table 5.1: Simplified scenario for matrix population.

Recipe **R1** lasts 2 minutes on machine A and 1 minute on machine B,

while recipe **R2** lasts 1 minute on machine A and 2 minutes on machine B. They are then split in **R1_A**, **R1_B**, **R2_A** and **R2_B**. The blocks composing the allocation matrix **A** are constructed below, considering the first iteration of the algorithm.

The machine matrix contains 1 for as much rows as the minutes of the dosage duration of the corresponding sub-recipe. Assume to consider the system at time instant $\bar{t} = 1$. The following matrices will be constructed:

$$\mathbf{A}_A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{A}_B = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (5.15)$$

The matrix that considers only the parking spots indicates with 0 that the spot is not required while the recipe is being processed under a machine, then it is reserved with a 1 for the remaining minutes of scheduling.

$$\mathbf{A}_{\text{parking}_{\text{limited}}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad (5.16)$$

The complete parking matrix used for the final allocation matrix implies both machines and parking spots, it is the sum of the previous matrices and results in:

$$\mathbf{A}_{\text{parks}} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad (5.17)$$

Then, there is the matrix that preserves the barrel integrity, by joining the sub-recipes belonging to the same barrel in the following manner, it is im-

possible for the solver to choose more shared barrel sub-recipes concurrently.

$$\mathbf{A}_{\text{barrels}} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad (5.18)$$

Finally, there is the matrix collecting the penalties. This matrix is composed as follows: odd rows refer to upper bounds, even rows refer to lower bounds; first 4 columns report for each sub-recipe j the sum of the dosage duration of the other sub-recipes competing for the same machine, while last 4 columns report upper bound $-U_k$ and lower bound $-(L_k - \epsilon)$.

$$\mathbf{A}_{\text{penalty}} = \begin{bmatrix} 0 & 0 & 2 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 2 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & -2 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix} \quad (5.19)$$

Next step is to complete the formulation with matrix $\mathbf{b}$: since the status of the system is stopped at iteration 1, before the decision of the solver, the vector contains 1 in each row, except of the block corresponding to $\mathbf{A}_{\text{parks}}$, where the sum of available spots is 4 (2 machines + 2 parking spots).

$$b_A = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad b_B = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad b_{\text{parks}} = \begin{bmatrix} 4 \\ 4 \\ 4 \\ 4 \\ 4 \end{bmatrix} \quad b_{\text{barrels}} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad (5.20)$$

The last element is dedicated to the penalty contribution and contains 8

rows, since the vector gathers upper and lower bounds for each sub-recipe:

$$b_{penalty} = \begin{bmatrix} 2 \\ 0 \\ 2 \\ 0 \\ 2 \\ 0 \\ 0 \end{bmatrix} \quad (5.21)$$

To better understand the parking spots allocation logic and the dynamic adjustments of matrices and vectors, assume to skip to time instant $\bar{t} = 2$. At the previous iteration the solver decided to start sub-recipes **R1_A** and **R2_B**. Matrices **A_A** and **A_B** become:

$$\mathbf{A}_A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \mathbf{A}_B = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (5.22)$$

The number of rows decreases to 4 since the *DoP* shortens of the elapsed minute. While the number of columns decreases to the 2 remaining sub-recipes **R1_B** and **R2_A**. As a consequence, **A_{parkinglimited}** becomes:

$$\mathbf{A}_{\text{parkinglimited}} = \begin{bmatrix} -1 & -1 \\ -1 & -1 \\ -1 & -1 \\ -1 & -1 \end{bmatrix} \quad (5.23)$$

The matrix is composed of -1 because the 2 remaining sub-recipes represent the last part of their respective barrel to complete the process, thus they will not require a further parking spot, but they will exit the system once finished. At the first iteration each $j - th$ sub-recipe reserves a parking spot through a 1 after their dosage duration, so it is necessary to cancel the

previous reservation through a -1. Hence the matrix $\mathbf{A}_{\text{parks}}$ becomes:

$$\mathbf{A}_{\text{parks}} = \begin{bmatrix} 0 & 0 \\ -1 & -1 \\ -1 & -1 \\ -1 & -1 \end{bmatrix} \quad (5.24)$$

Since both sub-recipes $\mathbf{R1}_A$ and $\mathbf{R2}_B$ last 2 minutes, the iteration $\bar{t} = 2$ correspond to the starting minute of the sub-recipes $\mathbf{R1}_B$ and $\mathbf{R2}_A$. It is clear from the matrices that the 2 spots occupied by these last sub-recipes are the machines $S = \{A, B\}$, while the parking spots are completely free. Actually, in this scenario they are not used, because when the first two sub-recipes finish, the machines interchange barrels and continue the process without interruption. Since only the last parts of both barrels lack, there is no more a mutual exclusivity constraint to respect:

$$\mathbf{A}_{\text{barrels}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (5.25)$$

Consequently, before the solver's decision of time instant $\bar{t} = 2$ sub-vectors b become:

$$b_A = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad b_B = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad b_{\text{parks}} = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 2 \end{bmatrix} \quad b_{\text{barrels}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (5.26)$$

The minutes of planning decrease to 4, and circulating sub-recipes are 2. Both sub-recipes $\mathbf{R1}_A$ and $\mathbf{R2}_B$ finish, so both barrels and both machines are available for other processes.

The following minute, the solver reads that the initial sub-recipes has finished at $\bar{t} = 2$ and understands that the other two sub-recipes can be scheduled.

Chapter 6

Implementation

This chapter details the translation of the mathematical formulation into a MATLAB script. The architecture of the algorithm, which dynamically evaluates the system state to schedule the available recipe, is mapped in the initial section. Figure 6.1 illustrates the logical sequence of the optimization loop.

6.1 System setup

At the beginning of the algorithm it is necessary to formalize the initial variables of the system, to have a direct and easy access to parameters to change.

First, the DoP variable is defined to establish the time horizon within which the entire dosage process must be completed. Subsequently, the `Logic_configuration` file is loaded. The script contains the system's specifications given by *Logic S.r.l.*, such as the number of machines, the capacity of the parking spots, the total number of available spots, and their physical dimensions as detailed in Table 3.1.

Then, the `Data_collection` file contains the ingredient data extracted from the documents provided by the company, which will be utilized for the formulation of the recipes and their respective specifications.

Another key parameter that needs to be flexible is the total number of recipes. Keeping it at the top of the algorithm through variable `Numb_Recipes`

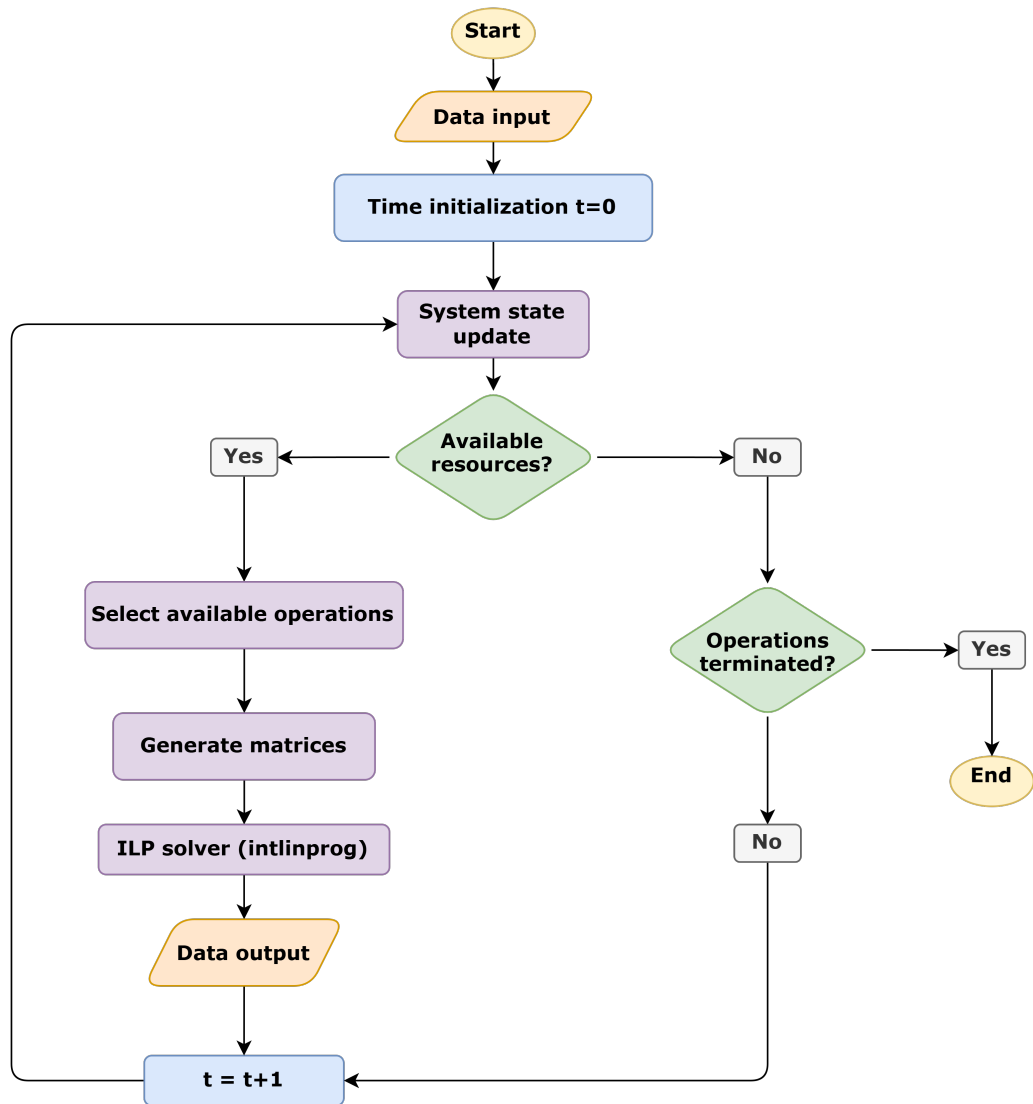


Figure 6.1: Flowchart of the optimization loop.

allows to change it easily.

6.1.1 Data Acquisition and Pre-processing

Raw data sheets can report undetected errors resulting in a misleading data reading. In the data set received by *Logic S.r.l.* there was an ingredient which stated a duration of hundreds of minutes. This could happen in every industrial reality, where any kind of unexpected event, like the finishing of an ingredient or the breakdown of the machine, could modify the data recording. This lead to a recipe that could not be finished. Pre-processing this data sheet, eliminating the disruptive influence, allowed to use exactly the dosing durations listed in the given technical sheet. For this reason this *JSSP* can be classified as deterministic, since each operation is given a predetermined and unique processing time [1].

6.1.2 Industrial specifications and Model assumptions

Real-world requirements According to the industrial specifications provided by the company, the operational workload consists of 80 distinct recipes within the production cycle. The structural composition of these recipes is variable: the average number of ingredients is 25, but with an operational variability that ranges from 5 to 45. The overall catalog accounts for 600 substances, uniformly allocated among the 5 operating machines. Furthermore, the finished recipes require payload limits between 1 and 25 [kg], with average output of 4[kg].

Lastly, the time horizon is set to 1440 minutes (24 hours), thus it does not correspond to an operator's shift, but rather to a cyclic operational horizon suited to the planning requirements.

Model delimitation The data sheet given by the company contained much information about the batch it came from and only codes, quantity and dosage durations were considered. Among the listed ingredients, only 25 identified as unique; therefore the requested quantities of recipes and ingredients per recipe could not be met. In fact, there has been a decrease to

a maximum of 31 recipes including between 10 and 25 ingredients, according to the company's approval. Also, the weight bound has not been taken into consideration, due to the limitations of ingredients and the consequent inability to mix a huge variety of components.

Finally, for the same reason, the total *DoP* did not need to be set to an entire day, but it could be reduced as necessary, to effectively test the capability of the optimization to complete the scheduling within the deadline.

6.1.3 Data organization

To actively use the information gathered, the script `Data_collection`, has been designed to organize the data in a table, as reported in Table 6.1. The generation steps are as follows:

1. Compute the amount of times the same ingredient code appears in the sheets and the frequency of occurrence via `groupsummary`;
2. Convert the time into numbers and calculate the dosage duration of each row of each ingredient and sum it to get the absolute duration, using `datetime`;
3. Compute the total weight of each ingredient;
4. Normalize the duration and the frequency to make them comparable;
5. Give a score based on the previous normalization and rank each name;
6. Assign machines based on a serpentine path (more details reported in the following).
7. Refine the allocation based on the least loaded machine (in terms of score).

To obtain an optimal load balancing, the ingredients were distributed following a serpentine allocation strategy. However, a first test revealed a structural bottleneck: by assigning the most frequent ingredient (the first one in Table 6.1) to machine A and the following ones to the other machines, the distribution was highly asymmetrical regarding the weight. To mitigate

Code	Occurrence	Weight	Duration	Frequency	Score	Machine
8007400	388	2.4893e+03	1.6027e+03	38.6839	2	A
8011206	80	233.2980	88.7206	7.9761	0.2615	B
8014151	76	323.4260	102.0194	7.5773	0.2595	C
1032687	20	122.8350	263.9569	1.9940	0.2162	D
1021406	41	102.3320	163.5036	4.0877	0.2077	E

Table 6.1: First lines of the ingredient information table.

this load disparity, a dynamic routing strategy was implemented: to redirect ingredients to the machine with the lowest cumulative load at any given step. Three different versions were evaluated: temporal load (cumulative processing duration), weight load and score-based load.

Among these criteria, the score-based approach proved to be the most robust. For each y -th ingredient, the score is computed as the sum of its normalized dosage duration and its normalized request frequency, defined as follows:

$$Score_y = \frac{d_y}{\max(D)} + \frac{f_y}{\max(F)}$$

where:

- d_y represents the time required to pour the y -th ingredient;
- $\max(D)$ is the maximum dosage duration among all available ingredients y ;
- f_y denotes the frequency, thus the number of times the y -th ingredient is requested to form a recipe;
- $\max(F)$ is the maximum recorded frequency among all ingredients y .

Through this normalized formulation, the algorithm effectively weighs both the physical time it takes for the machine to dispense an ingredient and its occurrence rate within the overall production batch.

Despite the adjustments, the final allocation remained unbalanced. This asymmetry highlights a decoupling between the dosage duration and the

physical weight of the ingredients in the data sheet. This suggests the presence of different sized valves, depending on the required batch volume, or the switching from a high-flow valve to a low-flow valve to ensure precision at the end of the operation.

Since the dataset does not provide this peculiarity, the algorithm cannot compensate for the flow changes. As a consequence, the workload imbalance cannot be eliminated. This is not a limitation of the allocation strategy, which successfully optimizes the routing as shown in *Chapter 7*.

6.1.4 Recipes generation

For the academic purpose of this project, the recipes do not come from existing databases but are generated through the function `Recipe_generation`, combining real ingredients and real dosage durations to maintain realism. The logic of the function can be summarized in Algorithm 1.

Algorithm 1 Randomized selection of ingredients to compose realistic recipes.

```

1: for the selected Numb_Recipes do
2:   Generate a random integer  $y$  between 10 and 25
3:   Sample  $y$  ingredients from the database
4:   Retrieve their corresponding identification codes
5:   for each code do
6:     Identify the assigned machine from the sorted list
7:     Extract weight from the complete database
8:     Populate RecipesComposition
9:   end for
10: end for

```

`RecipesComposition` is the table that gathers the composition of all the recipes generated by the script, including both the identification code of the ingredient and its weight, which is mainly used to differentiate between rows belonging to the same ingredient but different dosage durations.

Each time an ingredient is included in a recipe, it is deleted from the complete database. If a large number of recipes is generated, specific ingredients may be depleted; consequently, the algorithm progressively reduces the number of available options, until the number set in the variable

Number of recipes is reached.

6.1.5 Database generation

The next step consists of generating the database that contains all the data that the algorithm needs. It is named **RecipeDataBase** and follows the logic in Algorithm 2.

Algorithm 2 Generation of recipes information table.

```

1: for the selected number of recipes do
2:   for each row in RecipesComposition do
3:     Read the number of ingredients
4:     Retrieve dosage duration and machine assigned from the original
       database
5:     Compute the total visiting time of a barrel in a machine
6:     Record it in RecipeDataBase
7:   end for
8:   Compute cutoff time
9:   Compute cutoff index
10: end for

```

The **cutoff** time is a user-selected value, typically dictated by operational demands in the real industrial environments. In the presented work, the cutoff time is generally set to 10 hours, unless the duration of the task exceeds 8 hours. In that case, the cutoff is set to $d_j + 2h$. Initially, **CoT idx** is set equal to the cutoff, expressed as elapsed time from the start of the optimization to the theoretical completion time of the recipe, assuming it is completed without interruption. This index must be initialized in advance because it will subsequently be used to split the barrels.

The subsequent function **SplitSharedBarrels** completes the generation of the database and its output is shown in Table 6.2. **RecipeDatabase** becomes an extended version, while it is kept record of the compact database in **RecipeDataBase_original**. Each recipe in the **RecipeDataBase** is parsed and split into sub-recipes, each destined to be processed on a different machine. This procedure allows to shift the focus from the barrel to the individual sub-recipes. Even if it could arise a doubt on the integrity of a barrel, the correct formulation of the constraints allows to exploit this simplification.

Actually, to ensure traceability of the parts belonging to the same barrel, the **ID Barrel** remains unchanged for every sub-recipe. Consequently, the **CoT idx** is dynamically updated: compared to the original barrel cutoff CoT_q , the CoT_j for each sub-recipe j is reduced by the cumulative processing time of the remaining parts of the same barrel. This guarantees that every sub-recipes is completed with enough time left to process the subsequent sub-recipes. The cutoff column has been voluntarily omitted since in this scenario it is fixed to 10 hours.

The first sub-recipe that appears in Table 6.2 is **R1_A**. Each row is dedicated to a sub-recipe, hence there will be only one machine activated with its processing time in the adjacent column. The first line keeps the same ID as the relative barrel, while the following codes of the same recipe are numbered from $n + 1$ ahead to avoid duplicates.

Finally, last column **shared** equals 1 if the barrel must visit more than one machine to be completely filled.

Table 6.2 is an illustrative example of the subdivision of the first barrel.

ID	A	d _A	B	d _B	C	d _C	D	d _D	E	d _E	ID Barrel	Name	CoT idx	shared
1	1	3	0	0	0	0	0	0	0	0	1	R1 _A	574	1
6	0	0	1	15	0	0	0	0	0	0	1	R1 _B	586	1
7	0	0	0	0	0	0	1	5	0	0	1	R1 _D	576	1
8	0	0	0	0	0	0	0	0	1	6	1	R1 _E	577	1

Table 6.2: Subdivision of Barrel 1.

6.1.6 Temporal flow

After having rearranged all the data, obtaining a clear and detailed table, the optimization process can begin. First information to display at the beginning of the algorithm is the real starting date and time of the scheduling process, hence the `fix(now*1440)` is utilized, which is a function call that gives as output the current date and time as a serial date number.

The temporal flow of the scheduling process is managed through an iterative loop defined by `iCurrIter = 1:t_sample:durationOfPlanning`, where `t_sample` is T_s , and `durationOfPlanning` is DoP . This formulation ensures

that the simulation is updated in discrete time steps, covering the entire planning horizon.

Another index of the performance of the strategy presented in this thesis is the computational time required to solve the problem and complete the job scheduling. Hence, the computational time has to be recorded.

The variable that starts the minutes count is `tstart=tic`, where `tic` is a built-in function to start the timer. At the end of the optimization process, the `tend=toc` variable utilizes the built-in function `toc` to stop the time recording.

6.2 Matrices generation

The allocation matrix A and the constraint vector b are the fundamental basis of the ILP formulation, the success of the scheduling algorithm relies on the correct generation of these elements. Specifically, matrix $\mathbf{A}$ translates physical and temporal limitations into mathematical constraints, while vector b dictates the maximum capacity of the resources in the constraint formulation.

6.2.1 Matrix A

This section details the implementation of the mathematical formulation introduced in *Chapter 5* providing a more complete comprehension of the matrices construction, when necessary.

Machines The population process of the allocation matrix relies on scanning the `Recipe_hist` database, which tracks the pending sub-recipes (its generation via the `Recipe_List_update` script will be discussed later).

For every sub-recipe in the list, the algorithm identifies the assigned resource and its processing duration. Consequently, it populates the specific matrix block of that machine. For example, a 4-minute operation on machine A for sub-recipe 1 results in four consecutive ones in the first column of the machine A sector ($\mathbf{A}_A$). If sub-recipe 2 requires machine B, the algorithm

shifts to the machine B sector ($\mathbf{A}_B$) of the matrix, filling the second column with ones according to its specific duration. This effectively maps the spatio-temporal scheduling of each barrel as seen in the illustrative simplified example in *Chapter 5*.

A critical edge case may occur: if the processing duration of a candidate sub-recipe exceeds the total planning horizon, the algorithm assigns a high cost (e.g. 1000). This results in violating the respective constraint and rejecting the operation during the selection process.

Parking spots Algorithm 3 explains in detail the logic behind the construction of the matrix referred to the parking spots mapping.

Algorithm 3 Parking allocation logic to populate $A_{park_{limited}}$

```

1: for each candidate sub-recipe  $j$  in Recipe_hist do
2:   if sub-recipe  $j$  shares a barrel then
3:     Extract barrel ID and determine phase status (is_new_barrel,
       is_last_phase)
4:     Calculate working_mins on the assigned processing machine
5:     if is_new_barrel and not is_last_phase then
6:       Reserve parking slots for the entire horizon (A_parklimited =
       1)
7:       Free parking slots during working_mins (A_parklimited = 0)
8:     else if is_new_barrel and is_last_phase then
9:       Do not reserve parking slots (A_parklimited = 0)
10:    else if not is_new_barrel and not is_last_phase then
11:      Do not reserve new parking slots (A_parklimited = 0)
12:      Apply negative adjustment during working_mins
      (A_parklimited = -1)
13:    else if not is_new_barrel and is_last_phase then
14:      Permanently drop parking requirements (A_parklimited =
      -1)
15:    end if
16:  end if
17: end for

```

This algorithm is then summed to matrices A associated to the five machines, because as mentioned in the previous chapters, the objective is to exploit every free spot, so both parks and inactive machines are taken into

consideration. The resulting matrix is $\mathbf{A}_{\text{parks}}$.

The population happens in four different states:

- **New barrel, multiple phases:** when a barrel newly enters the system, a parking slot is reserved for the entire planning horizon except from the minutes during which it is processed by the machine;
- **Single-phase barrel:** if it requires only one machine, it won't need any waiting slot;
- **Started barrel, intermediate phase:** if the barrel has begun circulating in the system, new parking slots are not requested, because at its first operation they were already reserved; furthermore, a negative adjustment is necessary during the working minutes to delete the previously set ones;
- **Started barrel, last phase:** if the objective is to finish the filling, the barrel will exit the line, hence the parking requirement is permanently dropped off.

Barrel integrity To ensure the physical integrity of the barrels, the scheduling model must guarantee that sub-recipes belonging to the same barrel are never processed simultaneously. This mutual exclusion is mathematically enforced through the symmetric square matrix $\mathbf{A}_{\text{barrels}}$. For each candidate sub-recipe, the algorithm scans the previously evaluated operations up to the number of machines minus one (which is the maximum possible number of remaining phases). If a preceding sub-recipe shares the exact same barrel, a symmetric 1 is assigned to their intersection in the matrix. When passed to the solver, this formulation prevents any temporal overlap between parts of the same physical entity. Algorithm 4 is implemented in the first `if cycle` of Algorithm 3.

Algorithm 4 Barrel integrity logic

```

1: for each candidate sub-recipe  $j$  in Recipe_hist do
2:   if the sub-recipe has preceding operations to evaluate ( $j > 1$ ) then
3:     for each previous respective step do
4:       Determine the index of the previous operation
5:       if the previous operation belongs to the same barrel then
6:         Mark their intersection in the  $\mathbf{A}_{\text{barrels}}$  matrix with 1
7:         Apply the symmetric constraint with 1
8:       end if
9:     end for
10:   end if
11: end for

```

6.2.2 Vector b

Vector b represents the physical and time constraints for the sub-recipes. The physical constraints are collected in a sub-vector $b_1 = [b_{\text{machines}}^T, b_{\text{parks}}^T, b_{\text{barrels}}^T]^T$. Due to the fact that it has to comply with the dimension of matrix $\mathbf{A}$, it is subdivided into parts for the machines, where the maximum capacity is set to 1, a part for the overall parking spots, set to $n_m + n_p$ (machines+parking spots). This construction avoid exceeding the physical space to be occupied:

$$b_{\text{machines}} = \begin{bmatrix} \text{Capacity}_A \cdot \mathbf{1}_{RpH}^T \\ \text{Capacity}_B \cdot \mathbf{1}_{RpH}^T \\ \text{Capacity}_C \cdot \mathbf{1}_{RpH}^T \\ \text{Capacity}_D \cdot \mathbf{1}_{RpH}^T \\ \text{Capacity}_E \cdot \mathbf{1}_{RpH}^T \end{bmatrix} \quad (6.1)$$

$$b_{\text{parks}} = \left[\text{Capacity}_{\text{parks}} \cdot \mathbf{1}_{RpH}^T \right] \quad b_{\text{barrels}} = \left[\text{Capacity}_{\text{barrels}} \cdot \mathbf{1}_{RpH}^T \right]$$

where $\mathbf{1}$ equals a vector of ones.

Each iteration the vector b_1 is dynamically updated subtracting the vector *Schedule*, obtaining vector b as follows:

$$b = b_1 - \text{Schedule}$$

This updated vector collects the physical quantities running through the system at each single iteration. In fact, it is iteratively decreased of the number of units circulating, resulting in the remaining free spots.

This procedure allows to obtain both matrix $\mathbf{A}$ and vector b with three parts dedicated to capacity constraints for the machines, the parking spots and the barrels.

6.2.3 Upper bound and Lower bound

The construction of the upper and lower bounds follow the mathematical equations explained in *Chapter 5* and the placement engage both matrix $\mathbf{A}$ and vector b .

Matrix $\mathbf{A}_{\text{penalty}}$ To implement the penalty constraint the matrix $\mathbf{A}$ is dynamically expanded to accommodate the new variables. The base matrix is doubled with a new matrix of zeros, to create space for the decision variable δ_j , as discussed in *Chapter 5*. For each iteration, two constraint row vectors are computed: $A_{\text{penalty}U}$ and $A_{\text{penalty}L}$. The first section encodes the processing durations of active sub-recipes sharing the same machine, except of the current one, the second part addresses the penalty variables:

$$A_{\text{penalty}U} = \left[\sum_{\mathcal{M}_j} d_k \quad - U_k \right]$$

where $U_k = \sum_{\mathcal{M}_j} d_k + L_k$.

For the second vector:

$$A_{\text{penalty}L} = \left[- \sum_{\mathcal{M}_j} d_k \quad - (L_k - \epsilon) \right]$$

where $L_k = -(CoT_j - d_j)$.

Then, these rows are concatenated at the end of the allocation matrix, obtaining the matrix $\mathbf{A}$ that will be put as input of the solver.

Matrix b_{penalty} The right-hand side vector b is iteratively updated to set the limits of the previous equation. For each j the bound values $b_{\text{penalty}U}$ and $b_{\text{penalty}L}$ are established and represent the temporal limit, in fact they

are given by:

$$b_{penaltyU} = CoT_j - d_j$$

$$b_{penaltyL} = -(CoT_j - d_j) - L_k$$

Therefore, $b_{penalty}$ is vertically concatenated to b_1 , to obtain the final vector b that will be an input of the solver as well.

6.3 Objective function

The objective function cf_j for sub-recipe j contains information about the current iteration $\bar{t}$, its cost $c_j = 1$ or with the penalty $\tilde{c}_j = 5000$, the difference between the sub-recipe cutoff CoT_j and the planning horizon DoP , the dispatching rules SPT_j and $MWKR_j$ and the transport costs tc_j , as introduced in *Chapter 5*. The latter element was introduced through a comprehensive mapping of all possible barrel movements, prior to calculating the individual cost of the sub-recipes.

6.3.1 Movement tracking

The strategy adopted to compute the transport costs tc is to read all the remaining sub-recipes, with their actual, previous and future position, in order to compute the cost of displacement. Indeed, the transport mission is divided into two phases: pick-up and drop-off. First, the automated system must retrieve the barrel from its last position, then it must bring it to the drop-off destination as indicated in the database. This two-step journey is iteratively evaluated for all pending sub-recipes, thereby providing the cost function with the necessary information to evaluate the optimal scheduling choice.

In function `transport` the system is modeled as a physical structured space. The layout is divided into 13 strategic nodes, considering machines (1-5), parks (6-11) and input and output (12-13). The system arrangement becomes:

$$nodes = [12, 6, 1, 7, 2, 8, 3, 9, 4, 10, 5, 11, 13]$$

following the selected layout in Figure 3.1. Each node is assigned a size depending on the dimensions in Table 3.1, specifically taking into account the center of the entities, since it is assumed to be the robot's stop position:

$$sizes = [1.5, 0.5, 1.8, 0.5, 1.8, 0.5, 1.8, 0.5, 1.8, 0.5, 1.8, 0.5, 1.5]$$

Then, the symmetric cost matrix has dimension 13×13 , the number of rows i and columns s correspond to the number of nodes. Taking as an example $nodes(i = 1) = 12$ and $nodes(s = 2) = 6$, the related transport cost tc between these stations is built as follows:

$$\begin{aligned} sizes(1) &= 1.5; & sizes(2) &= 0.5 \\ centers(1) &= \frac{sizes(1)}{2} = 0.75 \\ centers(2) &= centers(1) + \frac{sizes(1)}{2} + \frac{sizes(2)}{2} = 1.75 \\ distance &= |centers(2) - centers(1)| = 1 \\ \mathbf{cost_matrix}(12, 6) &= \lfloor distance \rfloor = 1 \end{aligned}$$

where $center(i)$ and $center(s)$ represent the assumed stop position of the robot, and distances are rounded to the nearest integer value, to comply with the requirement of the ILP.

6.3.2 Objective function versions

The change of objective function across the four versions is easily managed through a **switch-case** control structure with a flag at the beginning of the algorithm, assuming values from 0 to 3. This modular approach allows to selectively append or exclude the reward strategies (short processing time and most work remaining) and the dynamic movement cost from the baseline formulation. The resulting entity is a cost vector, containing the individual value assigned to each sub-recipe, which serve as input for the solver.

6.4 Solver

To compute the optimal scheduling configuration, the system relies on the MATLAB built-in solver `intlinprog`, designed for MILP (Mixed ILP) problems. The process is managed through the `optimoptions` function to suppress the default output. When executing the solver iteratively, disabling the display of statistics is a choice to eliminate graphical load and accelerate the elaboration time.

The solver receives the cost vector c , the *intcon* parameter (that impose the integer constraint on all decision variables), the complete matrix $\mathbf{A}$ and vector b , and the lower and upper bound vectors set respectively to 0 and 1, providing a constraint for the solver to formulate binary decision variables.

The output is composed of three main parts:

- x : it is the vector of binary decisions, including scheduled recipes and penalty activation.
- `fval`: it is the overall optimal value of the optimization.
- `exitflag`: it represents whether the solver found the optimal solution (1) or there has been a problem (other numbers).

6.5 Database population

The first half of the vector output x of the solver is used for two main objectives: remove the started sub-recipes from the list of pending sub-recipes `Recipe_hist`, and delete the respective rows in the scheduling vector `Schedule` for the elapsed minute and the started sub-recipe. This allows a dynamic scaling of the matrices, in order to reduce their size according to the passing of time.

Once the sub-recipes history is updated, the database `RecipeDataBase` is completed with start and end time of each started sub-recipe j as follows:

$$\begin{aligned} start_j &= \bar{t}; \\ end_j &= \bar{t} + d_j \end{aligned}$$

Having the final decision of the solver, a dedicated tracking table is populated to record the actual movements of the robot, derived from the actively started sub-recipes. This registry acts as the final reference for the **transport** function. During the subsequent iteration, the function fetches these confirmed positions to map the current layout and compute the candidate trajectories for the next scheduling operation.

6.5.1 System state update

When the optimization starts, two main aspects must be coherently updated: the list of sub-recipes and the matrix of time. The script and function that deals with this management are, respectively, **Recipe_List_update** and **Update_Schedule**.

Update of Recipes The main logic behind the update of the list of remaining sub-recipes to be done relies on the reading of the output vector x of the solver, and it is explained in Algorithm 5.

Algorithm 5 Update of the sub-recipes list.

```

1: if there are pending sub-recipes then
2:   Retrieve the  $q$  barrel of the started sub-recipe  $j$ 
3:   Retrieve the other sub-recipes in  $\mathcal{B}_j$  sharing the same barrel
4:   if there are other operations for the same barrel then
5:     Update cutoff and cutoffIndex of the remaining operations by
       shifting their deadlines forward by the duration of the scheduled sub-
       recipe
6:   end if
7:   Remove iteration rows from Schedule
8: end if

```

This part of the algorithm has the aim of updating the cutoff time of the sub-recipes, taking into account the starting minute of its relative parts. When an operation on a barrel starts, the deadline of the same barrel parts must be shifted forward because the initial time reference for the cutoff time computation is the beginning of the optimization. In each sector of the

Schedule, then it is deleted the row corresponding to the current minute of scheduling, keeping the remaining minutes to follow the real passing of time.

Update of the Schedule The update of the vector *Schedule* deals with the update status of the transiting barrels, as in Algorithm 6.

Algorithm 6 Update of the sub-recipe part of the schedule.

```

1: for each sub-recipe  $j$  do
2:   if its barrel is currently in use then
3:     Retrieve the sub-recipes sharing the barrel
4:     Retrieve the last scheduled operation
5:     if its finishing time exceeds the current iteration then
6:       Update Schedule with 1 in the row associated to the  $j$ -th row
       of Recipe_hist
7:     else
8:       Update the same row with 0
9:     end if
10:  end if
11: end for

```

This function keeps 1 for the sub-recipes that cannot be selected because their barrel is occupied in an operation. It is fundamental for the scheduling since it prevents the solver from considering an awaiting sub-recipe as selectable despite its occupied barrel.

Chapter 7

Simulations and Results

This chapter presents the simulations performed to validate the proposed scheduling algorithm, demonstrating its operational efficiency under varying production scenarios.

7.1 Proof of concept

To evaluate the effectiveness of the implemented strategies, a subset of $n = 10$ recipes and $n_p = 6$ parking spots is analyzed through the 4 different versions of the objective function, described in *Chapter 5*. At first, it is possible to provide a visual representation of the scheduling to intuitively highlight the macroscopic differences, thanks to the limited number of sub-recipes $r = 49$. Subsequently, a quantitative analysis is performed to evaluate the Key Performance Indicators (KPIs):

- Elaboration time: the time elapsed to perform a simulation by a personal computer equipped with an AMD 3020e CPU operating at 1.20 GHz, with 8 GB RAM;
- Nlate: the number of recipes finishing after the CoT ;
- Average lead time: average completion time of a sub-recipe. It is calculated through the mean value of the completion time of all the recipes

$$\bar{lt} = \frac{\sum_{q=1}^n te_q - ts_q}{n}$$

where te and ts are the finishing and starting minute of each recipe q , respectively.

- Lead time lt_{max} : maximum completion time of the last recipe to finish

$$lt_{max} = \max(te_q)$$

- Inactivity of machines S (IoM_S): percentage of idle time of each machine. It is computed by subtracting the machine's active utilization percentage from 100% , where the utilization is the ratio between the total physical dosing time (wm) and the machine's maximum occupation time (om)

$$IoM_S = 100 - \frac{wm_S}{om_S}$$

The main versions of the cost functions can be summarized as follows:

- V0: the first scenario represents the starting point, where sub-recipes only aim at respecting their cutoff times and the total planning horizon;
- V1: the second version introduces the short processing time and most work remaining rules;
- V2: the third version adds to the V0 the transport costs;
- V3: the last scenario includes both the dispatching rules and the transport costs.

Figure 7.1 presents the simulation of V0. The x-axis represents the scheduling minutes, where only the useful values are reported, and the y-axis represents the machines S . The recipes are color-coded, thus it is possible to visualize in each machine the temporal location of a particular barrel. Black and striped blocks represent the occupation of machines by awaiting barrels that cannot be moved because of parking spots saturation. Empty slots indicate the idle times of machines.

Although the first version presents positive aspects, such as the continuous working time of machine D, there are some inefficiencies. The machines,

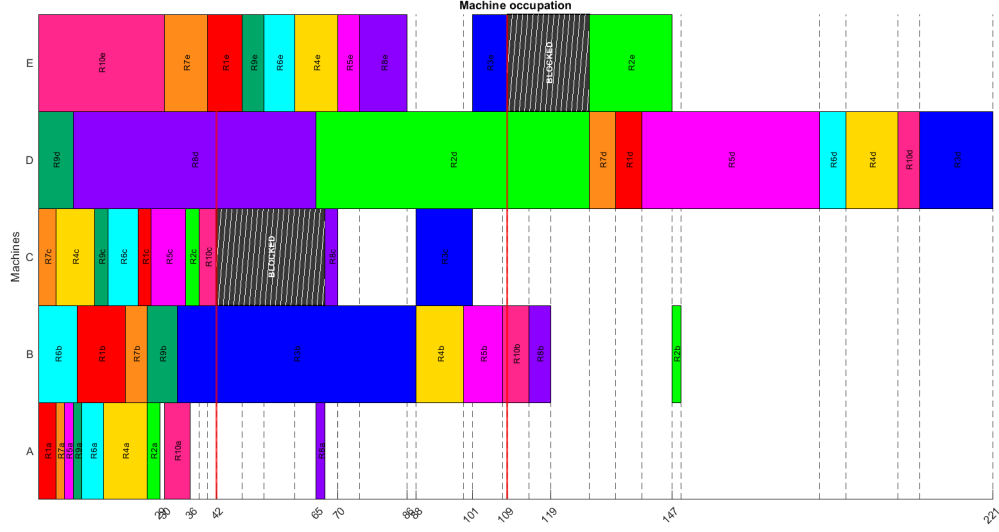


Figure 7.1: 10 recipes scheduling version V0.

except of D, show significant dead times: A (in the intervals 29-30 and 36-65), B (in the interval 119-147), C (in the interval 70-88), E (in the interval 86-101). Furthermore machine C and E are also used as parking spots for 25 and 19 minutes, respectively. It can be seen in both Figures 7.1 and 7.2, where two red lines indicate the starting minute of the blocking machine times, showing that the parking spots are entirely occupied at those time instants.

The second version V1 introduces the dispatching rules. This strategy aims at prioritizing short sub-recipes within machines to avoid bottlenecks, and long sub-recipes within a barrel to prevent it from being occupied for extended periods later in the production cycle, when the residual number of sub-recipes is lower and no other task can be processed while waiting. Indeed, Figure 7.3 shows a substantial improvement with respect to the previous version. Idle times and machine blocking times are drastically reduced, demonstrating a better parking spots management. It is also visible the prioritization of short recipes within machines, which has the biggest influence in this scenario.

Figure 7.4 represents version V2, where transport costs are added to version V0. As anticipated throughout the description of the problem, these

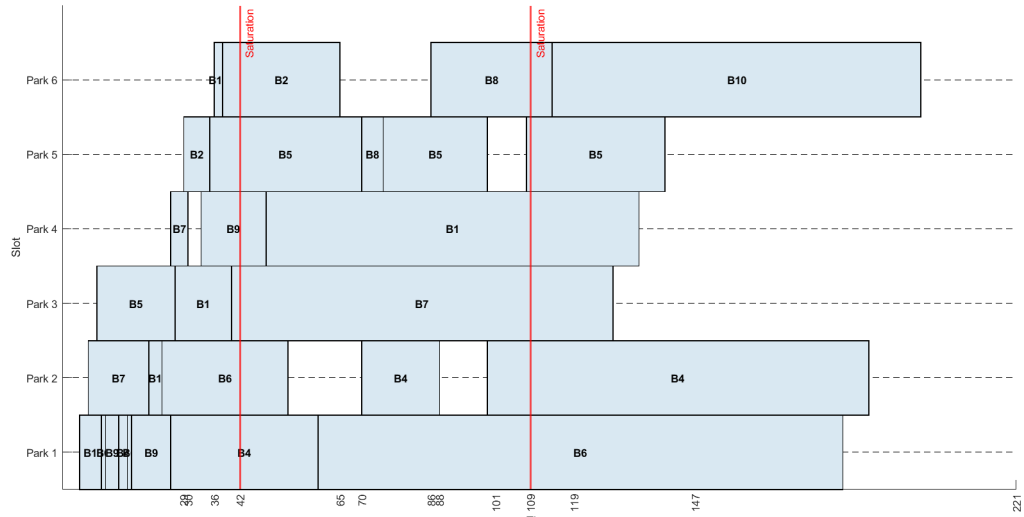


Figure 7.2: Saturated buffer in 10 recipes scenario, version V0.

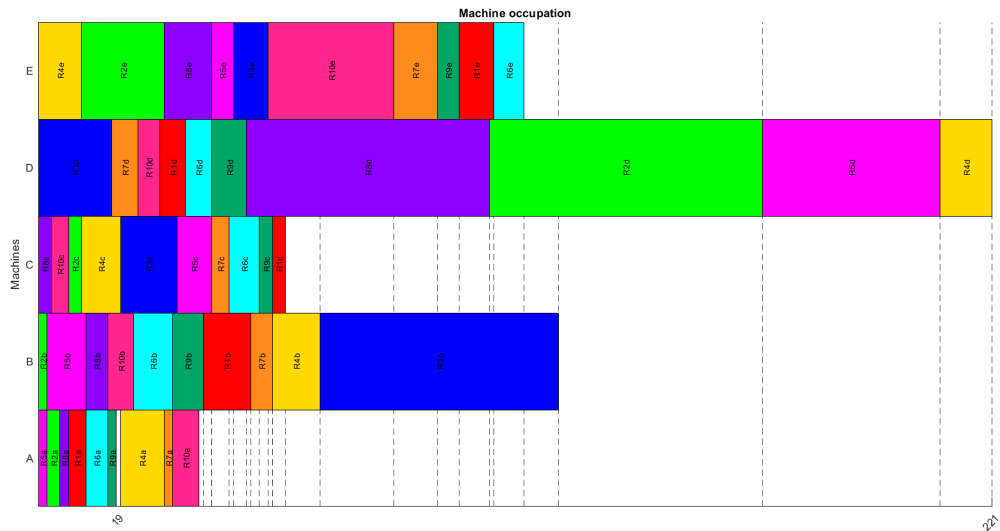


Figure 7.3: 10 recipes scheduling version V1.

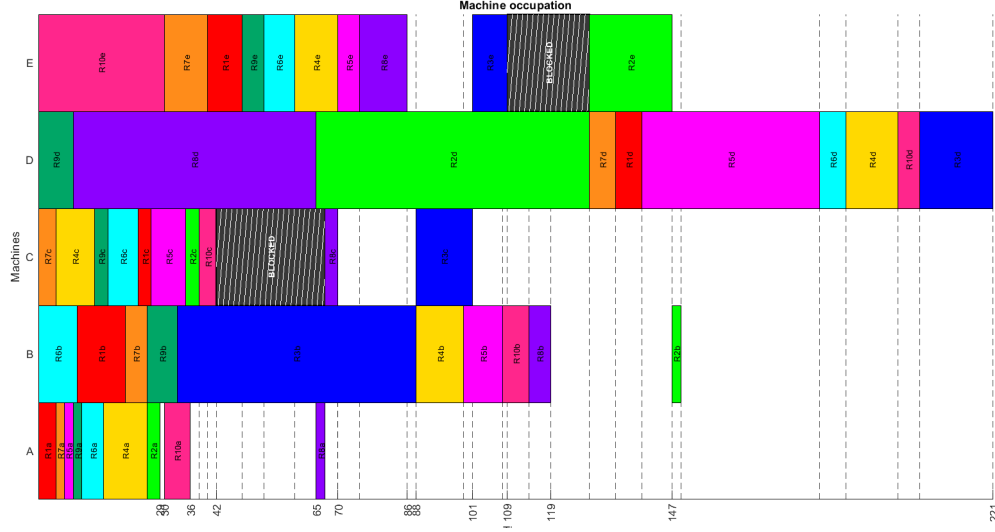


Figure 7.4: 10 recipes scheduling version V2.

costs are negligible. In fact, the two charts appear to be the same, as it will be clearer also from a quantitative point of view, through Table 7.1.

Last Figure 7.5 shows the optimized scheduling thorough the most complete version of the cost function, that includes the dispatching rules and the transport cost. Although the difference may be visually negligible, the effect of the transport costs is noted through the sub-recipes arrangement. For example, in machine B, $\mathbf{R1}_B$ (red) is displaced from minute 39 in V1 to minute 23 in V3, being closer to $\mathbf{R1}_A$ that finishes at minute 12. Another noticeable difference concerns $\mathbf{R6}$ in machines B and C. In version V1 the elapsed time between them is 13 minutes (in the interval 32-45), while in version V3 is 3 minutes (in the interval 45-48). It can be concluded that the effect on the overall completion time is almost absent, but the internal configuration of the scheduling is modified.

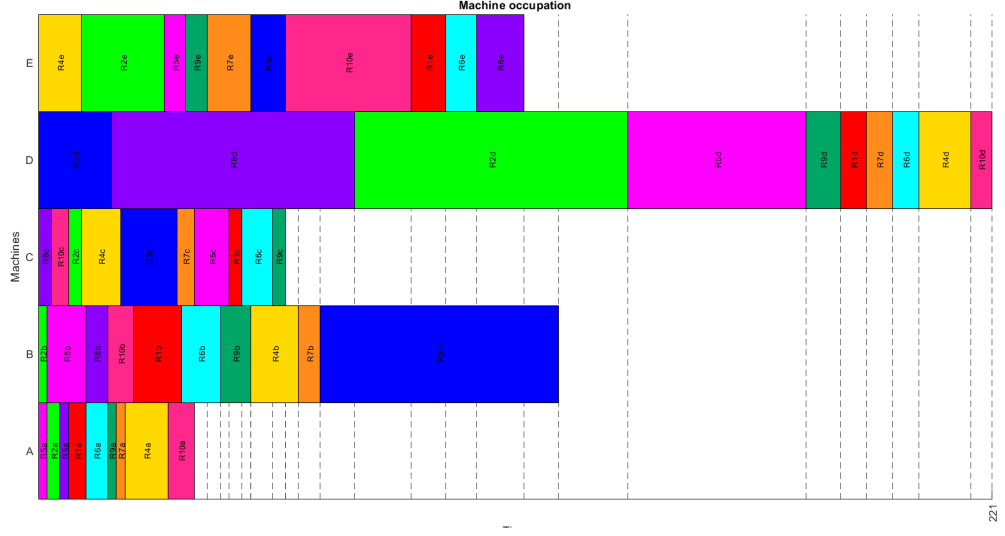


Figure 7.5: 10 recipes scheduling version V3.

Table 7.1 summarizes and compares the performance indicators of the simulations, giving a quantitative meaning to the visual differences.

Table 7.1: Performance indicators for the 10 recipes scenario across the four objective function versions.

Metric	V0	V1	V2	V3
Elaboration time (min)	0.88048	0.84788	0.90135	0.82226
Nlate	0	0	0	0
$\bar{lt}$ (min)	150.2	125.3	150.2	170.1
lt_{max} (min)	221	221	221	221
IoM_A (%)	44.78	2.63	44.78	0
IoM_B (%)	18.79	0	18.79	0
IoM_C (%)	42.57	0	42.57	0
IoM_D (%)	0	0	0	0
IoM_E (%)	23.13	0	23.13	0

The first notable result in Table 7.1 is the $Nlate$ parameter equal to zero

in each version: the completion of the recipes occurs within the total DoP . This means that the mathematical formulation of temporal and physical constraints was successful. To add relevance to this information, also the lt_{max} is consistent across each version, indicating that the changes compel the solver to explore different paths, without degrade the overall optimal processing time.

An interesting insight arises from analyzing the $\bar{lt}$: version 1 shows that the introduction of the discounts pushes the solver to finish the recipes as quickly as possible, achieving an $\bar{lt}$ of 125.3 minutes. Conversely, when combining also transport costs, a tiny difference occurs: the solver attempts to group together sub-recipe to avoid additional costs and the $\bar{lt}$ increases until 170.1 minutes. However, aligning with the company exigences, this increment turned out to be an irrelevant problem, since the primary industrial objective is to minimize the lt_{max} , rather than the $\bar{lt}$.

Regarding machine inactivity, the impact of the optimization logic introduced is significant: the solver succeeds in eliminate almost all dead times, optimizing at best the scheduling and exploiting the resources.

Lastly, the elaboration time indicates the elapsed time for the solver to complete the optimization and it is expressed in minutes. As it can be seen, with a small amount of recipes, it does not exceed a minute.

Version V3 reveals to be the most efficient one. Transport costs are a company's requirement, thus it is not possible to decide whether to consider them. However, simulations are fast to run with a small load, so it allows to perform both V2 and V3, whose only difference is in the presence of the added scheduling strategies, and choose the most suitable. Indeed, it may depend on operational requirements. Version V3 allows to minimize IoM , but at the expense of the $\bar{lt}$, hence version V1 would be a better choice if the focus shifts from completing the overall scheduling to minimize the individual transiting time of barrels.

7.2 Extended recipes list

In order to evaluate the optimization performance over a more realistic scenario, a subset of $n = 30$ recipes is considered. A visual representation of this

case would be chaotic, due to the numerous components involved; therefore, only the performance indexes are reported in Table 7.2.

Table 7.2: Performance indicators for the 30 recipes scenario across the four objective function versions.

Metric	V0	V1	V2	V3
Elaboration time (min)	2.98	4.00	3.03	3.96
Nlate	0	0	0	0
$\bar{lt}$ (min)	142.0	153.37	142.0	150.2
lt_{max} (min)	503	499	503	499
IoM_A (%)	59.48	61.79	59.48	60.61
IoM_B (%)	13.31	18.83	13.31	16.39
IoM_C (%)	10.71	7.16	10.71	5.15
IoM_D (%)	0.80	0	0.80	0
IoM_E (%)	9.54	20.36	9.54	19.95

The system's behavior undergoes significant changes due to the parking spots saturation and computational load. The first observation regards elaboration time: it is clear the not proportional, but exponential increase with respect to the 10-recipes scenario. In particular, the execution times more than triple.

Despite the increased work, parameter Nlate remains zero, proving the robustness of the scheduling model and parking allocation logic.

The trade-off between lt_{max} and $\bar{lt}$ becomes evident in this extended scenario. The optimization strategies make the solver compress the schedule from 503 to 499 minutes, at the expenses of the $\bar{lt}$ which increases from 142 minutes (V0,V2) to 150/153 minutes (V1,V3). This means that to ensure the shortest lead time, the solver keeps the barrels in the system for a longer time, finding a more optimal combination. Actually, the introduction of the transport costs serves as a tie-breaker, as it is seen in versions V1 and V3, the $\bar{lt}$ decreases by grouping operations based on their displacements.

Finally, the IoM percentages exposes the discussed apparent bottleneck

caused by machine D: its inactivity is calculated at hardly 1%. However, the previously analyzed parameters confirm that the solver victoriously optimizes the schedule, despite this inconvenience that does not allow to have an evenly distributed production.

7.3 Robustness validation with an alternative dataset

The results were obtained with a set of recipes coming from a randomized selection of ingredients; it is possible to generate alternative recipes to compare results, as in Table 7.3, keeping the target number set to $n = 30$. The results replicates the previous case study.

The algorithm maintains adherence to the constraints, resulting in zero number of late recipes in all four variations. The lt_{max} is successfully minimized between versions (V0,V2) and (V1,V3) from 481 to 472 minutes. As above, the scheduling compression sacrifices the $\bar{lt}$, that increases also in this scenario from 135.10 to 142 minutes. Furthermore, the IoM confirms the structural limit analysis. Machine D continues to work without hardly any interruption while machine A remains underutilized.

Table 7.3: Performance indicators with a different set of recipes to validate model robustness.

Metric	V0	V1	V2	V3
Elaboration time (min)	2.52	3.43	2.52	3.50
Nlate	0	0	0	0
$\bar{lt}$ (min)	135.10	142.07	135.10	142.77
lt_{max} (min)	481	472	481	472
IoM_A (%)	52.31	65.17	52.31	64.37
IoM_B (%)	4.64	14.44	4.64	10.98
IoM_C (%)	6.67	12.50	6.67	14.68
IoM_D (%)	1.87	0	1.87	0
IoM_E (%)	0.64	15.72	0.64	16.62

This recurring setting across different datasets provides the evidence of the workload imbalance being an inherent characteristic of the original data, and that with these assumptions the ILP scheduling consistently identifies the best possible routing configuration anyway.

The results obtained from this set of recipes illustrate more effectively the necessity of considering both versions, with and without dispatching rules V3 and V2 (maintaining transport costs as a mandatory term in the objective function). Depending on the operational objective, version V2 provides a better configuration when analyzing the $\bar{lt}$ and IoM , while version V3 represents a better trade-off if the aim is the minimization of the total completion time. Since the company expressed its aim is the latter one, the most complete version V3 is considered to be the best formulation in this thesis.

7.4 Robustness validation with a different parking spot capacity

To study the optimal number of parking spots, besides the original requirement of $n_p = 6$, a few simulations have been conducted. By changing the

7.4. ROBUSTNESS VALIDATION WITH A DIFFERENT PARKING SPOT CAPACITY63

parameter in the initial script `Logic_Configuration` it is possible to easily modify the number.

This simulation analyzes a specific subset of 20 recipes, based on V3 as the best-performing version of the cost function.

Table 7.4: Performance parameters comparison for a 20 recipes subset under varying parking slot capacities.

Metric	8 Slots	6 Slots	4 Slots	2 Slots
Nlate	0	0	0	0
$\bar{lt}$ (min)	160.20	144.20	123.55	102.60
lt_{max} (min)	301	301	301	307
IoM_A (%)	67.47	70.87	73.20	75.46
IoM_B (%)	30.32	35.02	40.08	44.60
IoM_C (%)	0.00	0.00	0.00	0.00
IoM_D (%)	0.00	0.00	0.00	8.47
IoM_E (%)	0.00	5.31	15.02	22.15

The above table shows the capability of the algorithm to adapt to changes. In all the scenarios, there are not recipes finishing after their deadline, so the optimization rearranges the scheduling in an adequate manner. While the total completion time suffers from an increase of a few minutes, the individual transiting time of each barrel is significantly reduced from 160.20 to 102.60 minutes. Lastly, due to the scarcity of parking slots, the IoM increases, but the list of recipes is successfully finalized.

In conclusion it is possible to confirm that the initially set number of parking spot is appropriate; although the 8-spots option decreases the IoM , the overall performance does not change, the lt_{max} remains 301 minutes, thus the spatial limitation of the physical layout is not an obstacle for the optimization success.

Chapter 8

Conclusions and Future work

This thesis addressed the scheduling optimization of an industrial robotic dosing system, based on a real-world scenario proposed by *Logic S.r.l.*. The main objective was to minimize the total completion time of a set of recipes, while respecting physical limitations of the plant, the availability of machines, the limited capacity of the parking spots, and time constraints associated with the barrels.

To achieve this objective, an Integer Linear Programming approach was developed and implemented in MATLAB. The mathematical formulation had to represent the characteristics of the system, as the allocation of barrels to machines, the integrity of the individual barrels, and the management of the available time windows to complete the processes through dedicated penalty variables. Furthermore, the cost function was extended with two dispatching rules, shortest processing time and most work remaining, together with the transport costs, associated with the robot movements.

The simulations results demonstrated that the developed model can successfully manage the system's constraints and find the optimal schedule. The tested scenarios included different sets of recipes and different parking spots capacity, proving that the number of late recipes remained equal to zero. This result confirms the effectiveness of the proposed formulation of respecting the temporal constraints. Moreover, the algorithm proved to adapt to different number of parking spots, continuing to find the optimal solution.

The comparison between the different versions of the objective function

highlighted the importance of utilizing an appropriate strategy. On one hand, the introduction of the dispatching rules helped the solver recognize the best scheduling by avoiding machines to be used as parking spots. On the other hand, the introduction of transport costs had a limited impact on the overall results, especially in the smaller scenario. However, it provided an additional criterion to choose among scheduling combinations, proving that even secondary aspects can contribute to the definition of the optimal result.

The simulations regarding a bigger scenario, further demonstrated the benefits of the proposed approach, even when using a different recipes composition. The reductions in completion time are limited, but they confirm that the implemented strategies improve the system's performance. At the same time, the results highlighted a trade-off between the total Lead time and the Average lead time of individual barrels. Minimizing the overall duration of the performance does not necessarily mean minimizing the individual time spent by a barrel inside the system.

Another important aspect is the influence of the parking spots number. In the analyzed 20-recipes scenario, the total completion time remained the same with four, six and eight spots, while increased with two spots. Reducing the parking spots caused the increase in machine inactivity, since resources had to be exploit to rest barrels between operations. These simulations validate the company's choice to include six parking spots, since increasing the number would cause a negligible reduction in machines inactivity and the violation of space constraints. Nevertheless, the best configuration can depend on specific production requirements.

Despite the results, some limitations must be noted. The proposed system is modeled as a static and deterministic problem, where information is known beforehand and unexpected events are not considered. Furthermore, the recipes used were generated through a randomized selection of ingredients, rather than being based on real batches. This choice allowed the algorithm to be tested under different conditions, but a more realistic dataset would provide a stronger proof of applicability to real industrial scenarios. Finally, due to its combinatorial and NP-hard nature, the algorithm becomes more demanding in terms of computational load as the number of recipes increases.

Future work could focus on some improvements. A first development

could be to extend the dataset by introducing a larger number of ingredients and recipes, to make the scenario more realistic. In particular, the use of real recipes would allow to consider production requirements more accurately, and obtain a more reliable evaluation of the performance.

A further extension would be to consider a system equipped with two robots. The single-robot configuration may imply limitations due to its availability, while two robots could increase the flexibility of barrels handling. However, the modification would lead to additional constraints to avoid conflicts between the two tools, or even the formulation of another optimization problem to manage the collaboration between them.

Finally, following the suggestions proposed by *Logic S.r.l.*, the model could be extended through a more dynamic management. A development would concern the advanced management of duplicated resources across different machines. In this scenario, if an ingredient runs out in one machine, the algorithm should be able to dynamically redirect the barrels to an alternative machine providing the same ingredient. Moreover, the system could be improved to manage temporary stock-outs: if an ingredient becomes completely unavailable, the algorithm should prioritize recipes that do not require it, or continue the production until the finished ingredient is the only one missing. Lastly, the framework could integrate strategies to handle unexpected events, such as sudden machine breakdowns or variations in processing times, adjusting the schedule based on the real-time state of the plant.

In conclusion, this thesis provides a mathematical approach for the scheduling optimization of a dosing system with limited parking spots. The results demonstrated the effectiveness of the ILP approach in managing the interactions among machines, barrels, parking spots and time constraints, while the introduction of additional scheduling strategies has improved the performance. Although further improvements are necessary to address real industrial requirements, the proposed method is a successful starting point to improve production planning and reduce bottlenecks in automated dosing systems.

Bibliography

- [1] H. Xiong, S. Shi, D. Ren, and J. Hu, “A survey of job shop scheduling problem: The types and models,” *Computers & Operations Research*, vol. 142, p. 105731, 2022.
- [2] S. Sršen and M. Mernik, “A JSSP solution for production planning optimization combining industrial engineering and evolutionary algorithms,” *Computer Science and Information Systems*, vol. 18, no. 1, pp. 349–378, 2020. DOI: 10.2298/CSIS201009058S
- [3] C. A. Floudas and X. Lin, “Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications,” *Annals of Operations Research*, vol. 139, pp. 131–162, 2005.
- [4] M. Imanparast, “Challenges and strategies for tackling NP-hard problems,” in *9th International Conference on Combinatorics, Cryptography, Computer Science and Computation (I4C)*, Bojnord, Iran, 2024.
- [5] S. Q. Liu, E. Kozan, M. Masoud, Y. Zhang, and F. T. Chan, “Job shop scheduling with a combination of four buffering constraints,” *International Journal of Production Research*, vol. 56, no. 9, pp. 3274–3293, 2018. DOI: 10.1080/00207543.2017.1401240
- [6] Z. Lissioued, S. Ourari, and H. Ait Haddadène, “A milp model to minimize total tardiness in a job shop scheduling problem with blocking and no-wait constraints,” *RAIRO - Operations Research*, vol. 59, pp. 2545–2558, 2025. DOI: 10.1051/ro/2025109
- [7] H. Fakhravar, “Combining heuristics and exact algorithms: A review,” *arXiv preprint arXiv:2202.02799*, 2022. DOI: 10.48550/arXiv.2202.02799

- [8] A. Arisha, P. Young, and M. El Baradie, “Job shop scheduling problem: An overview,” in *International Conference for Flexible Automation and Intelligent Manufacturing (FAIM)*, Dublin, Ireland, 2001, pp. 682–693.
- [9] D. C. Hajariwala, S. S. Patil, and S. M. Patil, “A review of metaheuristic algorithms for job shop scheduling,” *Engineering Access*, vol. 11, no. 1, pp. 65–67, 2025.
- [10] C. Rajendran and O. Holthaus, “A comparative study of dispatching rules in dynamic flowshops and jobshops,” *European Journal of Operational Research*, vol. 116, pp. 156–170, 1999.
- [11] P. D. D. Dominic, S. Kaliyamoorthy, and M. S. Kumar, “Efficient dispatching rules for dynamic job shop scheduling,” *The International Journal of Advanced Manufacturing Technology*, vol. 24, pp. 70–75, 2004. DOI: 10.1007/s00170-002-1534-5
- [12] R. A. Liaqait, S. Hamid, S. S. Warsi, and A. Khalid, “A critical analysis of job shop scheduling in context of industry 4.0,” *Sustainability*, vol. 13, no. 14, p. 7684, 2021. DOI: 10.3390/su13147684
- [13] P.-Q. PAN, *Integer Linear Programming (ILP)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 275–294, ISBN: 978-3-642-40754-3. DOI: 10.1007/978-3-642-40754-3_10
- [14] J. Buske and T. Jendle, “Integer linear programming applied to production planning,” M.S. thesis, Chalmers University of Technology, Gothenburg, Sweden, 2024.
- [15] K. Genova and V. Guliashki, “Linear integer programming methods and approaches – a survey,” *Cybernetics and Information Technologies*, vol. 11, no. 1, 2011.
- [16] S. Boyd and J. Mattingley, “Branch and bound methods,” Notes for EE364b, Stanford University, Winter 2006-07, Mar. 2007.

Acknowledgments

As the final project of my academic career comes to an end, I would like to thank my supervisor Prof. Toffanin Chiara, who had the patience to guide me through every inconvenience. She had a crucial role in assisting my progress and thanks to her suggestions I succeeded in completing my most important work by now.

Thanks to my colleagues. They helped me getting though the chaos during the last three years and a half. We shared not only notes, but also afternoons, days and weekends trying to survive.

Thanks to my friends, that have always been there, no matter what. Thanks for the massive comprehension when I disappeared in a black hole for weeks during exams season and for asking me whatever they want thinking I am a dictionary or an encyclopedia.

Thanks to my boyfriend, he found me a little bit missing and saw something in me that I later on acknowledged. This achievement is mainly because of his support and I will forever be thankful to him.