UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA INDUSTRIAL

SYNERGY BETWEEN MACHINE LEARNING AND OPTIMIZATION MODELS: FORECASTING OUTCOMES IN PARALLEL MACHINES SCHEDULING PROBLEM

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN GESTIÓN DE OPERACIONES

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERA CIVIL INDUSTRIAL

CATALINA ALEJANDRA MIRANDA PIZARRO

PROFESOR GUÍA:
SEBASTIÁN RÍOS PÉREZ

PROFESOR CO-GUÍA:
PABLO CLEVELAND ORTEGA

MIEMBROS DE LA COMISIÓN:
FERNANDO ORDOÑEZ PIZARRO
EMILIO CARRIZOSA PRIEGO

SANTIAGO DE CHILE
2025

RESUMEN DE LA TESIS PARA OPTAR AL GRADO DE:
MAGÍSTER EN GESTIÓN DE OPERACIONES
Y MEMORIA PARA OPTAR AL TÍTULO DE:
INGENIERA CIVIL INDUSTRIAL
POR: CATALINA ALEJANDRA MIRANDA PIZARRO
FECHA: 2025
PROF. GUÍA: SEBASTIÁN RÍOS PÉREZ
PROF. CO-GUÍA: PABLO CLEVELAND ORTEGA

SINERGIA ENTRE APRENDIZAJE AUTOMÁTICO Y OPTIMIZACIÓN: PREDICCIÓN DE RESULTADOS EN EL PROBLEMA DE CALENDARIZACIÓN DE MÁQUINAS PARALELAS

Implementar un modelo de optimización para programar pedidos en la industria del hormigón, con la meta de minimizar atrasos, conlleva un alto costo computacional debido a la naturaleza NP-hard del problema. Esto limita su aplicabilidad en entornos operativos que requieren respuestas en tiempo real. Si bien las heurísticas ofrecen tiempos de ejecución más bajos, no garantizan soluciones de alta calidad dentro del marco temporal requerido. Esta tesis explora un enfoque híbrido que combina Investigación de Operaciones (OR) y Aprendizaje Automático (ML) para aproximar los resultados de los modelos de optimización, buscando preservar la calidad de las soluciones y reducir drásticamente el esfuerzo computacional.

La investigación consta de dos etapas. En la primera, se diseña una metaheurística que integra heurísticas constructivas, para generar soluciones iniciales factibles, y heurísticas de mejora, para refinarlas mediante reasignación de tareas. Su desempeño se contrasta con una formulación de Programación Lineal Entera Mixta (MILP) y con una asignación aleatoria, utilizando datos sintéticos que simulan una semana de pedidos en una planta de hormigón. La metaheurística alcanza soluciones con un gap promedio de solo 8.3% en comparación con el modelo exacto, lo que representa una mejora promedio de 2968.8% respecto a la asignación aleatoria, requiriendo apenas un 10% del tiempo de ejecución del MILP.

En la segunda etapa, estas soluciones generadas por la metaheurística se usan para entrenar un modelo predictivo de ML con una estructura de ensamble híbrido (Random Forest, AdaBoost y XGBoost). El modelo de ML aproxima el mapeo entre instancias y valores de la función objetivo, alcanzando un gap promedio de 9.4%. Además, produce predicciones en menos de un segundo, lo que representa una reducción del costo computacional de varios órdenes de magnitud en comparación con el modelo de optimización.

La contribución teórica de esta tesis radica en demostrar que los modelos de ML funcionan como predictores de los resultados de optimización en problemas de calendarización NP-hard, logrando aprender la estructura latente del espacio objetivo del MILP. Esto avanza el campo de ML para la Investigación de Operaciones al mostrar que los datos generados por la optimización pueden aprovecharse para entrenar modelos de ML que entreguen soluciones casi óptimas en tiempo real. Más allá de su relevancia en la industria del hormigón, la metodología propuesta provee un marco generalizable para integrar enfoques de OR y ML en contextos de calendarización de gran escala.

ABSTRACT OF THE THESIS SUBMITTED FOR THE DEGREE OF:
MASTER IN OPERATIONS MANAGEMENT
AND FOR THE TITLE OF:
INDUSTRIAL CIVIL ENGINEER
BY: CATALINA ALEJANDRA MIRANDA PIZARRO
DATE: 2025
ADVISOR: SEBASTIÁN RÍOS PÉREZ
CO-ADVISOR: PABLO CLEVELAND ORTEGA

SYNERGY BETWEEN MACHINE LEARNING AND OPTIMIZATION MODELS: FORECASTING OUTCOMES IN PARALLEL MACHINES SCHEDULING PROBLEM

Implementing an optimization model to schedule orders in the concrete industry, with the goal of minimizing lateness, entails a high computational cost due to the NP-hard nature of the problem. This hinders its applicability in operational settings that require real-time responses. While heuristics offer lower execution times, they do not guarantee high-quality solutions within the required time frame. This thesis explores a hybrid approach that combines Operations Research (OR) and Machine Learning (ML) to approximate the outcomes of optimization models, aiming to preserve solution quality while drastically reducing computational effort.

The research proceeds in two stages. First, a tailored metaheuristic is developed, integrating constructive heuristics, to generate feasible initial solutions, and improvement heuristics, to refine them through task reassignment. The metaheuristic's performance is benchmarked against a Mixed-Integer Linear Programming (MILP) formulation and a random assignment baseline, using synthetic datasets that simulate one week of orders in a concrete plant. Results show that the metaheuristic achieves solutions with an average optimality gap of only 8.3% compared to the exact model, representing an average improvement of 2968.8% relative to random assignment, while requiring merely 10% of the MILP execution time.

In the second stage, these metaheuristic-generated solutions are leveraged to train a predictive ML model with a hybrid ensemble structure (Random Forest, AdaBoost, and XGBoost). The ML model approximates the mapping between problem instances and objective function values, achieving an average gap of 9.4% relative to test and validation data. Crucially, it produces predictions in under one second, representing a computational cost reduction of several orders of magnitude compared to the optimization model.

The theoretical contribution of this work lies in demonstrating that ML models can serve as surrogate predictors of optimization outcomes in NP-hard scheduling problems, effectively learning the latent structure of the MILP objective landscape. This advances the emerging field of Machine Learning for Operations Research by showing that optimization-generated data can be exploited to train ML models that deliver near-optimal solutions in real time. Beyond its practical relevance for the concrete industry, the proposed methodology provides a generalizable framework for integrating optimization and learning-based approaches in large-scale scheduling contexts.

A mi mamá y papá, a quienes amo con todo mi corazón.
Les dedico este y cada uno de mis logros.

Con amor, Catalina

To my mom and dad, whom I love with all my heart.
I dedicate this work and all my achievements to them.

With love, Catalina

Agradecimientos

En primer lugar, quiero agradecer a mis profesores Sebastián y Pablo, sin ustedes esto no habría sido posible. Gracias por confiar en mí, permitirme trabajar en esta investigación y por todas las conversaciones de orientación - de todo tipo - que tuvimos a lo largo del proceso, al igual que todas las oportunidades que me han dado; aprendí muchísimo en cada momento, lo sigo haciendo y espero continuar. Asimismo, quiero agradecer al profesor Charles por abrirme la puerta en mi primera vez como ayudante de un ramo, así como a todos los miembros de equipos docentes que contribuyeron a mi entusiasmo por obtener este doble título.

Quiero agradecer a mi familia, en especial a mi mamá y papá, Olga y Alberto: gracias por su apoyo incondicional, por aguantar mi mal genio cuando estuve estresada, por las conversaciones, por llevarme a todas partes y por los tecitos de madrugadas; valoro todo lo que han hecho por mí desde siempre, los amo con todo mi corazón. También a mi abue, quien me inculcó las matemáticas y siempre me ha dado ánimo y confianza en todo momento. A mi hermano, Carlos, por todo el apoyo, orientación e inspiración por estudiar, que siempre ha infundido en mí. A mi tío Carlos, por guiarme en el mundo de la Ingeniería Industrial y Operaciones, por su apoyo y entusiasmo en cada etapa de mi carrera. A mis primos Felipe y Matías, por apoyarme y alegrarse junto a mí en cada logro.

A Gonzalo, quien conocí en segundo año y se ha convertido en parte fundamental de mi vida. Gracias por todo tu apoyo y amor en todo momento, y por ayudarme en confiar en mí.

A mis amigos, por entender cuando desaparecía por los intensivos, por su apoyo, por alegrarse conmigo por cada logro y por quedarse conmigo en todo este proceso. En particular, a Arantxa, Antonia, Isidora y mis otras amigas del colegio; Diana, Camila, que conocí como mechona y mantuvimos nuestra amistad hasta el día de hoy, ya las tres profesionales, y a todos quienes pasaron por diferentes momentos de mi etapa universitaria, los llevo en una parte especial de mi corazón. A Maca y su familia - que son todos familia para mí -, valoro que estén conmigo todos estos años que nos conocemos.

A mis maestros de ballet y danza contemporánea, Daniela y Carlos: gracias por tolerar que faltara y llegara tarde a ensayos y clases, y aún así, permitir que participara de las funciones. No sé qué habría sido de mí si hubiera dejado de bailar todos estos años. Gracias por permitirme entrar a sus creaciones y corazones, valoro cada palabra, gesto y momento.

A todos quienes nombré y a los que no explícita o implícitamente: gracias. Los quiero y llevo conmigo siempre.

Acknowledgments

In the first place, I want to thank my professors Sebastián and Pablo; without you this would not have been possible. Thanks for trusting me, for allowing me to work on this research and for all the guidance conversations - of every kind - we had throughout the process, as well as for all the opportunities you have given me; I learned immensely from each moment, I still do and I hope to continue doing it. I also want to thank Professor Charles for opening the door to my first time being teaching assistant in a course, likewise to all the members of teaching teams that contributed to my enthusiasm for pursuing this double degree.

I want to thank my family, specially my mom and dad, Olga and Alberto: thanks for your unconditional support, tolerating my bad moods when I was stressed, our conversations, driving me everywhere and the late-night teas; I deeply value everything you have done for me since always, I love you with all my heart. I also want to thank my grandma, who showed me math and has always giving me encouragement and confidence. To my brother, Carlos, for the guidance and inspiration to study and being a better person, which he has always instilled in me. To my uncle Carlos, for guiding me into the Industrial Engineering and Operations world and for his enthusiasm throughout each step of my career. To my cousins Felipe and Matías, for the support and for celebrating each achievement with me.

To Gonzalo, whom I met in my second year and who has become a fundamental part of my life. Thanks for all your support and love at each moment, and for helping me trust myself.

To my friends, for understanding when I disappeared in the study-intensive periods, for celebrating with me each accomplishment and for staying by my side in the whole process. In particular, to Arantxa, Antonia, Isadora and my others school friends; to Diana and Camila, whom I met in freshman year and we keep our friendship until now, the three of us being professionals, and to all those who shared different moments of my university years, I keep each of you in a special place of my heart. To Maca and her family - all of them my family too -, I truly value having you by my side through all these years we have known each other.

To my ballet and contemporary dance teachers, Daniela and Carlos: thanks for tolerating my absences and late arrivals to classes and rehearsals, and still allowing me to participate in shows. I do not know what would have become of me without dancing all these years. Thanks for letting me be part of your creations and hearts, I appreciate each word, gesture and moment.

To all whose I named and to those I did not mention explicitly or implicitly: thank you. I love you and I always carry you with me.

Table of Content

1. Background **1**
- 1.1. Introduction and Motivation . . . 1
- 1.2. Objectives . . . 3
  - 1.2.1. General Objective . . . 3
  - 1.2.2. Specific Objectives . . . 3
- 1.3. Expected Results . . . 3
- 1.4. Scope and Limitations . . . 3
- 1.5. Structure . . . 4

2. Theoretical and Methodological Framework **5**
- 2.1. Theoretical Framework I: Optimization . . . 5
  - 2.1.1. Key Concepts . . . 5
  - 2.1.2. Literature Review and Related Work . . . 9
  - 2.1.3. Identified Research Gaps . . . 12
- 2.2. Theoretical framework II: Machine Learning . . . 12
  - 2.2.1. Key Concepts . . . 12
  - 2.2.2. Literature Review and Related Work . . . 15
  - 2.2.3. Identified Research Gaps . . . 16
- 2.3. Methodological Framework . . . 19
  - 2.3.1. Methodology . . . 19
  - 2.3.2. Techniques . . . 22

3. Optimization solution **27**
- 3.1. Dataset . . . 28
  - 3.1.1. Data Processing . . . 29
  - 3.1.2. Instances . . . 29
- 3.2. Formulations . . . 30
  - 3.2.1. Optimization Model . . . 30
  - 3.2.2. Construction Heuristics . . . 33
  - 3.2.3. Improvement Heuristics . . . 36
  - 3.2.4. Metaheuristic . . . 42
- 3.3. Implementation . . . 44
  - 3.3.1. Optimization Model . . . 44
  - 3.3.2. Heuristics . . . 45
  - 3.3.3. Solution Validation and Feasibility Check . . . 46
- 3.4. Results . . . 47
  - 3.4.1. Schedules . . . 47

3.4.2. Lateness . . . 50
3.4.3. Execution Time . . . 54
3.4.4. Lateness versus runtime . . . 58
3.5. Conclusions of the Optimization Solution . . . 59

4. Machine Learning Solution **61**
4.1. Input Dataset . . . 62
4.1.1. Rows of the Dataset: Experiments . . . 62
4.1.2. Columns of the Dataset: Features . . . 63
4.2. Machine Learning Model . . . 65
4.2.1. Classification and Regression Models . . . 65
4.2.2. Building the Output . . . 66
4.3. Implementation . . . 67
4.3.1. Data Processing . . . 67
4.3.2. Model Fitting . . . 68
4.3.3. Test and Validation . . . 70
4.4. Results . . . 70
4.4.1. Performance Metrics of the Predictions . . . 70
4.4.2. Execution Time . . . 73
4.4.3. Comparison with Optimization Models . . . 73
4.5. Conclusions of the Machine Learning Solution . . . 74

5. Discussion and Conclusions **76**
5.1. Discussion . . . 76
5.2. General Conclusions . . . 77
5.3. Future Work . . . 78

Bibliography **79**

Annexes **81**
A. Preliminary Analysis . . . 81
B. Dataset used for the Optimization Model and Heuristics . . . 84
C. Random Feature Selection Results for Each Model with Metrics . . . 87
D. Parameter Tuning and Sensitivity Analysis Results . . . 91

List of Tables

2.1. Comparison of key papers addressing the RMC Dispatching Problem 11
2.2. Confusion matrix of binary classification 13
2.3. Comparison of key papers addressing the integration between ML and Optimization fields 18
2.4. Specific objectives linked with their expected results, including the section in which it will be developed 21
3.1. Dataset columns used for the optimization model and heuristics 28
3.2. Initial instances 30
3.3. New instances creation criteria 30
3.4. Sets and parameters used in the optimization model 32
3.5. Decision variables of the optimization model 32
3.6. Ordering and assignment criteria of construction heuristics 33
3.7. Parameters used in Gurobi for the optimization model 44
3.8. Parameter values used for model implementation 44
3.9. Parameter values used in the heuristics implementation 45
3.10. Average total lateness obtained for each day after applying the optimization model and heuristics 51
3.11. Average runtime in seconds for each method, after being executed in all instances of each day 54
4.1. Experiments used to create rows of the ML input dataset 63
4.2. Features included in the ML input dataset 64
4.3. Values for total lateness in ML output 66
4.4. Division of ML input dataset into training, test, and validation subsets 67
4.5. Number of rows per class in the original (unbalanced) and balanced datasets . 67
4.6. Use of balanced and unbalanced datasets during training and evaluation 68
4.7. Variables selected using Random Feature Elimination with Cross-Validation (RFECV) for the Machine Learning model 69
4.8. Parameter tuning grid for each ML model 69
4.9. Parameter tuning resulting values for each part that integrates Machine Learning model 69
4.10. Performance metrics for each Regressor–Classifier combination 70
4.11. Performance metrics for each Regressor–Classifier combination, grouping instances by 'Day' and 'Size' 71
4.12. Runtime of proposed Machine Learning approach, in seconds 73
4.13. Performance of ML model compared to optimization and heuristic methods . . 73
A.1. Linear Regression R^2 results across variable sets 81
A.2. Variable Sets Used in the Experiments 81
A.3. F1-scores for Random Forest and SVM across variable sets 82

A.4. R^2 performance for SVR and Random Forest across variable sets 83
B.1. Attributes of the original dataset, used to implement the optimization model and heuristics 86
C.1. Cross-validation RMSE results for regression models 88
C.2. Cross-validation RMSE results for classification models 90
D.1. Parameter tuning results for XGBoost Classifier 91
D.2. Parameter tuning results for AdaBoost Classifier 91
D.3. Parameter tuning results for Random Forest Classifier 91

List of Figures

1.1. Diagram that illustrates the problem . . . 1
1.2. Solution approaches for scheduling problems . . . 2
2.1. Diagram showing how Adaboost classifier works . . . 15
2.2. Methodology of the thesis . . . 20
2.3. Classical decision-making process (Talbi, 2009, p. 2) . . . 22
2.4. Guidelines for solving a given optimization problem using metaheuristics (Adapted from Talbi, 2009, p. 77) . . . 23
2.5. DS process flow (Adapted from Kotu & Deshpande, 2019) . . . 24
2.6. Example diagram of 10-fold cross-validation (Zhou, 2021) . . . 25
3.1. Diagram of the optimization solution. . . . 27
3.2. Uses of the initial data set . . . 28
3.3. Example of time variables transformation to continuous values . . . 29
3.4. Flow diagram of the optimization model . . . 31
3.5. Flow diagram of the construction heuristics . . . 33
3.6. Example of different approaches to schedule dispatch trains between the construction heuristics RH1, RH2 and R . . . 34
3.7. Flow diagram of the Improvement Heuristics . . . 37
3.8. Possible task movements allowed in heuristic *IH1* . . . 37
3.9. Possible task movements allowed in heuristic *IH2* . . . 40
3.10. Flow diagram of the Metaheuristic . . . 42
3.11. Example of Gantt Chart of Jobs, obtained for $Tuesday\_50$ using the Metaheuristic 47
3.12. Example of Gantt Chart of Dispatches, obtained for $Tuesday\_50$ using the Metaheuristic . . . 48
3.13. Example of Gantt Chart of Jobs from the client's perspective, obtained for $Tuesday\_50$ using the Metaheuristic . . . 48
3.14. Comparison between optimization model and metaheuristic schedules, obtained for instance $Monday\_60$ . . . 49
3.15. Comparison of the objective function value, Total Lateness, obtained after executing the different methods on each instance . . . 52
3.16. Comparison of the Maximum Lateness of tasks in schedules obtained after executing the different methods on each instance . . . 53
3.17. Comparison of the average Total Lateness, obtained after executing the different methods on each instance, grouped by subset range . . . 53
3.18. Comparison of the average runtime, obtained after executing the different methods on each instance, grouped by instance size . . . 55
3.19. Comparison of the runtime obtained after executing the different methods on each instance . . . 56

3.19. Comparison of the runtime obtained after executing the different methods on each instance . . . 57
3.19. Comparison of the runtime obtained after executing the different methods on each instance . . . 57
3.20. Lateness versus Runtime performance for each method . . . 58
4.1. Diagram of the proposed Machine Learning solution . . . 61
4.2. Diagram of the dataset creation process for the ML stage . . . 62
4.3. Flow diagram of the ML hybrid approach . . . 65
4.4. Diagram of the ML output construction . . . 66
4.5. Average RMSE calculated using test and validation sets . . . 72
4.5. Average RMSE calculated using test and validation sets . . . 72
A.1. Comparison of Random Forest (RF) and Support Vector Machine (SVM) classifiers performance, using F1-score . . . 82
A.2. Performance comparison of Random Forest (RF) and Support Vector Machine (SVM) regressors, using RMSE . . . 83
D.1. Random Forest Classifier Sensitivity Analysis . . . 92
D.2. AdaBoost Classifier Sensitivity Analysis . . . 92
D.3. XGBoost Classifier Sensitivity Analysis . . . 92
D.4. Random Forest Regressor Sensitivity Analysis . . . 93
D.5. AdaBoost Regressor Sensitivity Analysis . . . 93
D.6. XGBoost Regressor Sensitivity Analysis . . . 93

Chapter 1

Background

1.1. Introduction and Motivation

Having an efficient schedule for tasks is important, for example, to reduce costs or to provide a better level of service. However, generating said schedule is not an easy problem to solve. In particular, for the concrete industry, there are many other extra factors to consider when arranging dispatch orders, such as the perishability of concrete, limited plant and trucks capacity, strict time windows, the existence of dispatch trains, among others. Creating an efficient schedule for these orders may prove to have a real impact on the business, but the question remains, how can such a schedule be obtained?

The problem consists of assigning, ordering and determining the starting time of purchase orders in a concrete plant, to maximize the provided service level; that is, to minimize the total lateness of deliveries. Picture there is a set of 10 feeding hoppers, acting as machines, and the orders correspond to the tasks that should be processed in them. Additionally, these orders can be grouped into dispatch trains, which are characterized by sharing the same delivery destination, a subset of feasible machines where they can be processed, and a certain spacing time between concrete deliveries to the client that must be adhered to, as illustrated in Figure 1.1. It is important to note that each task is characterized by its own release, processing, delivery times and due date.

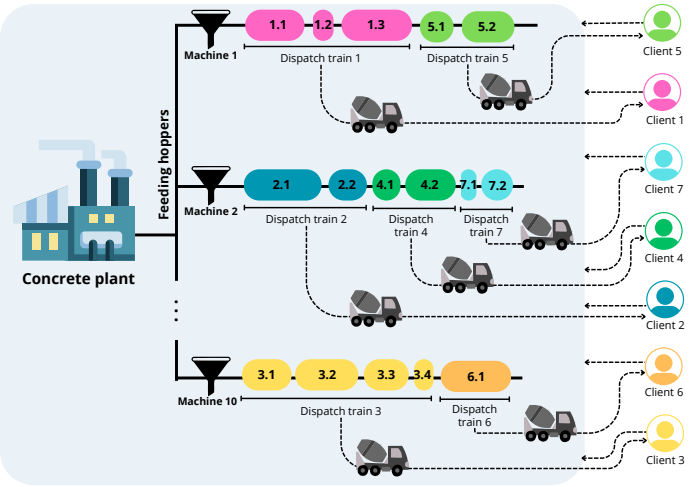


Figure 1.1: Diagram that illustrates the problem

Most of the time, scheduling problems are solved using optimization models. This approach allows to obtain optimal solutions, but the time required to reach these solutions grows exponentially as the complexity of the problem is increased. In concrete industry, "large scale dispatching problems are technically characterized as classical NP-Hard problems, which means that they can not be solved optimally with existing methods in a polynomial time" (Maghrebi, Periaraj, Waller, & Sammut, 2014b). Additionally, machine scheduling problems of this type "can be solved by polynomial-depth backtrack search and thus are members of NP" (Lenstra, Kan, & Brucker, 1997). Therefore, the computational time required to solve these NP-hard problems makes its application in real time incompatible.

Heuristics are an alternative solution methoddue to their usually lower execution time. However, they can not guarantee an optimal solution, allowing infeasible or sub-optimal solutions to be obtained as output of these methods. Some measures can be taken to address this issue, such as applying a post-processing routine to make infeasible solutions feasible. However, this becomes counterproductive for applications that require real-time responses, as the routine inevitably increases the method's execution time.

Taking the above into consideration, it is necessary to develop a new method to solve the scheduling problem for such applications, as shown in Figure 1.2.

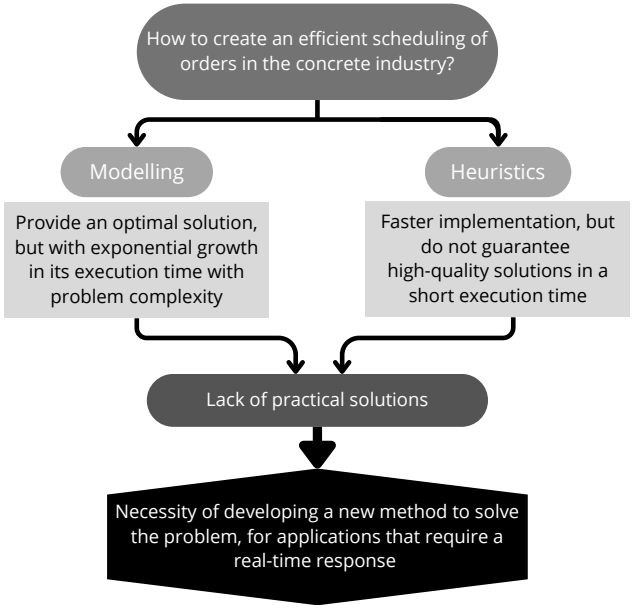


Figure 1.2: Solution approaches for scheduling problems

Nowadays, Machine Learning tools are widely used in different applications. In particular, developing models that can be trained to predict values. Therefore, using these tools may be a good approach to solve the problem.

In this thesis, the possibility of predicting the results of an optimization model, that is the value of its objective function, using Machine Learning tools is investigated. If proven feasible, this approach would allow to obtain outputs, with an acceptable loss of accuracy, within significantly reduced execution times. This, in turn, would enable the efficient generation of solutions, making the approach suitable for applications that require real-time updates.

1.2. Objectives

1.2.1. General Objective

Given the aforementioned, the general objective of the thesis is defined as:

Obtain the objective function value for the scheduling problem of orders in a concrete plant that requires real-time updates with a short execution time and acceptable loss, combining tools from the fields of Machine Learning and Optimization.

1.2.2. Specific Objectives

Development of this thesis can be divided into two stages, each of them with their specific objectives associated, which are detailed below.

- SO1: Design, develop, and evaluate a Mixed-Integer Linear Programming model tailored to the specific characteristics of the problem.
- SO2: Design, develop, and evaluate a metaheuristic method, comprising construction heuristics for generating initial solutions and improvement heuristics for approaching the optimum.
- SO3: Design and build a comprehensive dataset for training Machine Learning models, by generating and collecting solutions from the metaheuristic under different instances with and without perturbations.
- SO4: Design, develop, and evaluate a Machine Learning model that, by leveraging and integrating results obtained from heuristic approaches, predicts the optimal objective function value of the scheduling model, achieving acceptable accuracy with reduced execution time.

1.3. Expected Results

The research question to be responded is if it is possible to train a Machine Learning model to predict the relations between the space of feasible solutions and the objective function of a Mixed-Integer Linear Programming model. If so, it is expected to compare the quality and efficiency of obtaining solutions after modifications in the input, comparing with the use of heuristics and the optimization model itself.

1.4. Scope and Limitations

To solve the scheduling problem, the following assumptions are made:

- Trucks used to dispatch tasks from the plant to customers are assumed to be an unlimited and always available resource. Consequently, they do not constitute a potential bottleneck in the operation. Therefore, the only possible source of bottlenecks in the system lies in the feeding hoppers, which are modeled as the machines.
- A single machine type is considered. Therefore, while there are feasible and unfeasible machines to process a task, they all belong to the same type. If additional machine

types are required, defining a machine types set and indexing accordingly would suffice. Nevertheless, it is important to note that each machine has a task specific processing time, which may differ among the feasible machines.

- Uncertainty and processing interruptions are not considered, meaning the problem is defined as deterministic.

Due to computational constraints, it is not feasible to execute the algorithms (MIP optimization model, a series of construction and improvement heuristics, and a GRASP metaheuristic) using the entire initial dataset (which comprises one week's worth of orders). Therefore, the dataset is subdivided into smaller subsets for the experiments.

1.5. Structure

Given the nature of the problem, this thesis covers two major areas of research: Optimization and Machine Learning. Consequently, it is structured to address each area separately, followed by their integration.

First of all, with the aim of achieving a better understanding of the problem, a literature review is carried out in Chapter 2, including a theoretical framework of relevant concepts and related works. In addition, the methodology used to reach each of the objectives mentioned previously is explained in detail.

Chapter 3 describes the datasets, optimization model, and heuristics used, along with their implementation results. Chapter 4 follows a similar structure, but focuses on the Machine Learning model.

As for Chapter 5, it integrates both areas covered in the thesis. Thereby, the discussion of the work and the main conclusions of the research, as a whole, are presented. Additionally, future work is also included in this chapter.

Chapter 2

Theoretical and Methodological Framework

This chapter is structured into three main sections. Each of the first two sections is dedicated to one of the core research areas addressed in this thesis: Optimization and Machine Learning. In the initial subsections of each part, the key concepts and definitions relevant to the respective field are introduced. These are followed by a review of the most pertinent literature and related work. The last section presents a detailed description of the research methodology and techniques adopted for this study.

2.1. Theoretical Framework I: Optimization

2.1.1. Key Concepts

Ready-Mixed Concrete Dispatching Problem

The Ready-Mixed Concrete (RMC) Dispatching Problem arises in the concrete industry and, in brief, involves scheduling and dispatching concrete orders from a producing plant to customers within specified time windows, aiming to minimize delivery lateness. In fact, it can be described as "delivering a specified amount of concrete to customers from depots using capacitated trucks. At each location in the transportation network, the trucks are expected to start and leave within the specified time window that is required to load and unload the concrete. Furthermore, a penalty is incurred for not delivering concrete to a customer" (Maghrebi, Periaraj, Waller, & Sammut, 2016).

Scheduling Problems

In (Blazewicz, Ecker, Pesch, Schmidt, & Weglarz, 2001), there is a complete description of scheduling problems. Specifically, they noted that this type of problems is characterized by:

- T: Tasks
- P: Processors, or machines
- R: Additional resources

Taking into consideration that "each task is to be processed by at most one processor at a time [...] and each processor is capable of processing at most one task at a time" (Blazewicz

et al., 2001), they proposed that scheduling problems may be summarized as "assign processors from P and (possibly) resources from R to tasks from T in order to complete all tasks under the imposed constraints".

It is also important to mention that they characterized a task $T_j \in T$ by a series of components. The most relevant for this thesis among them are the following:

1. Vector of processing times p_j: required time for processor P_i to process task T_j
2. Arrival time (ready time) r_j: time when task T_j becomes ready to be processed
3. Due date d_j: time when task T_j is required to be completed
4. Weight (priority) w_j: relative urgency of task T_j

Unrelated Parallel Machines Scheduling Problem

According to (Brucker, 2007), the Unrelated Parallel Machines Scheduling Problem (UPMSP) corresponds to a class of scheduling problems, in which, as the name suggests, a set of machines can process tasks in parallel and unrelated form. This means that machines operate independently from one another, and tasks do not need to be processed sequentially across machines. Consequently, the problem reduces to "assign jobs i to positions k on machines j". However, each possible assignment is associated with a processing time p_{ij} and a cost kp_{ij}, which must be considered in the scheduling decision.

There exist multiple mathematical formulations to solve this problem. However, (Brucker, 2007) proposes a typical three-step approach:

1. Compute the processing times for each machines in an optimal schedule.
2. Solve the scheduling problem by optimizing a given metric, such as *Total lateness*.
3. Construct the optimal schedule based on the optimal solution obtained in the previous step.

Lateness

Given an schedule, there are many parameters and performance measures that can be calculated. Lateness is one of them, and it represents how much time a task is late, defined by the difference between the time when it is completed and its due date, as can be seen in Equation 2.1 (Blazewicz et al., 2001).

$$\text{Lateness } L_j = \text{Completion time } C_j - \text{due date } d_j \tag{2.1}$$

When solving scheduling problems using optimization modeling, it is usual to define Lateness as a variable included in the objective function.

Optimization modeling

When searching for an optimal solution to a problem is a necessity, a good option to solve it is through an optimization model. This means defining decision variables and a series of equations to represent the problem, which is generally solved with the help of a computer.

As presented in (Boyd & Vandenberghe, 2009), the principal components of an optimization problem are:

- **Optimization variables:** represent the decisions to be made in the model and are explored during the optimization process to identify the optimal solution. For example, they may represent the quantity to be transported from one location to another.
- **Objective function:** an equation that defines the goal of the optimization problem, such as minimizing cost or maximizing profit.
- **Constraint functions:** a series of equations that impose limitations or requirements on the solution space, such as capacity or resource availability constraints.

Additionally, to implement this type of problem it is necessary to include parameters, which are fixed pieces of information that must be considered but cannot be modified. These parameters are usually derived from the specific context of the problem.

Linear Programming Problem

According to (Bertsimas & Tsitsiklis, 1997), a Linear Programming (LP) problem is characterized by defining:

- Linear cost function
- Set of linear equality and inequality constraints
- Continuous decision variables

Furthermore, it can be written, in a general form, such as the one displayed in Equation 2.2.

$$\begin{aligned} &\textit{minimize} && c'x \\ &\textit{subject to} && a_i'x \geq b_i, i \in M_1 \\ &&& a_i'x \leq b_i, i \in M_1 \\ &&& a_i'x = b_i, i \in M_1 \end{aligned} \tag{2.2}$$

Where x are the optimization or decision variables; c is the vector of costs, thus, $c'x$ is the objective or cost function; and the set of inequalities and equality functions are the linear constraints.

Mixed-Integer Linear Programming Problem

Mixed-Integer Linear Programming (MILP) problems have the same characterization as common Linear Programming problems, except that the optimization variables can be either integer or continuous (Bertsimas & Tsitsiklis, 1997).

Feasible, Infeasible and Optimal Solutions

In (Bertsimas & Tsitsiklis, 1997), a series of relevant definitions about possible solutions obtained after solving the problem are presented. Following the notation used in 2.2, "a vector x satisfying all the constraints is called a feasible solution or feasible vector". Then, it is possible to conclude that an infeasible solution does not satisfy all the constraints of the problem. Furthermore, given a minimization problem as the one presented in 2.2, "a feasible solution x^* that minimizes the objective function (that is, $c'x^* \leq c'x$, for all feasible x) is called an optimal feasible solution or, simply, an optimal solution".

Heuristics

Heuristics are "strategies using readily accessible, loosely applicable information to control problem solving" (Martí, Pardalos, & Resende, 2018). According to the same book, "any given solution/heuristic is not guaranteed to be optimal but heuristic methodologies are used to speed up the process of finding satisfactory solutions where optimal solutions are impractical". Thus, they provide a faster way to approach a practical optimal solution.

Metaheuristics

Metaheuristics are "methods used to design Heuristics and may coordinate the usage of several Heuristics toward the formulation of a single method" (Martí et al., 2018). According to the same book, some of the principal characteristics of this type of methods include a higher computational cost compared to traditional heuristics, and the ability to "reduce the effect of the myopic nature of heuristic approaches by diversification mechanisms".

Greedy Randomized Adaptive Search Algorithm

The Greedy Randomized Adaptive Search Algorithm (GRASP) is an "iterative two phase search method" (Marinakis, Migdalas, & Pardalos, 2006), where each phase consists of:

1. Construction phase: building up an initial feasible solution
2. Local search phase: improving the initial solution obtained in the previous phase, using a local search procedure to find a local optima

These two phases are executed in each iteration, and then, "the final result is simply the best solution found over all iterations" (Marinakis et al., 2006).

An example of a GRASP procedure for minimization is shown in Algorithm 1 next (Resende & Ribeiro, 2013), where it can be seen the two phases described previously. Additionally, it is important to mention that GRASP algorithm searches in each step, updating the best and possible solutions, and with a random component to increase the probability of finding better global solutions.

Algorithm 1 Example of GRASP heuristic for minimization

Input: $MaxIterations$, $random\_seed$
Output: Proposed solution S^*
Set $f^* \longrightarrow \infty$
for $k = 1, ..., MaxIterations$ **do**
    $S \longrightarrow$ GreedyRandomizedAlgorithm($random\_seed$)
    if S is not feasible **then**
        $S \longrightarrow$ RepairSolution(S)
    end if

    if $f(S) < f^*$ **then**
        $S^* \longrightarrow S$
        $f^* \longrightarrow f(S)$
    end if
end for
return S^*

Construction and Improvement Heuristics

An approach for distinguishing between different types of heuristics is to classify them as either construction or improvement heuristics, which in the scheduling context, can be defined as:

- Greedy Construction heuristics: "start without a schedule and gradually construct one by adding one job at a time" (Pinedo, 2012)
- Improvement heuristics: "begin with a complete schedule, which may be selected arbitrarily, and then try to obtain a better schedule by manipulating the current schedule" (Pinedo, 2012)

Local Search

A local search method is a type of heuristic that iteratively searches for a better solution, in a local space. In other words, it relies on making small changes to "gradually improve towards a local optimum" (Beek, Roa, Dullaert, & Vigo, 2018). This approximation algorithm is simple, flexible and 'exhibits good practical performance in empirical studies" (Korupolu & Plaxton, 2000).

It is also important to note that local search heuristics rely on operators, which are defined by a set of characteristics that influence the solution. However, there is a trade-off: "complex operators can perform a more extensive search of the solution space and thus can reach higher quality solutions, but the number of possible changes and thus the effort required to find improving changes increases with their complexity" (Beek et al., 2018). Therefore, it is common to limit their scope to improve the execution time of the algorithm and, thus, reduce the implementation time.

2.1.2. Literature Review and Related Work

Given the complexity of scheduling problems, it is possible to find a wide range of different approaches to solve them in the literature, from exact optimization models to heuristic and metaheuristic algorithms, each with its own strengths and limitations.

Several optimization models have been applied to scheduling problems, with Mixed-Integer Linear Programming being one of the most commonly used. In those cases, it is common to observe that the decision variables represent the sequencing and assignment of tasks to machines, and the objective function aims to minimize some metric, such as maximum lateness. It has been proven that this is an approach that leads to an optimal solution, but its resolution is NP-hard, which complicates its practical implementation when working with real-world data.

In light of what was mentioned above, new mathematical formulations or relaxation methods have been developed, typically based on standard models that are modified to approximate the optimal solution with a lower computational cost. In particular, column generation methods are widely used, where equations are manipulated to form an equivalent problem that is easier to solve. However, these approaches usually imply making assumptions that lead the solution away from the optimum.

To address large-scale scheduling problems where exact models are not feasible, heuristics are frequently employed. This is a different approach to achieve a lower computational cost. In the literature, it is possible to find classic heuristics such as local search methods, as well as new algorithms or variations of existing ones. However, it is important to note that these approaches do not necessarily lead to the optimal solution.

Metaheuristic implementations aim to explore more of the solution space, trying to avoid local optima. Typical examples of these methods are Genetic Algorithm and GRASP, the latter being an iterative, two-phased algorithm that mixes two traditional heuristics. Although the metaheuristic approach has a lower computational cost than optimization models and yields better results than traditional heuristics, it also does not guarantee optimality.

In Table 2.1, a comparison of some key studies related to the RMC Dispatching problem are presented, including the various approaches discussed above.

Article	Brief description	Method	Data	Results	Comments
(Maghrebi et al., 2016)	A column generation algorithm for vehicle routing problems with time window constraints are applied to the RMC dispatching problem	Dantzig Wolf formulation (column generation)	9 instances selected randomly from an actual field data, delivering to 197 customers per day	Nearly 10 times faster than MIP with only $\sim$ 1% increase in distance	Proposes mathematical method; 2 steps (master problem and subrpoblem)
(Maghrebi, Periaraj, Waller, & Sammut, 2014a)	Comparison of a column generation approach and a robust genetic algorithm for solving RMC delivery problems	Robust Genetic Algorithm vs. Column Generation	6 instances randomly selected from real field data of an active RMC company in Adelaide, Australia; covering a two-month period with deliveries ranging from from 19 to 198 per day	Both approaches provided feasible solutions for around 99% of customers. CG solutions resulted in 20% lower cost, while the Robust GA was 40% faster	Comparative study between two methods.
(Maghrebi et al., 2014b)	Application of Benders decomposition to solve RMC dispatching problems	Benders Decomposition	A single-day instance with 22 customers	Near-optimal solution obtained in practical computational time	Two-step mathematical method (master problem and subproblem)

Continued on the next page

Article	Brief description	Method	Data	Results	Comments
(Marinakis et al., 2006)	A bilevel formulation for vehicle routing problems and a solution method based on a genetic algorithm	Genetic Algorithm + Traveling Salesman Problem	Two datasets: one with 14 instances and another with 20 large-scale vehicle routing problems	First set achieved an average gap of 0,479%; second set, 0,826%	Two-level method (GA for upper level, TSP for lower level)
(Balin, 2011)	Comparison of a robust genetic algorithm and the Longest Processing Time rule for parallel machine scheduling problem with fuzzy processing times	Robust Genetic Algorithm vs. Longest Processing Time Rule	10 randomly generated datasets with 9 jobs and 4 identical parallel machines	GA provided more efficient solutions but occasionally produced infeasible ones; worst GA solution still outperformed LPT	Proposes an enhanced GA variant and compares its performance with Longest Processing Time rule

Table 2.1: Comparison of key papers addressing the RMC Dispatching Problem

As previously mentioned and shown in Table 2.1, various mathematical formulations and metaheuristic methods are used to solve the scheduling problem in the context of the RMC Dispatching problem. While some studies report achieving the optimal solution, they also occasionally obtained infeasible solutions, where not all customer orders are successfully scheduled. Moreover, many implementations rely on simplifying assumptions such as:

- The release time of all tasks is set to 0.
- All machines are identical and capable of processing any task.
- Dispatch trains are modeled as a single task to be processed.

Due to these simplifications, the complexity of the problem is reduced, enabling implementation at larger scales. However, the likelihood of generating infeasible solutions or failing to reach optimality, is increased.

In some cases, more favorable results are reported, but they are typically based on significantly smaller datasets than those used in this thesis.

In summary, a wide range of different approaches have been proposed to address the scheduling problem in the RMC dispatching with acceptable computational cost, from mathematical models and formulations to heuristics and metaheuristic methods, and their variants. Nonetheless, most implementations rely on assumptions that do not capture all the real-world conditions and data scale, often resulting in feasible solutions that do not reach the optimum, or infeasible solutions.

2.1.3. Identified Research Gaps

From the literature review analysis presented in section 2.1.2, it is possible to identify different research gaps, which are detailed next.

In the first place, scalability issues can be noticed. Several optimization-based approaches rely on small or medium-sized datasets, which are also typically simplified by assumptions. Then, although some of them achieve high-quality or even optimal solutions, their performance on large-scale, real-world instances, such as those addressed in this thesis, remains largely unexplored.

Referring to heuristic and metaheuristic implementations, they occasionally produce infeasible results, for example, leaving customer orders unassigned in the proposed schedule. This problem may be corrected using other methods, but in the context of this thesis, that is an expensive alternative. Thus, the possibility of obtaining unfeasible solutions undermines their practical utility for industrial dispatching systems, where feasibility is a strict requirement.

Two typical approaches can be observed in the literature: optimization-based methods with the aim of achieving optimal solutions and approximation methods as an alternative for reducing execution time. In particular, column generation, Bender Decomposition, and Metaheuristic methods can also be classified into those two categories, providing improvements in either computational cost or solution quality. However, very few studies succeed in balancing both simultaneously at a sufficient level for operational deployment.

Finally, most works rely on either purely mathematical formulations or heuristic methods. Thus, the potential of hybrid methods that integrate optimization models with data-driven or learning-based techniques to improve scalability and robustness is still underexplored. A more detailed analysis of the hybrid approach will be presented in section 2.2.3.

2.2. Theoretical framework II: Machine Learning

2.2.1. Key Concepts

Machine Learning

According to (Rebala, Ravi, & Churiwala, 2019), "Machine learning (ML) is a field of computer science that studies algorithms and techniques for automating solutions to complex problems that are hard to program using conventional programing methods". In simple words, the idea is that the computer can learn from examples and then apply the knowledge to new data. Thus, Machine Learning models can describe or predict data, through supervised, unsupervised, or with more sophisticated learning approaches.

Supervised and Unsupervised Learning

The main difference between these two approaches lies in the data needed as input for the model: in supervised learning, giving the correct label is required, while the computer can learn without knowing the correct answer in unsupervised learning (Rebala et al., 2019).

Supervised Learning: Classification and Regression

Classification and regression, both supervised learning problems, learn and predict labeled data, obtaining different types of results: Classification uses labeled data into different categories, binary or multi-class, specified previously in the input of the model. On the other hand, the aim of regression problems is to predict the value of a quantitative variable (Rebala et al., 2019).

Classification performance metrics

To evaluate the performance of a model on classification problems, a series of metrics can be used, defined by (Zhou, 2021) as follows:

- **Accuracy:** proportion of correctly classified samples.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.3}$$

- **Precision:** proportion of predicted positive samples that are actually positive.

$$Precision = \frac{TP}{TP + FP} \tag{2.4}$$

- **Recall:** proportion of actual positive samples that were correctly predicted.

$$Recall = \frac{TP}{TP + FN} \tag{2.5}$$

- **F1-score:** harmonic mean of precision and recall; balances both metrics in a single value.

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{2.6}$$

where True Positive (TP) and True Negative (TN) correspond to the values correctly classified, respectively, while False Positive (FP) and False Negative (FN) are classified by the model as the incorrect class, as shown in the confusion matrix presented next (Table 2.2), adapted from (Zhou, 2021).

Table 2.2: Confusion matrix of binary classification

True class	Predicted class	
	Positive	Negative
Positive	TP	FN
Negative	FP	TN

Regression performance metrics

Regarding performance metrics for regression problems, (Kuhn & Johnson, 2013) define the following:

- **Root Mean Squared Error (RMSE):** measures the average magnitude of the prediction error. It is expressed in the same units as the original data and is typically

interpreted as the average distance between the observed values and the model's predictions.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} = \sqrt{mean\{(y_i - \hat{y}_i)^2\}} \tag{2.7}$$

- **Coefficient of determination (R2):** represents the proportion of variance in the observed data that is explained by the model.

Likewise, two other commonly used aggregate metrics for forecasting are included in this research, as defined by (Kotu & Deshpande, 2019):

- **Mean Absolute Error (MAE):** measures the average magnitude of the absolute errors between predicted and actual values, without considering their direction.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) = mean\{|y_i - \hat{y}_i|\} \tag{2.8}$$

- **Mean Absolute Percentage Error (MAPE):** expresses the prediction accuracy as a percentage, by averaging the absolute percentage errors between actual and predicted values.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{100 \cdot |y_i - \hat{y}_i|}{y_i}\right] = mean\left\{\frac{100 \cdot |y_i - \hat{y}_i|}{y_i}\right\} \tag{2.9}$$

Ensemble models

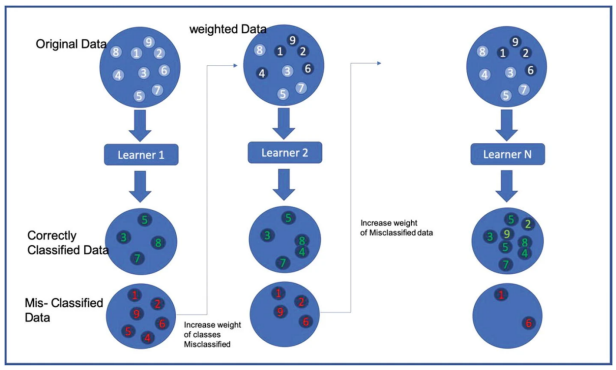
According to (Kumar & Jain, 2020), "ensemble learning is a combination of multiple machine learning techniques performed together". These combinations are sets of different or equal algorithms, trained as individual learners first, which are then, combined to obtain the final output (Zhou, 2021). Examples of this type of methods are Random Forest, AdaBoost and XGBoost algorithms.

Random Forest

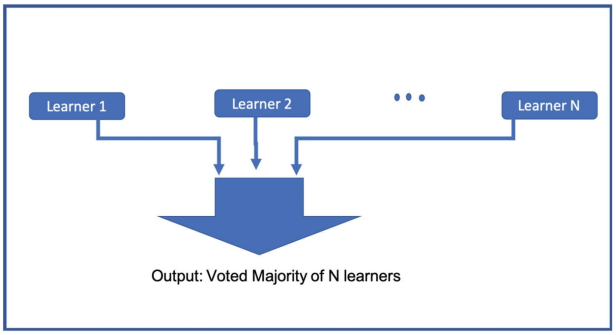
Random Forests (RF) belong to ensemble learning, it is composed by multiple Decision Tree models and can be used as a classification or regression algorithm (Rebala et al., 2019). In particular, "RF selects from a subset of k features randomly generated from the feature set of the node. The parameter k controls the randomness, where the splitting is the same as in traditional decision trees if $k = d$, and a split feature is randomly selected if $k = 1$" (Zhou, 2021). The final prediction is obtained through an "aggregation of results from all the Decision Trees" (Rebala et al., 2019).

AdaBoost

It combines weak learners, typically Decision Tree models, to obtain predictions. In each iteration, it applies a penalization to the wrong values to correct the prediction in the next one to finally, obtain a result with good performance. An example of this algorithm used as classifier is shown in Figure 2.1 (Kumar & Jain, 2020).



(a) Performing each weak learner



(b) Combining weak learners outputs to obtain final boosted result

Figure 2.1: Diagram showing how Adaboost classifier works

XGBoost

According to (Kumar & Jain, 2020), this algorithm improves other similar methods. One of its characteristics is that "it dynamically determines the depth of decision trees used as weak learners, with penalization parameters added for preventing trees with high depth. This prevents overfitting and improves performance".

Overfitting

According to (Kotu & Deshpande, 2019), overfitting is a common issue in supervised Machine Learning models. It refers to "the process of building a model specific to the training data that achieves close to full accuracy on the training data. However, when this model is applied to new data or if the training data changes somewhat, then there is a significant degradation in its performance". This problem also extends to regression models.

2.2.2. Literature Review and Related Work

Different approaches and applications in the literature merge the fields of Machine Learning and Optimization. In particular, many ML algorithms incorporate optimization techniques to compute model parameters or improve prediction accuracy. One of the most common cases involves applying variations of optimization methods to enhance the performance of ML models. For example, some authors use optimization models to modify the objective functions of Random Forest predictor optimizers or even, to find an optimally representative example of a Random Forest model.

Conversely, another typical approach involves using ML techniques within Optimization methods. One of the most common examples consists in using ML tools for demand forecasting, where the generated predictions serve as input data for an optimization model. But what if the application of Machine Learning techniques within Optimization not only supports but also improves the optimization process itself?

Several studies support this idea. For instance, authors have integrated Reinforcement Learning into heuristics to improve the performance of optimization algorithms. Similarly, other works have also applied ML techniques to reduce the complexity of optimization solvers, thereby decreasing their execution time.

These examples share the goal of improving the performance of optimization models, by reducing execution time while maintaining solution quality. However, achieving both goals simultaneously is not always possible. In fact, there are cases where the runtime is decreased with the new approach, but at the cost of higher error margins. Yet, there are other cases in which it is feasible. For example, using modified Neural Networks to approximate the solution set of an optimization algorithm.

This research follows the latter approach, investigating the feasibility of replacing an optimization model with a ML method that produces high quality solutions within a short execution time.

In Table 2.3, a comparison of key studies related to methods that combine the fields of Machine Learning and Optimization is presented, covering the different approaches described above.

2.2.3. Identified Research Gaps

From the reviewed literature in section 2.2.2, it is possible to identify several gaps that motivate the present research. First, although many studies have explored the integration of Machine Learning (ML) and Optimization, most of them focus on using Optimization techniques to enhance ML performance or on leveraging ML as a supportive tool within Optimization pipelines. In contrast, relatively few works investigate the possibility of replacing an Optimization model with a ML method that approximates its solutions with high accuracy and significantly lower computational cost.

Similar to the case of Optimization research, ML studies often report a trade-off between solution quality and execution time. That is, while some ML-based approaches are able to reduce computational effort, they do so at the expense of higher error margins; conversely, methods that preserve solution accuracy typically exhibit runtimes that remain too high for real-time or near real-time applications. Thus, the simultaneous achievement of low runtime and high-quality solutions is still an open challenge.

ML studies also present scalability issues and simplifying assumptions, as most of the optimization-based approaches. In particular, most existing ML works are applied to theoretical problems or, on the other hand, problems with assumptions that do not capture the complexity of the real world. In particular, this limits their generalization to real-world industrial contexts. The concrete dispatching problem studied in this thesis provides a relevant and practical case where predictive models must balance efficiency and accuracy to be useful in operational environments.

These gaps highlight the need for research that not only demonstrates the feasibility of approximating Optimization outcomes using ML, but also validates such methods in realistic industrial settings, ensuring both computational efficiency and solution quality.

Article	(Lombardi, Milano, & Bartolini, 2016)	(Xu, Shen, & Liu, 2025)	(Teodoro, Monaci, & Palagi, 2024)	(Forel, Parmentier, & Vidal, 2023)	(López-Rojas & Cruz-Villar, 2024)	(Spieckermann, Minner, & Schiffer, 2025)
Brief description	Proposes a methodology for integrating ML with Optimization techniques for thermal-aware workload dispatching.	Introduces a reinforcement learning-based hyper-heuristic framework to enhance the performance of Column Generation methods.	Evaluates the viability and efficiency of using MILP formulation for finding an optimal representative tree from a Random Forest model.	Proposes an explanation method that employs Integer-Programming methods to obtain relative and absolute explanations for Random Forest and Nearest-Neighbor predictors.	Presents a sufficient condition under which a Neural Network can approximate the solution set of an optimization algorithm.	Uses ML techniques to reduce problem complexity for a subsequent Combinatorial Optimization solver by predicting a relevant subset of variables.
Problem context	Thermal-aware workload dispatching problems.	Vehicle Routing Problem with Time Windows (VRPTW) and Bus Driver Scheduling Problem (BDSP).	Ten different applications of binary classification.	Simulated inventory and Uber routing, both integer programming problems.	Sequential Quadratic Programming (numerical experiment) and Model Predictive Control problem.	Fixed-Charge Transportation Problem (FCTP).
ML techniques	Artificial Neural Networks and Decision Trees.	Reinforcement Learning.	Random Forest.	Random Forest and Nearest Neighbors.	Neural Networks.	Graph Neural Network (GNN).
Optimization techniques	Local Search, Constraint Programming, Mixed Integer Non-Linear Programming, and SAT Modulo Theories.	Column Generation.	Mixed-Integer Linear Programming (MILP).	Integer Programming.	KKT conditions-based non-linear programming.	Combinatorial Optimization and two established FCTP meta-heuristics.

Continued on the next page

Article	(Lombardi et al., 2016)	(Xu et al., 2025)	(Teodoro et al., 2024)	(Forel et al., 2023)	(López-Rojas & Cruz-Villar, 2024)	(Spieckermann et al., 2025)
Data	Data extracted from a predictive model or harvested from a real system. 20 instances, each with 288 jobs to be mapped on 48 cores (6 jobs per core).	Solomon benchmark data (six problem types of 8–12 instances with 100 customers) for VRPTW and a randomly generated data for BDSP.	Benchmark 10 datasets from the UCI ML repository related to binary classification tasks.	Synthetic (dimensions from 5 to 500) and real-world data (Uber movement data, modeled using 45 nodes and 93 edges).	One synthetic and other constructed by iterations of an experiment.	Benchmark data sets of FCTP, including variants with edge capacities, fixed-step costs, and blending constraints.
Results	High-quality predicted solutions, but accompanied by a large margin of error.	RLHH outperforms traditional heuristics in runtime and solution quality.	Model effectively achieved a shallow interpretable tree approximating the tree ensemble decision function.	Interpretable explanations with provably minimal explanation distance.	The precision of the Neural Network approximation with its faster implementation, shows that it can replace numerical algorithms to solve optimization problems of the presented family.	High-quality solutions across all datasets, with significantly lower runtime compared to state-of-the-art mixed-integer linear programming.

Table 2.3: Comparison of key papers addressing the integration between ML and Optimization fields

2.3. Methodological Framework

2.3.1. Methodology

This thesis proposes a structured methodology to address the RMC Dispatching problem, using a combination of optimization, heuristic, and machine learning techniques. A dataset is used to implement an optimization model and a series of heuristics, designed to generate dispatch schedules for customer orders. These schedules are evaluated using a performance metric.

The optimization model is used as a lower-bound reference, while a random assignment heuristic as an upper-bound reference to compare the performance of the different construction heuristics. Once the best among them is chosen, improvement heuristics are implemented to further refine the solution, aiming to approach the optimum given by the optimization model.

The best-performing construction and improvement heuristics are then integrated into a metaheuristic approach. In particular, a GRASP-like metaheuristic was designed, given its effective and efficient performance in this type of problem (section 2.1.2). Once the results of this metaheuristic are sufficiently close to the optimum provided by the optimization model, they can be used as input for the machine learning model.

In the Machine Learning section, a hybrid approach is adopted, combining both classification and regression models. Specifically, the selected models were Random Forest, AdaBoost, and XGBoost, due to their efficiency and interpretability. Each of these models was implemented as both a classifier and a regressor. Linear approaches like Linear Regressions or Support Vector Machine models, were discarded due to poor performance in a previous analysis (Annex A).

All Machine Learning models are trained using the training portion of the dataset. The hybrid procedure is then applied to the test and validation data as follows: first, the classification models predict categorical labels. Then, based on these predicted classes, the dataset is divided accordingly, and the regression models are applied to each category to predict total lateness as an output of the complete method.

An overview of the entire methodology is presented in Figure 2.2.

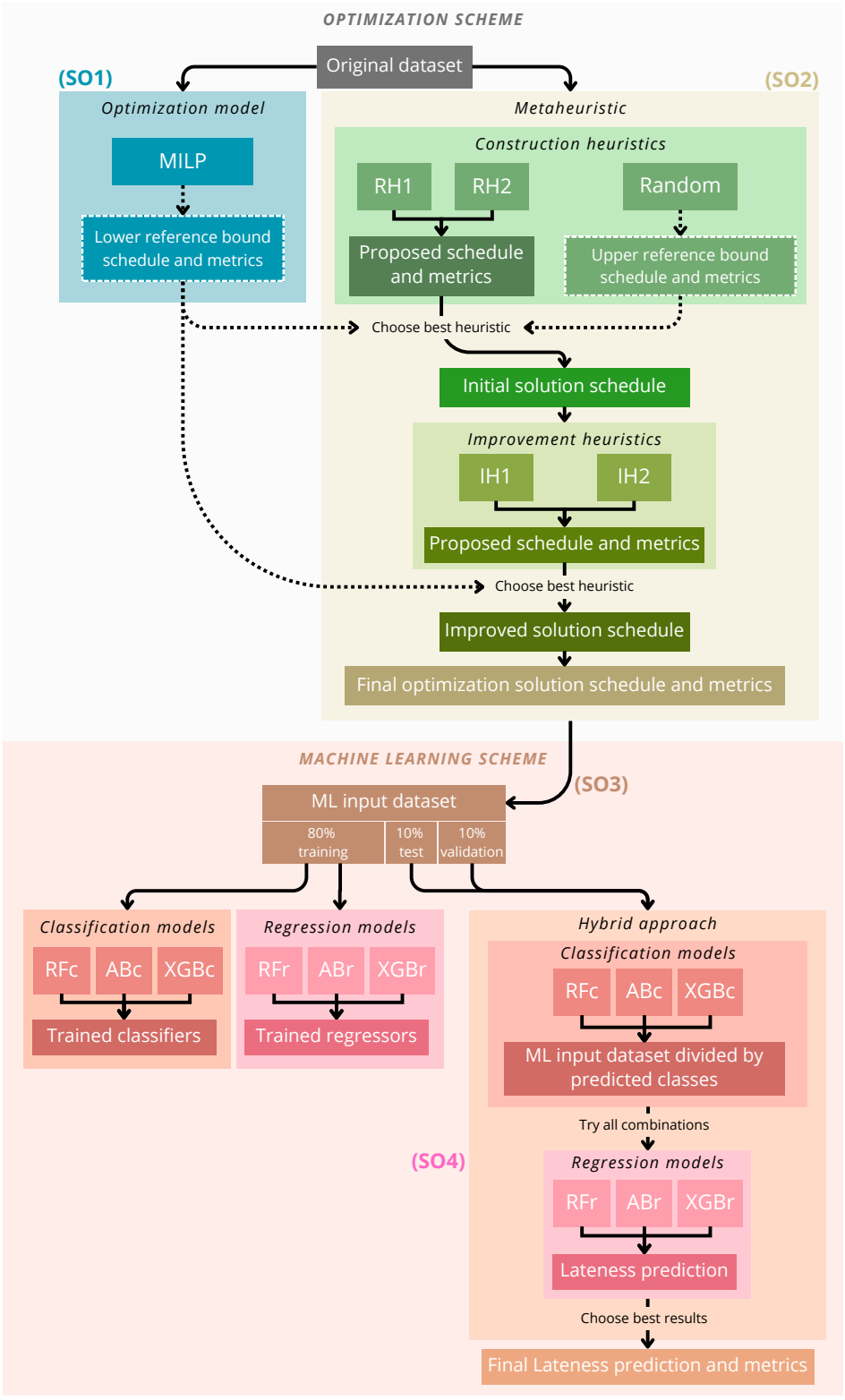


Figure 2.2: Methodology of the thesis

It is worth to mention that each step of the methodology presented previously, is linked with an specific objective. To represent this, the acronym of each objective is included in Figure 2.2. Additionally, the relation between them, along with the expected results, is presented in Table 2.4.

Specific Objectives		Methodology	
Acronym	Description	Scheme	Expected results
SO1	Design, develop, and evaluate a Mixed-Integer Linear Programming model tailored to the specific characteristics of the problem	Optimization	Optimization model
SO2	Design, develop, and evaluate a metaheuristic method, comprising construction heuristics for generating initial solutions and improvement heuristics for approaching the optimum	Optimization	Random, construction, and improvement heuristics, metaheuristic
SO3	Design and build a comprehensive dataset for training Machine Learning models, by generating and collecting solutions from the metaheuristic under different instances with and without perturbations	Machine Learning	Input for Machine Learning model
SO4	Design, develop, and evaluate a Machine Learning model that, by leveraging and integrating results obtained from heuristic approaches, predicts the optimal objective function value of the scheduling model, achieving acceptable accuracy with reduced execution time	Machine Learning	Machine Learning model

Table 2.4: Specific objectives linked with their expected results, including the section in which it will be developed

2.3.2. Techniques

Datasets

In this thesis, two datasets will be utilized. The first dataset is used to implement the optimization model and heuristics. It contains information about tasks and the concrete plant, and will be described in further detail in Chapter 3. The second dataset will be created from the results of the metaheuristic and is employed to train the machine learning models, with more details provided in Chapter 4.

Optimization model

As outlined in (Talbi, 2009), and illustrated in Figure 2.3, a classical decision-making process and the main steps in optimization modeling consist of:

- **Formulate the problem:** Identify the decision-making problem and define its objective. This formulation may be imprecise.
- **Model the problem:** Represent the problem using mathematical expressions. The aim is to build an abstract mathematical model capable of representing and solving the problem.
- **Optimize the problem:** Solve the model through some algorithm to obtain a solution that can be either optimal or suboptimal.
- **Implement a solution:** Test the solution obtained in the previous step in a real or simulated scenario. If the solution meets the necessary criteria is considered "acceptable". Otherwise, the solution is "unacceptable" and the optimization model or algorithm needs to be improved.

In practice, it is common to iterate through these steps to refine the model and reach a better solution.

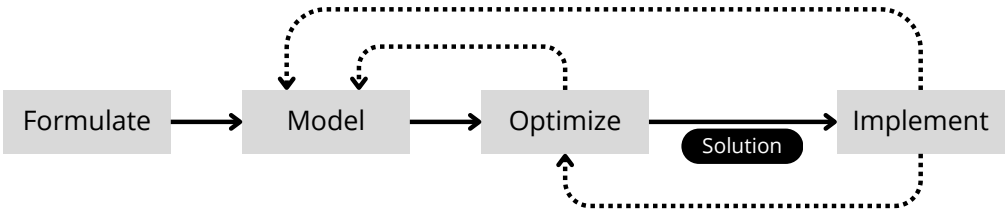


Figure 2.3: Classical decision-making process (Talbi, 2009, p. 2)

In particular, a MILP optimization model is used in this thesis.

Heuristics

Two types of heuristics are considered in this work: construction heuristics and improvement heuristics. The construction heuristics aim to generate an initial feasible solution and are based on a greedy algorithmic approach. The improvement heuristics are then applied to refine these initial solutions in an attempt to approach the optimal result provided by the MILP model.

According to (Talbi, 2009), when formulating an optimization problem, it is necessary to assess whether the problem can be solved efficiently and how it can be approached. In many real-world cases, solving such problems is not feasible in polynomial time. In such situations, metaheuristics offer a promising alternative.

Once a metaheuristic approach is decided, several steps are recommended. In this thesis, all classes of heuristics have the same general guidelines, which are illustrated in Figure 2.4 and described below.

- **Design:** It is important to design a representative metaheuristic, which implies defining how solutions are evaluated and the operation form of search operators. Additionally, a well-defined objective function and appropriate handling of constraints, tailored to the context and complexity of the problem, are essential.
- **Implementation:** This step consists in selecting or developing the algorithm, which may be an existing one, a new formulation, or a hybrid of both. Furthermore, it also includes the selection of appropriate software to implement the algorithm, considering the complexity of the problem.
- **Parameter Tuning:** Gives flexibility and robustness to the algorithm, influencing the efficiency and effectiveness of the search process. Thus, it is crucial to use optimal values for them.
- **Performance Evaluation:** the book mentions three sub-steps to apply for performance evaluation. First, an experimental design should be defined, that considers goals, instances and relevant factors. Second, solution quality and other metrics, such as computational effort and robustness, must be measured. Finally, results should be clearly reported, through plots and interactive visualizations.

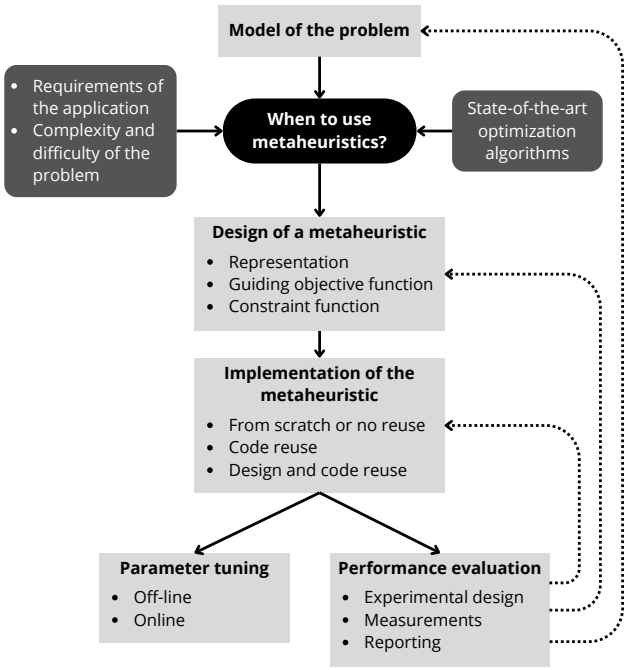


Figure 2.4: Guidelines for solving a given optimization problem using metaheuristics (Adapted from Talbi, 2009, p. 77)

Machine Learning model

Various authors agree that there is no unique process framework to apply Machine Learning (ML) models in the Data Science (DS) world. However, the Cross Industry Standard Process for Data Mining (CRISP-DM) is commonly used and has been adapted for broader Data Science applications (Kotu & Deshpande, 2019). Thus, it will be adopted in this research.

As illustrated in Figure 2.5, the framework consists of five steps:

1. Prior knowledge of the problem context
2. Data preparation
3. ML modeling
4. ML implementation
5. Resulting knowledge obtained after completing the process

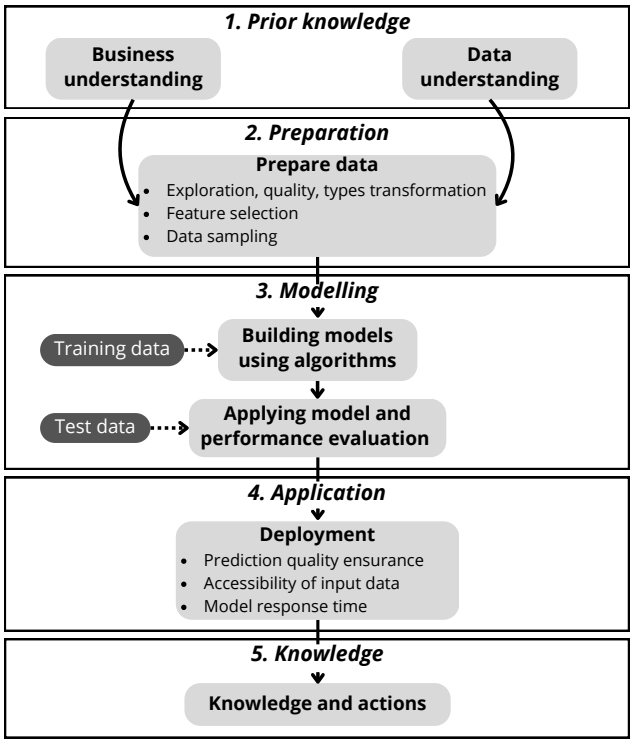


Figure 2.5: DS process flow (Adapted from Kotu & Deshpande, 2019)

This framework aligns with the typical steps involved in building Machine Learning models, as explained in (Kuhn & Johnson, 2013):

1. Pre-processing the predictor data
2. Estimating model parameters
3. Selecting predictors for the model
4. Evaluating model performance
5. Fine tuning class prediction rules

Data splitting

Splitting the dataset into training, test, and validation subsets allows to define and fit models with a portion of the data while evaluating its performance with new and unseen samples. This helps to detect problems such as overfitting (Rebala et al., 2019). In other words, "the 'training' dataset is the general term for the samples used to create the model, while the 'test' or 'validation' dataset is used to qualify performance" (Kuhn & Johnson, 2013). Additionally, the same author mentions that the split can be performed using different strategies, including simple random sampling and stratified sampling, the latter of which preserves the distribution of categorical classes.

Cross-Validation

It is a widely used method in the Machine Learning world to, for example, prevent overfitting. In fact, "resampling methods, such as cross-validation, can be used to produce appropriate estimates of model performance using the training set" (Kuhn & Johnson, 2013). The most common approach is k-fold cross-validation, where the data is divided into k equal parts named *folds*, where $k-1$ are used for training and the remaining one for testing, as shown in Figure 2.6 (Zhou, 2021).

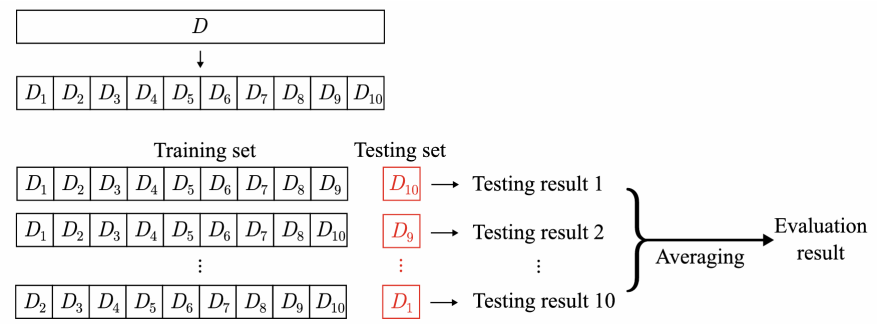


Figure 2.6: Example diagram of 10-fold cross-validation (Zhou, 2021)

Parameter tuning

Beyond choosing the correct Machine Learning model, it is also necessary to set values for its hyper-parameters to achieve a good performance (Zhou, 2021), in other words, values for the "parameters which cannot be directly estimated from the data" (Kuhn & Johnson, 2013). For that purpose, there are several libraries to perform a search among a set of potential values for the ones that produce the best results. This thesis applies grid search with cross-validation to perform parameter tuning, a method that "builds a model for each possible combination of all of hyperparameter values in the grid, evaluates each model, and selects the hyperparameters that yields the best results" (Ghatak, 2019).

Feature selection

Feature selection refers to identifying an adequate subset of input variables to use, which is crucial because not all of them are relevant for the model to learn (Zhou, 2021). As noted in (Kotu & Deshpande, 2019), "it optimizes the performance of the data science algorithm and it makes it easier for the analyst to interpret the outcome by reducing the number of attributes or features that one must contend with".

In this work, Recursive Feature Elimination with Cross-Validation (RFECV) is used. This method evaluates model performance across cross-validation folds for different subsets of features, identifying the optimal number of features. Indeed, "The number of features selected is tuned automatically by fitting an RFE selector on the different cross-validation splits (provided by the cv parameter). The performance of each RFE selector is evaluated using scoring for different numbers of selected features and aggregated together. Finally, the scores are averaged across folds and the number of features selected is set to the number of features that maximize the cross-validation score" (scikit learn, 2025).

Model selection

In (Kuhn & Johnson, 2013), the following considerations are recommended when choosing a Machine Learning model:

- Start with several models that are the least interpretable and most flexible
- Investigate simpler models that are less opaque
- Consider using the simplest model that reasonably approximates the performance of the more complex methods

Chapter 3

Optimization solution

The problem under study involves a set of concrete discharge hoppers, which are treated as machines that process customer orders. Each order can be seen as a single dispatch train, consisting of a group of tasks. These tasks, are characterized by sharing the same delivery address, a feasible subset of machines on which they can be processed, and a required spacing time between consecutive deliveries to client, which must be respected.

The aim of this chapter is to describe the scheme designed to assign, order, and determine the start times of all tasks in order to generate an optimal schedule that maximizes service levels by minimizing the total lateness of customer orders.

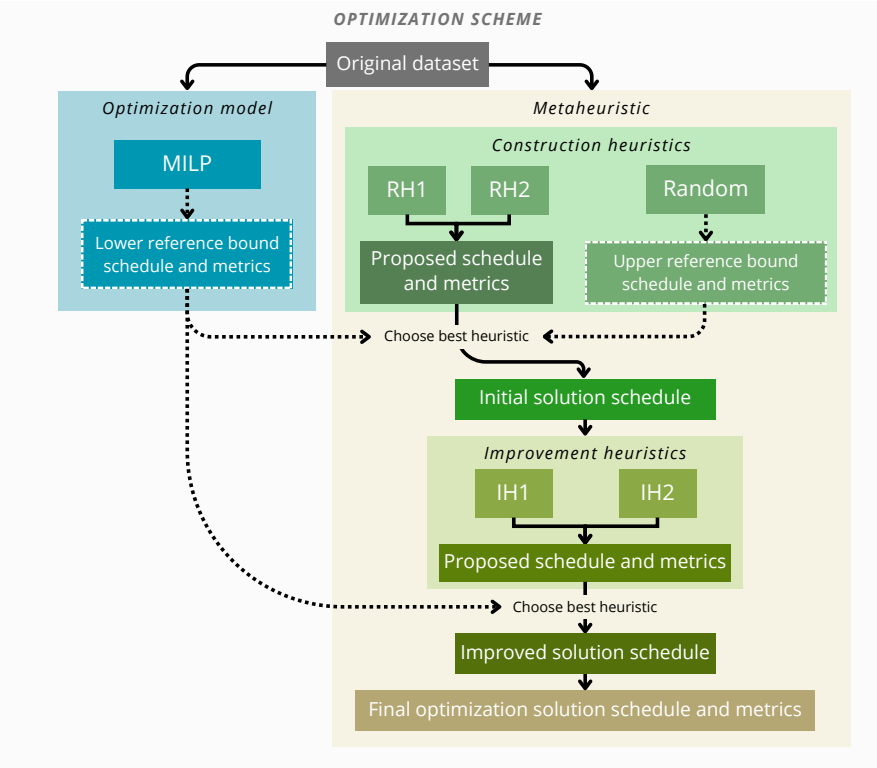


Figure 3.1: Diagram of the optimization solution.

3.1. Dataset

In this section, the data used for the optimization model and construction heuristics is described in detail.

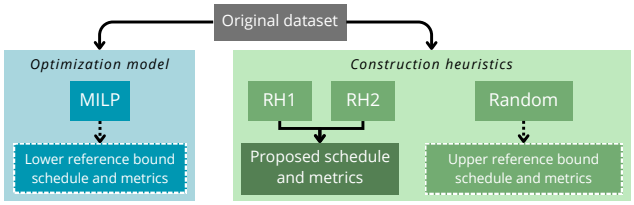


Figure 3.2: Uses of the initial data set

The original dataset is synthetic and simulates the operations of a concrete plant. It contains 3,366 rows, each representing a task to be processed in the plant and dispatched to a customer within a given week. It is important to note that these rows do not necessarily correspond to orders placed during that week, but rather to all dispatches scheduled to occur in that period. Due to the existence of dispatch trains, a single customer order may be divided into multiple dispatches, each recorded as a separate task in the dataset. For example, the order "*Monday_order_1*" may be split into the dispatches "*Monday_order_1_1*" and "*Monday_order_1_2*", which would appear as two distinct rows.

The dataset contains 40 columns with information regarding the orders and the concrete plant. A complete list of features is described in Table B.1 in Annex B, while the most relevant ones are summarized below, in Table 3.1.

Variable	Description
`order_id`	Order identification number
`dispatch_id`	Dispatch identification number
`due_date_day`	Day of the week when the dispatch is required to arrive at the delivery address of the client
`due_date_time`	Time of the day when the dispatch is required to arrive at the delivery address of the client
`spacing`	time between dispatches received by the customers
`load_time_p1` - `load_time_p10`	load time of task in machine p
`travel_time_p1` - `travel_time_p10`	travel time from machine p in the concrete plant to the customer, using a truck
`unload_time`	unloading time
`return_time_p1` - `return_time_p10`	return time of the truck from the customer to machine p of the concrete plant

Table 3.1: Dataset columns used for the optimization model and heuristics

It is important to mention that the 10 feeding hoppers in the concrete plant are modeled as machines p, where not every machine can process every task. Thus, for each task j, a subset of machines P_j is defined, containing those capable of processing it. As a result, the variables representing travel and return times contain null values for infeasible machine-task combinations.

3.1.1. Data Processing

Treatment of Missing Values

As mentioned earlier, some variables contain null values. However, no imputation or modification will be performed. The reason for this is that, for the model and heuristics, a subset of feasible machines will be defined, ensuring that tasks are never assigned to a machine corresponding to a column with null values.

Transformation of Time Variables

To facilitate data manipulation, all temporal variables and parameters will be defined as continuous values, representing the number of minutes elapsed since the beginning of the week. The transformation is defined as follows:

- Monday at 00:00 hours corresponds to minute 0.
- Times on Monday are represented by the number of minutes elapsed since minute 0 of that day.
- Times on subsequent days are represented by the number of minutes elapsed since minute 0 of the week.

Figure 3.3 illustrates an example of this transformation. For each record, the time difference from 00:00 of the same day is first calculated and expressed in minutes. Then, the number of full days elapsed since Monday 00:00 is converted to minutes and added to the previous value.

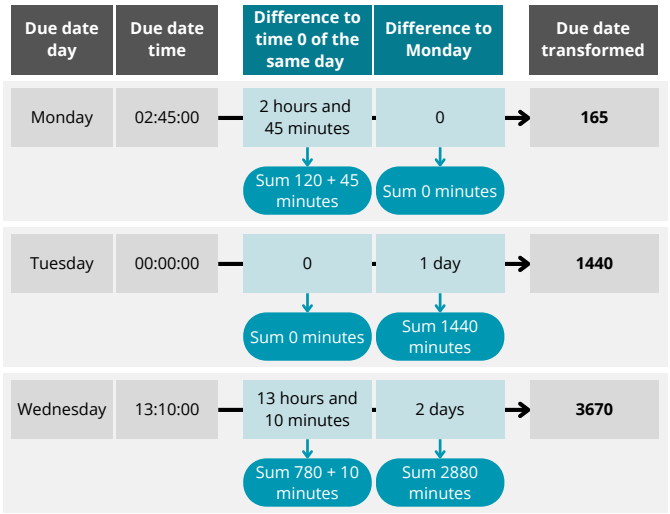


Figure 3.3: Example of time variables transformation to continuous values

3.1.2. Instances

To execute the optimization model and the different heuristics, a set of instances is defined. As a starting point, the dataset is divided by weekday: each of the resulting five subsets containing the orders with a due date set on that specific weekday, including all tasks belonging to those dispatch trains.

In this dataset, each column corresponds to a specific parameter and each row represents an individual task, whereas an order is defined as the complete set of tasks belonging to a single dispatch train. Based on this structure, the principal characteristics of the initial instances are shown in Table 3.2 below.

Instance	Size	Number of orders	Description
Monday	666 rows × 41 columns	178	Tasks to be dispatched on Monday
Tuesday	711 rows × 41 columns	182	Tasks to be dispatched on Tuesday
Wednesday	711 rows × 41 columns	189	Tasks to be dispatched on Wednesday
Thursday	654 rows × 41 columns	172	Tasks to be dispatched on Thursday
Friday	624 rows × 41 columns	170	Tasks to be dispatched on Friday

Table 3.2: Initial instances

However, in the implementation phase, due to computer capacity limitations, the instances defined in Table 3.2 could not be executed. Thus, new subsets were created.

The first division criterion was the same as the one previously mentioned, ending with five instances: one for each working day of the week, identical to the five instances described in table 3.2. Subsequently, 4 subsets were created for each day's data using different criteria, as presented in Table 3.3.

Criteria	Description	Values
Weekday	Due date day of tasks	{Monday,Tuesday,Wednesday,Thursday,Friday}
Size	Approximate number of tasks comprising the instance	{50,60,70,75}
Subset range	Time window from which the subset was selected	{First hours, Midday, Afternoon, Random}

Table 3.3: New instances creation criteria

Combining all possible values of the three criteria in Table 3.3 results in a total of 80 distinct instances.

3.2. Formulations

3.2.1. Optimization Model

This section details the optimization model used as the lower reference bound. The model takes the initial dataset as input and produces a schedule proposal as output (Figure 3.4).

The model is formulated as an Unrelated Parallel Machines Scheduling Problem (UPMSP). For simplicity, time is treated as a continuous variable, which allows to schedule tasks at specific moments within the one week time horizon. As described in Section 3.1.1, all temporal parameters are expressed in minutes, with minute 0 corresponding to 00:00 on Monday.

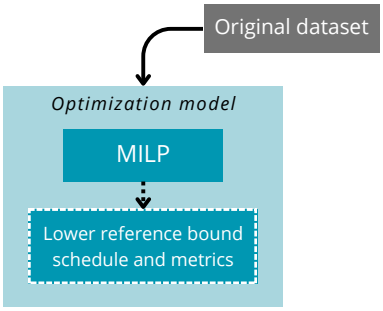


Figure 3.4: Flow diagram of the optimization model

The formulation accounts for the existence of dispatch trains and, for each task, defines a subset of feasible machines capable of processing it. Machine availability constraints are also incorporated.

Sets and Parameters

A description of the sets and parameters used in the model are described in Table 3.4.

Sets	
Symbol	Description
J	Tasks to be processed
M	Machines
P_j	Feasible machines to process the task j
T	Dispatch trains
$Pred_j$	Predecessors of task j in its dispatch train
Di	Dummy tasks; fictitious initial nodes to form the sequences
Df	Dummy tasks; fictitious final nodes to form the sequences
Auxiliary sets	
$JUDf$	$\{J \cup Df\}$
$DiUJ$	$\{Di \cup J\}$
$DiUJUDf$	$\{Di \cup J \cup Df\}$
$Xidxs$	$\{(i,j,p) \mid j \in JUDf, i \in DiUJ, p \in P_j \cap P_i, i \neq j\}$
$Yidxs$	$\{(j,p) \mid j \in DiUJUDf, p \in P_j\}$
Parameters	
dd_j	Due date of the task j.
r_{jp}	Release time of the task j, if processed by machine p_j.
$tprec_j$	Spacing time between the tasks of a dispatch train. In other words, the minimum amount of time that must elapse between consecutive deliveries for the customer.
$tcarga_{pj}$	Loading time of task j onto the truck, after been processed by machine p.

Continued on the next page

Symbol	Description
$tproc_{jp}$	Processing time of task j in machine p, defined as the sum of travel time to the client and loading time in the plant for each task-machine combination.
w_j	Weight assigned to task j.
$smax$	Maximum amount of time a task can wait from its release until it begins to be processed.
h	Slack
M	Big M

Table 3.4: Sets and parameters used in the optimization model

Decision Variables

A brief description of the decision variables defined for the model is shown in Table 3.5.

Symbol	Type	Description	Sets
X_{ijp}	Binary	1 if task j is assigned right after task i, to machine p_j	$(i,j,p) \in Xidxs$
Y_{jp}	Binary	1 if task j is assigned to machine p_j	$(j,p) \in Yidxs$
T_j	Continuous	Initial time when the task j starts to be processed, in minutes	$j \in J$
L_j	Continuous	Lateness of task j, in minutes	$j \in J$

Table 3.5: Decision variables of the optimization model

Model Formulation

$$\min \sum_{j \in J} w_j L_j \tag{3.1}$$

s.t.

$$\sum_{p \in P_j} Y_{jp} = 1, \qquad \forall j \in DiUJUDf \tag{3.2}$$

$$T_j \geq r_{jp} \cdot Y_{jp}, \qquad \forall j \in J, \forall p \in M_j \tag{3.3}$$

$$L_j \geq T_j + \sum_{p \in P_j} tproc_{jp} \cdot Y_{jp} - dd_j, \qquad \forall j \in J \tag{3.4}$$

$$T_j \geq T_{j-1} + tprec_j, \qquad \forall j \in Pred_j \tag{3.5}$$

$$\sum_{j \in JUDf} \sum_{p \in P_i \cap P_j, i \neq j} X_{ijp} = 1, \qquad \forall i \in DiUJ \tag{3.6}$$

$$\sum_{i \in DiUJ} \sum_{p \in P_i \cap P_j, i \neq j} X_{ijp} = 1, \qquad \forall j \in JUDf \tag{3.7}$$

$$X_{ijp} \leq \frac{Y_{jp} + Y_{ip}}{2} \qquad \forall (i,j,p) \in Xidxs \tag{3.8}$$

$$T_j - T_i \geq tcarga_{ip} \cdot X_{ijp} + M \cdot (X_{ijp} - 1) \qquad \forall i,j \in J, i \neq j, \forall p \in \{P_j \cup P_i\} \tag{3.9}$$

$$T_j \leq T_{j-1} + tprec_j + smax \qquad \forall j \in Pred_j \tag{3.10}$$

$$X_{ijp} \in \{0,1\}, \qquad \forall i,j \in J, \forall p \in \{P_j \cap P_i\} \tag{3.11}$$

$$Y_{jp} \in \{0,1\}, \qquad \forall j \in J, \forall p \in P_j \tag{3.12}$$

$$T_j, L_j \geq 0, \qquad \forall j \in J \tag{3.13}$$

The objective function of the model, presented in Equation 3.1, describes the weighted total lateness of the schedule proposal, defined as the sum of Lateness L_j weighted by w_j, of all tasks j. Consequently, Equation 3.4 defines Lateness, through the difference in time between the moment when the task is received by the customer and its due date.

As for the other constraints, Equation 3.2 imposes all tasks must be assigned to a single machine; all the tasks will be processed by definition. Furthermore, Equation 3.3 requires that the start time of each task must be later than its release time, while Equation 3.10 enforces that each task must not wait more than $smax$ minutes between being released until the moment it starts to be processed.

With respect to Equation 3.5, it enforces the compliance of the required spacing time between consecutive tasks of the same dispatch train. Constraint 3.9, models machine availability, ensuring that a task is not processed on a feasible machine when it is busy.

Equations 3.6 and 3.7 define the task sequences, requiring that each task has exactly one immediate successor and predecessor, respectively. Likewise, Equation 3.8 establishes that tasks may be successors or predecessors only if they have been assigned to the same machine. Finally, Equations 3.11, 3.12, and 3.13 define the nature of the variables.

3.2.2. Construction Heuristics

This section describes the construction heuristics, which take the initial dataset as input and generate a proposed schedule as output. Additionally, a random assignment method is included and used as an upper reference bound.

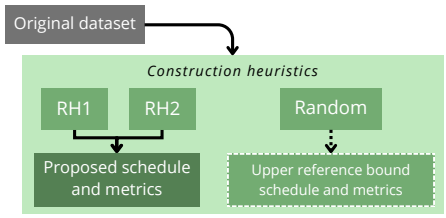


Figure 3.5: Flow diagram of the construction heuristics

The construction heuristics proposed in this section follow a greedy approach, in which tasks are ordered and assigned to feasible machines according to predefined criteria. Specifically, two heuristics with different ordering rules (RH1 and RH2) were created, along with a random one (R), designed to serve as a referential upper bound, as shown in Table 3.6. All of them have the same structure essentially: Each task is first prioritized according to the ordering criteria and subsequently assigned to a feasible machine following the assignment criteria.

	RH1	RH2	R
Ordering criteria	Due Date	Release time	Random
Assignment criteria	Random, weighted by processing time	Random, weighted by processing time	Processing time

Table 3.6: Ordering and assignment criteria of construction heuristics

It is worth noting that Release time, Due date and Processing time correspond to the parameters r_{mj}, dd_j and $tproc_{jp}$, respectively, from the optimization model described in Section 3.2.1. Additionally, all other parameters presented in that table are used in the heuristics, except for those specific to the optimization model, namely w_j, $smax$ and M. Furthermore, a probability parameter p_G was introduced.

As for the sets, the construction heuristics were defined using the same first seven sets of the optimization model, presented in Table 3.4 of Section 3.2.1. That is, all the sets presented in the table, without considering the auxiliary ones.

All construction heuristics account for the existence of dispatch trains, but their handling differs. While RH1 and RH2, allow assigning tasks from a specific train to different machines, the random R imposes that all the tasks belonging to the same train must be assigned to the same feasible machine, scheduled immediately one after the other. Consequently, certain orderings and assignments possible in RH1 and RH2 are not feasible under R. An example illustrating these differences is presented in Figure 3.6.

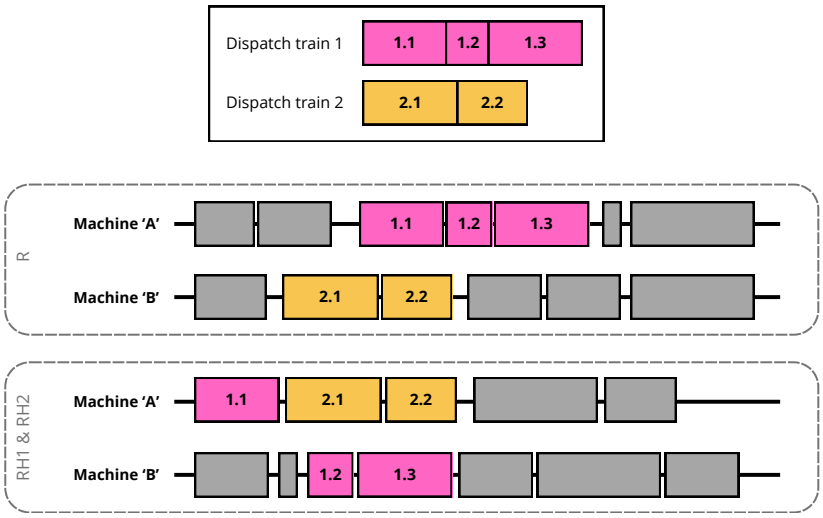


Figure 3.6: Example of different approaches to schedule dispatch trains between the construction heuristics RH1, RH2 and R

Finally, it should be noted that $RH1$ and $RH2$ include a random component during the assignment step. This is necessary to introduce the variability required to generate diverse starting solutions for the improvement heuristics.

A more detailed description of the construction heuristics, along with their pseudo-codes, is presented next.

Random Heuristic *R*

The random heuristic was designed and implemented to serve as an upper referential bound. As shown in Algorithm 2, tasks are first ordered randomly. Then, for each dispatch train, the first task is assigned to the feasible machine which has the best processing time for it. Also, the remaining tasks of the train are assigned immediately afterward to the same machine.

Algorithm 2 Heuristic R

Input: Dataset, random seed
Output: Proposed schedule and Lateness
random_J ←List of tasks j to be processed, ordered randomly
best_tproc[j] ←List of feasible machines to process task j, sorted by min$\{tproc_{jp}\}$
for j in *random_J* **do**
    if j is the first element of a dispatch train: **then**
        Add task j to the sequence of machine *best_m*
        Add all remaining tasks of j's dispatch train to the sequence of machine *best_m*
    else
        skip
    end if
end for
Calculate schedule times and metrics associated with tasks j, considering spacing time when it is necessary

Construction Heuristic *RH1*

Initially, tasks are ordered by their due date dd_j. Then, each task is assigned to a feasible machine to be processed. This choice prioritizes the minimum processing time, weighted by a probability factor p_G. It is important to notice that all the tasks belonging to a specific train may not be assigned to the same machine. However, the order between them, and thus the order in which they are received by the client, must be the same as the one defined in the initial dataset.

Algorithm 3 Heuristic RH1

Input: Dataset, random seed, p_G
Output: Proposed schedule and Lateness
sec_dd ←List of tasks to be process in each machine m, ordered by dd_j
best_tproc[j] ←List of feasible machines and their processing times, to process task j, sorted by min$\{tproc_{jp}\}$
for j in *sec_dd*: **do**
    machines_to_review ← List of feasible machines to process task j
    for i in range(*machines_to_review*): **do**
        $Probabilities\_j \leftarrow (1 - p_G) * *(valores - 1) * p_G$ ▷ Definition of weights
        $Probabilities\_j / = sum(Probabilities\_j)$ ▷ Normalization
        machine ← choose randomly among *machines_to_review*, weighted by *Probabilities_j*
        Remove *machine* from *machines_to_review*
        if The sequence of *machine* is not defined yet: **then**
            if j is not assigned yet and is the first task of its train: **then**
                Calculate schedule times of j as the first task of *machine* sequence
                Define the sequence of *machine* beginning with j, with their schedule times
                Remove j of the tasks waiting to be processed
            end if
        end if
    end for
end for

for j in *sec_dd*: **do**
    if Task j has not assigned yet: **then**
        $best\_lateness, best\_machine \leftarrow$ Auxiliary variables
        $machines\_to\_review \leftarrow$ List of feasible machines to process task j
        for i in range($machines\_to\_review$): **do**
            $Probabilities\_j \leftarrow (1 - p_G) * *(valores - 1) * p_G$ ▷ Definition of weights
            $Probabilities\_j / = sum(Probabilities\_j)$ ▷ Normalization
            $machine \leftarrow$ choose randomly among $machines\_to\_review$, weighted by $Probabilities\_j$
            Remove $machine$ from $machines\_to\_review$
            if The sequence of $machine$ is defined: **then**
                $jprev\_Sec \leftarrow$ previous task in the sequence
                if j is the first task of the train: **then**
                    $m\_disp \leftarrow$ Final processing time in the plant of task $jprev\_Sec$
                else
                    if The first task of their train was assigned to a machine: **then**
                        $Tatraso0 \leftarrow$ Final processing time of j - $jprev\_Sec + spacing - tproc_{machine,j}$
                        $m_disp \leftarrow max\{$Final processing time of task $jprev\_Sec, Tatraso0\}$
                    end if
                end if
                $Ti\_tentative \leftarrow max\{rh_{machine,j}, m\_disp\}$ ▷ Possible initial processing time for j
                $Tfc\_tentative \leftarrow Ti\_tentative + tproc_{machine,j}$
                $L\_tentative \leftarrow max\{0, (Tfc\_tentative - dd_j)\}$
                if $L\_tentative < best\_lateness$: **then**
                    $(best\_lateness, best\_i, best\_machine) \leftarrow (L\_tentative, i, machine)$
                    if There is not lateness: **then**
                        break
                    end if
                end if
            end if
        end for
        Calculate the scheduling times of task j as if it was assigned to *best_machine* sequence
        Assign j to *best_machine* sequence, with their respective times
        Remove j of the tasks waiting to be processed
    end if
end for

Construction Heuristic *RH2*

Heuristic RH2 follows essentially the same formulation as RH1. The only difference between them lies in the ordering criteria: while RH1 prioritizes tasks by due date dd_j (see Algorithm 3), RH2 orders them by release time r_{jp}. Thus, replacing *sec_dd* in Algorithm 3 with a list sec_{rh} ordered by rh_{jp}, is sufficient to obtain the pseudo-code and exact formulation for heuristic RH2.

3.2.3. Improvement Heuristics

After the outputs of the construction heuristics were analyzed and compared to the results and metrics of the optimization model and random heuristic, they were used as inputs for the improvement heuristics.

Thus, this section will detail the improvement heuristics, which take as input a schedule obtained as initial solution of construction heuristic $RH1$ and $RH2$ shown in the previous section, and produce an improved proposal schedule as output.

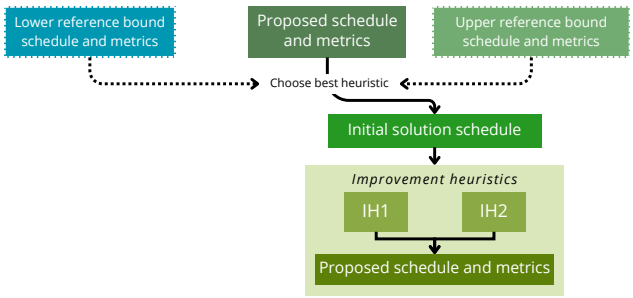


Figure 3.7: Flow diagram of the Improvement Heuristics

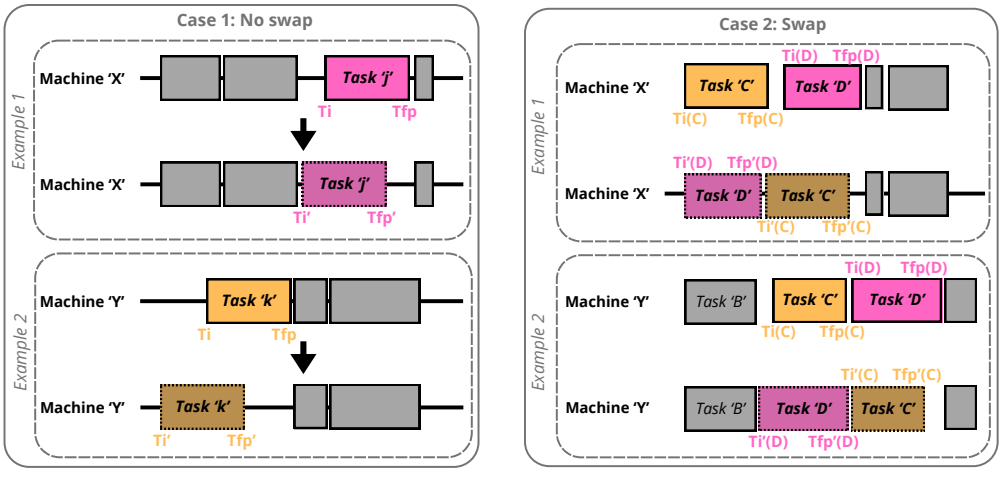
The idea behind these type of heuristics is to attempt to improve the solution. In this case, these algorithms evaluate the possibility of doing task movements subject to feasibility constraints. Moreover, a movement is applied only if it meets all feasibility conditions and is beneficial, that is, it brings the result of the principal metric, Lateness, closer to the optimum given by the optimization model.

Two improvement heuristics were designed, differing in the scope of their movements: $IH1$ swaps or advances tasks belonging to the sequence of the same machine, while $IH2$ evaluates similar movements but between the sequences of two different machines.

The sets and parameters are the same as the ones used in the construction heuristics, except for the probability-related parameter, p_G. Dispatch trains are handled consistently with the construction heuristics, preserving the same assignment rules and constraints for scheduling their tasks. A more detailed description of the improvement heuristics, along with their pseudo-codes, is presented next.

Improvement Heuristic *IH1*

The main idea of this heuristic is analyze each machine's sequence separately and evaluate the feasibility and convenience of moving tasks within it. To do that, two cases were defined, which are described below and illustrated in Figure 3.8.



(a) No swap case

(b) Swap case

Figure 3.8: Possible task movements allowed in heuristic *IH1*

1. No swap: Selects a task j and evaluates the possibility of advancing it earlier in the sequence, to reduce its lateness, without disrupting the original schedule for the rest of the tasks.

2. Swap: Selects a pair of consecutive tasks, j and $j+1$, and evaluates the possibility of swapping between them, to reduce their combined lateness, without disrupting the original schedule for the rest of the tasks.

It is important to mention that, in each case, a series of conditions have to be checked to ensure the feasibility of the reassignment. In particular, when a movement of a single or pair of tasks is being evaluated, it is necessary to look backward within the machine sequence, as well as both forward and backward within the dispatch train.

Finally, a "Swap condition" is also checked, defined by the expression presented in Equation 3.14, as the difference between the relative positions of tasks in their dispatch train, multiplied by Δ_L factor, where Δ_L factor corresponds to a subtraction between the original and new lateness of task, as shown in Equations 3.15 and 3.16 below.

$$(Train_D\_lenght - index_D\_Train_D) \cdot \Delta_{LD} - (Train_C\_lenght - index_C\_Train_C) \cdot \Delta_{LC} \quad (3.14)$$

$$\Delta_{LD} = max\{0, L_D - L'_D\}, L'_D = max\{0, Tfc'_D - dd_D\} \quad (3.15)$$

$$\Delta_{LC} = max\{0, L'_C - L_C\}, L'_C = max\{0, Tfc'_C - dd_C\} \quad (3.16)$$

The pseudo-code of this heuristic is presented in Algorithm 4.

Algorithm 4 Heuristic IH1

Input: Initial feasible schedule with their Lateness, smax
Output: Proposed schedule and Lateness

for each *machine* in the initial schedule used as input: **do**
  if there is a sequence of tasks assigned to *machine*: **then**
    for i in the sequence of *machine*: **do**
      ▷ CASE 1: NO SWAP (ADVANCE TASK) :
      ▷ Previous task in the sequence condition :
      if j is the first task of the sequence: **then**
        $Tfp\_ant \leftarrow 0$ ▷ There is no previous task
      else
        $Tfp\_ant \leftarrow$ final processing time in the plant, of the previous task $j-1$
      end if
      $min\_start \leftarrow max\{rh_{mj}, Tfp\_ant\}$ ▷ Next task in the train condition:
      if j is the last task of the train: **then**
        $Tfc\_suc \leftarrow 0$ ▷ There is no next task in their train
      else
        $Tfc\_suc \leftarrow$ final processing time in the plant, of the next task in the train - *tprec* - *smax*
      end if ▷ Previous task in the train condition:
      if j is the first task of their dispatch train: **then**
        $Tfc\_pred \leftarrow 0$ ▷ There is no previous task in the train
      else
        $Tfc\_pred \leftarrow$ final processing time in the plant, of the previous task in their train + *tprec*
      end if
      Define new schedule times for task j ($Ti\_newD, Tfp\_newD, Tfc\_newD$, L_newD) and updated their values in the solution

▷ CASE 2: SWAP (BETWEEN TASKS "C" AND "D") :
if j is not the first task in the sequence of $machine$: **then**
▷ Previous task in the sequence condition, of task j ("D") :
if j is the second task of the sequence: **then**
$Tfp\_antD\_ant \leftarrow 0$ ▷ There is no task $(j-2)$ in the sequence
else
$Tfp\_antD\_ant \leftarrow$ final processing time, of the previous previous task in the sequence, $(j-2)$
end if
$min\_startD \leftarrow max\{rh_{mj}, Tfp\_antD\_ant\}$
▷ Next task in the train condition, of task j ("D") :
if j is the last task of the train: **then**
$Tfc\_sucD \leftarrow 0$ ▷ There is no next task in their train
else
$Tfc\_sucD \leftarrow$ final processing time of the next task - $tprec$ - $smax$
end if
▷ Previous task in dispatch train condition, for task j ("D") :
if j is the first task of their dispatch train: **then**
$Tfc\_predD = 0$ ▷ There is no previous task in the train
else
$Tfc\_predD \leftarrow$ final processing time of the previous task in their train + $tprec$
end if
Define new possible schedule times for task j (Ti_newD, Tfp_newD, Tfc_newD, L_newD)
▷ Previous task in the sequence condition, of task $j-1$ ("C") :
$min\_startC \leftarrow max\{rh_{m(j-1)}, Tfp\_newD\}$
▷ Next task in the train condition, of task $j-1$ ("C") :
if $j-1$ is the last task of the train: **then**
$Tfc\_sucD \leftarrow 0$ ▷ There is no next task in their train
Set $TfcMax\_sucD$ as an auxiliary value
else
$Tfc\_sucD \leftarrow$ final processing time in the plant, of the next task - $tprec$ - $smax$
$TfcMax\_sucD \leftarrow$ final processing time in the plant, of the next task - $tprec$
end if
▷ Previous task in dispatch train condition, for task $j-1$ ("C") :
if $j-1$ is the first task of their dispatch train: **then**
$Tfc\_predC \leftarrow 0$ ▷ There is no previous task in the train
Set $TfcMax\_predC$ as an auxiliary value
else
$Tfc\_predC \leftarrow$ final processing time in the plant, of the previous task in their train + $tprec$
$TfcMax\_predC \leftarrow$ final processing time in the plant, of the previous task in their train + $tprec$ + $smax$
end if
Define new possible times for task $(j-1)$ (Ti_newC, Tfp_newC, Tfc_newC, L_newC)
$TfcMax\_newC \leftarrow min\{TfcMax\_sucD, TfcMax\_predC\}$
if $Tfc\_newC <= TfcMax\_newC$: **then** ▷ Swap feasibility
if $swap\_criteria > 0$: **then** ▷ Swap convenience
Swap tasks, updating the result
end if
end if
end if
end for
end if
end for

Improvement Heuristic *IH2*

This heuristic evaluates whether it is possible to move tasks between machines. To achieve that, a task is removed from its original sequence and inserted into a new one on a different machine. As illustrated in Figure 3.9, this insertion could be at the end of the new sequence or between tasks.

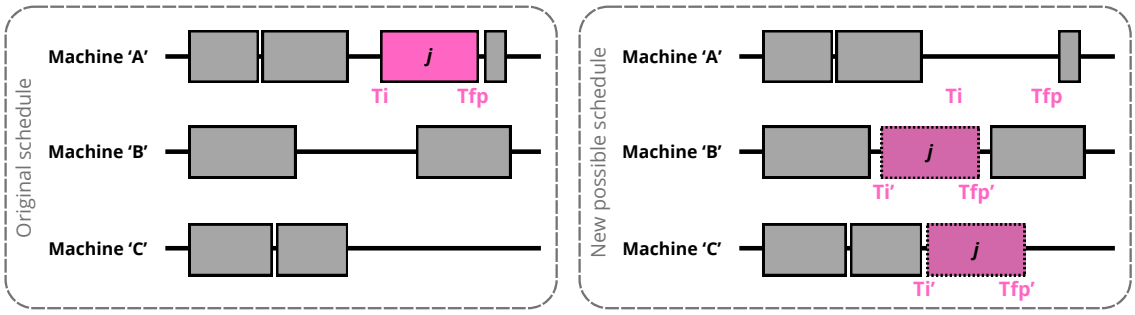


Figure 3.9: Possible task movements allowed in heuristic *IH2*

As shown in Algorithm 5, heuristic $IH2$ begins by choosing the first task in the sequence of the machine with the longest idle time, defined as the amount of time a machine is available to process tasks but is not actively processing any. Then, it selects the task j with the highest lateness, which becomes the candidate for reassignment.

A tentative insertion of task j into a new sequence is defined by computing its scheduling times under different scenarios. Then, the feasibility of the insertion is evaluated; that is, whether task j can be placed in a new sequence, without disrupting the original schedule.

To evaluate the feasibility of the insertion, a series of conditions were taken into consideration. They are described in Section 3.3.3.

If a potential insertion is defined and found feasible, the algorithm evaluates whether executing the movement is beneficial, by comparing the original lateness of the task with the new one obtained after reassignment.

Algorithm 5 Heuristic IH2

Input: Initial feasible schedule with their Lateness, *smax*, *while_timeout*
Output: Proposed schedule and Lateness
1: *machines_review* ← auxiliary list with machines not yet checked, ordered by idle time
2: **while** *len*(*machines_review*) > 0: **do**
3: *actual_machine* ← machine with highest idle time
4: **if** *actual_machine* in *machines_review*: **then**
5: *Jm_sorted* ← List of tasks that can be processed by *actual_machine*, butnot in its current sequence, ordered by decreasing lateness
6: **for** *task* in *Jm_sorted*: **do**

▷ DEFINE A TENTATIVE INSERTION OF TASK j IN *actual_machine* :
Initialize $Ti\_tentative \leftarrow rh_{actual\_machine,j}$
Initialize $Tfc\_tentative \leftarrow Tfc - Ti + Ti\_tentative$
▷ Case 1: if j is the first of their train, check condition forward :
if j is the first of the train and len(train)>1: **then**
  if $Tfc\_tentative + tprec + smax$: **then** ▷ does not satisfy $smax$ condition
    $Tfc\_tentative \leftarrow Tfc_{j+1} - tprec - smax$
    $Ti\_tentative \leftarrow Ti + Tfc\_tentative - Tfc$
  end if
▷ Case 2: if j is the last of their train, check condition backward :
else if j is the last of their train: **then**
  if $Tfc_{j-1} + tprec > Tfc\_tentative$: **then** ▷ does not satisfy times conditions
    $Tfc\_tentative \leftarrow Tfc_{j-1} + tprec$
    $Ti\_tentative \leftarrow Ti + Tfc\_tentative - Tfc$
  end if
▷ Case 3: if j is neither the first nor last of the train, check conditions forward and backward
else
  if len(train)>1: **then**
    ▷ Insert task j to the right :
    if $Tfc\_tentative + tprec + smax < Tfc_{j+1}$ or $Tfc_{j-1} + tprec > Tfc\_tentative$: **then**
      $Tfc\_tentative \leftarrow max\{Tfc_{j-1} + tprec, Tfc_{j+1} - tprec - smax\}$
      $Ti\_tentative \leftarrow Ti + Tfc\_tentative - Tfc$
    ▷ Insert task j to the left :
    else if $Tfc\_tentative + tprec > Tfc_{j+1}$ or $Tfc_{j-1} + tprec + smax < Tfc\_tentative$: **then**
      $Tfc\_tentative \leftarrow min\{Tfc_{j+1} - tprec, Tfc_{j-1} + tprec + smax\}$
      $Ti\_tentative \leftarrow Ti + Tfc\_tentative - Tfc$
    end if
  end if
end if
Calculate *insertion_index*, according to $Ti\_tentative$

▷ FEASIBILITY OF THE INSERTION :
if *insertion_index* is par: **then**
  $Tfp\_tentative \leftarrow Tfp - Ti + Ti\_tentative$
  if If the insertion is at the end of the sequence: **then**
    $Tfc\_tentative \leftarrow Tfc - Ti + Ti\_tentative$
    $L\_tentative \leftarrow max\{Tfc\_tentative - dd_j, 0\}$
    if $L\_actual > L\_tentative$: **then**
      Insert j to *actual_machine* and update solution
      Remove task j from their original sequence
    end if
  else if the insertion is between tasks of the sequence: **then**
    $Tfc\_tentative \leftarrow Tfc - Ti + Ti\_tentative$
    $L\_tentative \leftarrow max\{Tfc\_tentative - dd_j, 0\}$
    if $L\_actual > L\_tentative$: **then**
      Insert j to *actual_machine* and update solution
      Remove task j from their original sequence
    end if
  end if
end if
end for
Remove *actual_machine* from *machines_review*
end if
end while

3.2.4. Metaheuristic

This section presents the metaheuristic, which takes the initial dataset as input and produces a proposed schedule as output.

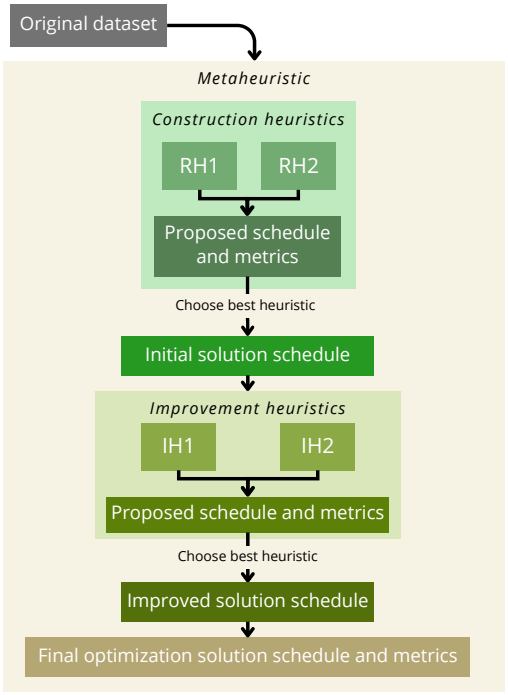


Figure 3.10: Flow diagram of the Metaheuristic

This metaheuristic was designed following a GRASP approach. Firstly, a construction heuristic is selected at random to obtain an initial feasible solution. Subsequently, a recursion is initialized, where in each iteration an improvement heuristic is randomly chosen and applied in an attempt to enhance the current solution. This process is executed iteratively, until the solution reaches a predefined improvement threshold or the maximum number of iterations defined as part of the input is reached.

Since the metaheuristic integrates both construction and improvement heuristics, the parameters and sets used are the same as the ones described before in their respective sections. This also applies to all the restrictions and movement types that each type of heuristic considers.

As mentioned above, the first step is to randomly select and execute a construction heuristic, saving its results. This schedule and performance metrics become the input for the improvement heuristic, which is also chosen randomly, executed, and whose output is also stored.

It is important to mention the usage of a random seed. The construction heuristics have a random component, which introduces variability into their results. Thus, in the metaheuristic, an initial random seed was defined for the first iteration, and then updated in each subsequent iteration within the recursion. Consequently, in each iteration, the metaheuristic begins from a different initial solution before applying improvements.

Regarding the second part of the metaheuristic, an auxiliary variable was defined to monitor the improvement in the results until a predefined tolerance value is reached. Following this approach, within a single iteration the improvement heuristics have to be executed more than once. Thus, in each inner iteration, an improvement heuristic is randomly chosen and executed using as input the output obtained during the previous inner iteration. This description corresponds to the recursive loop inside the *while* structure in Algorithm 6.

Therefore, one iteration of the metaheuristic consists of one execution of a construction heuristic to obtain an initial feasible solution, plus multiple executions of the improvement heuristics, where in each iteration within the improvement loop, the solution is brought closer to the optimum given by the optimization model. This final solution is saved. Then, the metaheuristic iterates, returning the best solution found across all iterations.

Algorithm 6 Metaheuristic

Input: Dataset, $smax$, p_G, $seed\_start$, $n\_iterations$, $GRASP\_tolerance$, $tolerance$, $timeout\_for$, $while\_timeout$
Output: Proposed schedule and Lateness
Set $Lateness\_total$ with an initial value
for i in range($n\_iterations$): **do**
    $construction\_heuristic \leftarrow$ choose randomly between $RH1$ and $RH2$
    Set a seed that changes in each iteration, as a function of $seed\_start$
    Execute the heuristic and save the results
    Set initial value for $improvement$
    Set $k = 0$
    while $improvement > tolerance$: **do**
        $k = k + 1$
        $improvement\_heuristic \leftarrow$ choose randomly between $IH1$ and $IH2$
        Execute the heuristic and save the results
        ▷ Calculate new value for $improvement$:
        $improvement \leftarrow |previous\_lateness - actual\_lateness| * 100/(previous\_lateness + 1)$
    end while
    if convergence is reached, e.g., Lateness difference $< GRASP\_tolerance$: **then**
        break
    end if
    if $Lateness\_total > Lateness\_actual$: **then**
        Update values for $best\_solution, Lateness\_total$ and $iteration$
    end if
    if Maximum time $timeout\_for$ is reached: **then**
        break
    end if
end for
if $Lateness\_total > Lateness\_actual$: **then**
    Update values for $best\_solution, Lateness\_total$ and $iteration$
end if

3.3. Implementation

3.3.1. Optimization Model

Tools and Coding

The optimization model was implemented in Python using Gurobi Optimizer (version 10.0.3) within a Jupyter Notebook environment. The experiments were executed on a computer equipped with an Intel Core i5-8250U CPU @1.60Hz and 8GB of RAM.

To ensure the attainment of optimal solutions, no restrictions such as time limits or node limits were imposed. In fact, Feasibility and Numeric Focus parameters were configured as shown in Table 3.7, to achieve the same objective. However, due to the complexity of the problem and processing capacity limitations of the computer, the model was executed for 80 different instances, as explained in Section 3.1.2. This allowed the model to achieve optimal solutions within acceptable computational times and to compare the results in different scenarios, testing the robustness of the model.

Parameter	Value
Feasibility Tolerance	0,000000001
Numeric Focus	3

Table 3.7: Parameters used in Gurobi for the optimization model

Sets and parameters values

To ensure the existence of correctly formed sequences, creating fictitious initial and final nodes was necessary, as described in Section 3.3.1. The remaining sets were created using the original data or a combination of the sets already defined.

It is important to mention that the parameter values shown in Table 3.4 were explicitly provided in the previously described dataset. However, there are five of them specifically created for the implementation of the model.

All parameters with their respective values are presented in Table 3.8.

Symbol	Value
dd_j	`due_date_day` + `due_date_time`, in minutes, after the transformation described in Section 3.1.1
r_{jp}	$dd_j - tproc_{jm} - h$, in minutes
$tprec_j$	`spacing`, in minutes
$tcarga_{pj}$	`load_time_pj`, in minutes
$tproc_{jm}$	`travel_time_pm` + `load_time_pm`, in minutes
w_j	$1, \forall j \in J$
$smax$	180 minutes
h	15 minutes
M	$3 \cdot \max_{j \in J} dd_j$

Table 3.8: Parameter values used for model implementation

The task weights w_j were set to 1 for all tasks, for simplicity. The values of $smax$ and h were chosen according to industry practices, in the same way that r_{jp} was calculated based on typical definitions for this type of problem.

Finally, M represents the mathematical Big M parameter used in optimization models. Thus, it was defined to be instance-dependent, to be an appropriate bound for the constraints of the problem.

Dummies

Fictitious initial nodes were created to define the starting points of the task sequences that each machine processes. Specifically, ten rows were added to the dataset, simulating the first task of the sequence in each machine, with the following characteristics:

- The columns `spacing`, `load_time_p1` - `load_time-p10` and `unload_time`, were set to 0.
- Column `order_id` was filled with identifiers created to link each fictitious node with its corresponding machine.
- Columns `travel_time_p1` - `travel_time_p10` and `return_time-p1` - `return_time_p10`, were set to 0 for the machine corresponding to the initial node, and NA in all other cases. For example, the initial node of machine 1 has `load_time_p1` = `return_time_p1` = 0 and NA's in all the other columns related to travel and return time.

Analogously, ten rows of fictitious final nodes were created and added to the dataset.

3.3.2. Heuristics

The heuristics were implemented in Python within a Jupyter Notebook environment. The computational experiments were carried out on a computer equipped with an Intel Core i5-8250U CPU @1.60Hz and 8GB of RAM.

The heuristics were executed for the same 80 instances mentioned before, in Section 3.1.2. Furthermore, the parameters used were set to the same values as the ones used by the model, shown in Table 3.8, complemented by the ones belonging specifically to the heuristics. All of them are presented in Table 3.9.

Parameter	Values					
	R	RH1	RH2	IH1	IH2	Metaheuristic
random seed	42	42	42	-	-	-
P_G	-	0.3	0.3	-	-	0.3
$smax$	-	-	-	180	180	180
$while\_timeout$	-	-	-	-	300	300
$seed\_start$	-	-	-	-	-	5
$n\_iterations$	-	-	-	-	-	100
$GRASP\_tolerance$	-	-	-	-	-	0.001
$timeout\_for$	-	-	-	-	-	300
$tolerance$	-	-	-	-	-	0.001

Table 3.9: Parameter values used in the heuristics implementation

All these parameter values were defined to ensure comparability with the optimization model results. Also, executing the heuristics across the different instances allowed testing their robustness and achieving near-optimal solutions within an acceptable computational time.

3.3.3. Solution Validation and Feasibility Check

In addition to using the same parameters, sets, instances, and other elements to ensure compatibility between the optimization model and the heuristics, a solution validation and feasibility check were applied.

Following the problem analysis, a set of necessary and sufficient conditions for its resolution was defined. These conditions are presented below:

- Correct assignment of tasks:
  - → All tasks are processed
  - → All tasks are uniquely assigned
  - → All tasks are assigned to feasible machines
- Times coherence:
  - → All tasks begin their processing after their release time
  - → All tasks finish processing after their start processing time
  - → All tasks are delivered to the client after their start processing time
- Machines availability:
  - → All tasks begin their processing while the machine is available
- Dispatch trains:
  - → All tasks arrive to the client before the subsequent one of their train
  - → All consecutive tasks in a train arrive to the client respecting the spacing time between them
  - → The client's waiting time between any pair of consecutive tasks of a train is less than $smax$

Keeping that in mind, a series of functions were designed to verify that all the solutions satisfy these conditions. Therefore, the validity and coherence of each solution is corroborated, ensuring the comparability between the outputs of the different methods.

These functions were applied to the solutions obtained by the construction and improvement heuristics. Since the metaheuristic integrates both, if each part was verified individually, then the entire algorithm also satisfies the conditions.

Although the optimization model should inherently satisfy all these conditions if correctly built, the checking functions were still applied to verify that this was indeed the case.

3.4. Results

3.4.1. Schedules

The outputs of the optimization model and the heuristics are sufficient to build proposed schedules. In particular, these results enable the construction of three types of Gantt charts:

1. Gantt Chart of Jobs: shows the start and finish processing times of each job on its assigned machine within the plant
2. Gantt Chart of Dispatches: shows the start processing time in the plant and the moment when each part of the dispatch train order was received by the client
3. Gantt Chart of Jobs from the customer's perspective: shows the time interval during which each part of the train order was at the client, including both arrival and the unloading time.

Examples of each type of Gantt chart are presented in Figures 3.11, 3.12, and 3.13. These graphs illustrate the schedules obtained after applying the metaheuristic to the tasks in instance *Tuesday_50*.

As shown in Figure 3.11, all tasks are assigned to a specific machine to be processed in the plant, ensuring compliance with machine availability and sequence constraints. Furthermore, in those three figures it is also possible to notice that the schedules respect the spacing time and ordering of all the tasks belonging to the same dispatch train. Thus, the proposed schedules presented are indeed feasible solutions. The same type of analysis was carried out in more detail for each instance.

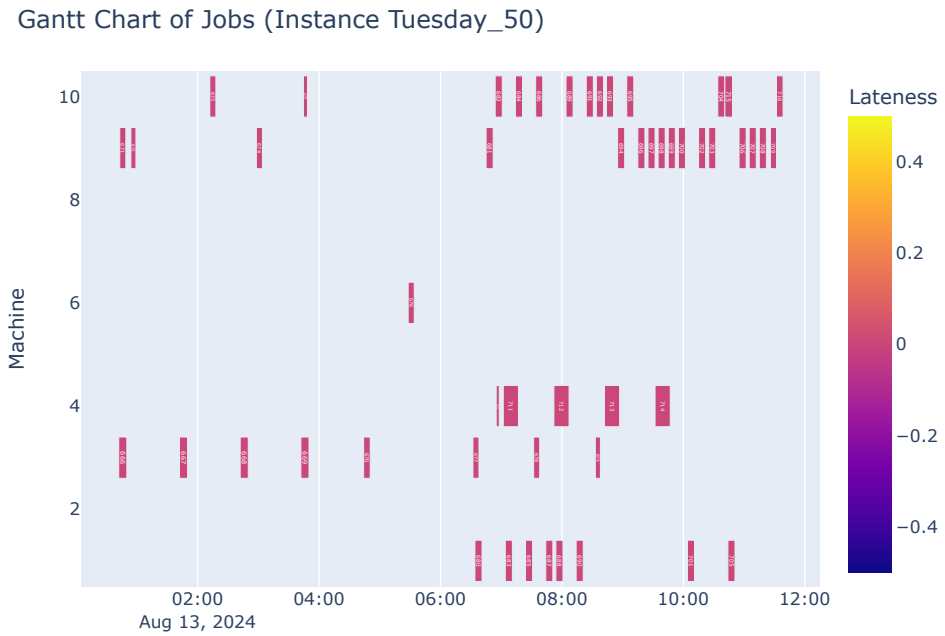


Figure 3.11: Example of Gantt Chart of Jobs, obtained for *Tuesday_50* using the Metaheuristic

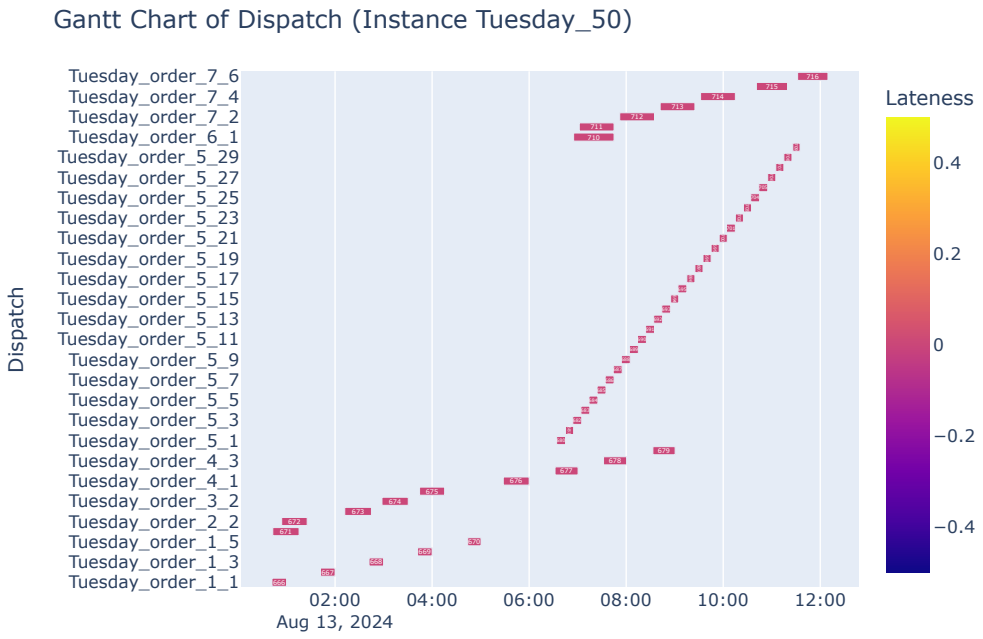


Figure 3.12: Example of Gantt Chart of Dispatches, obtained for *Tuesday*_50 using the Metaheuristic

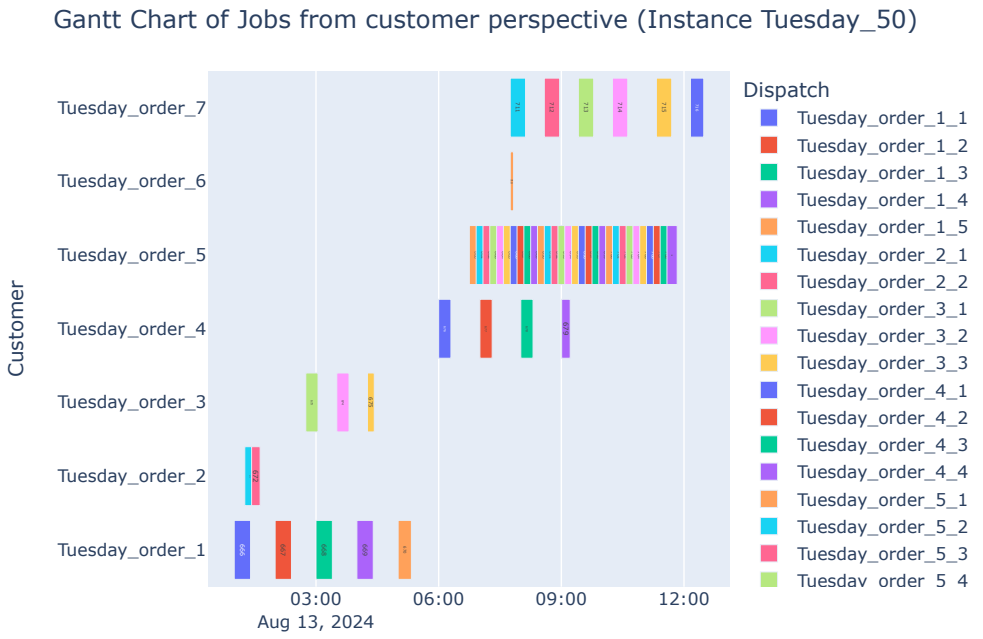
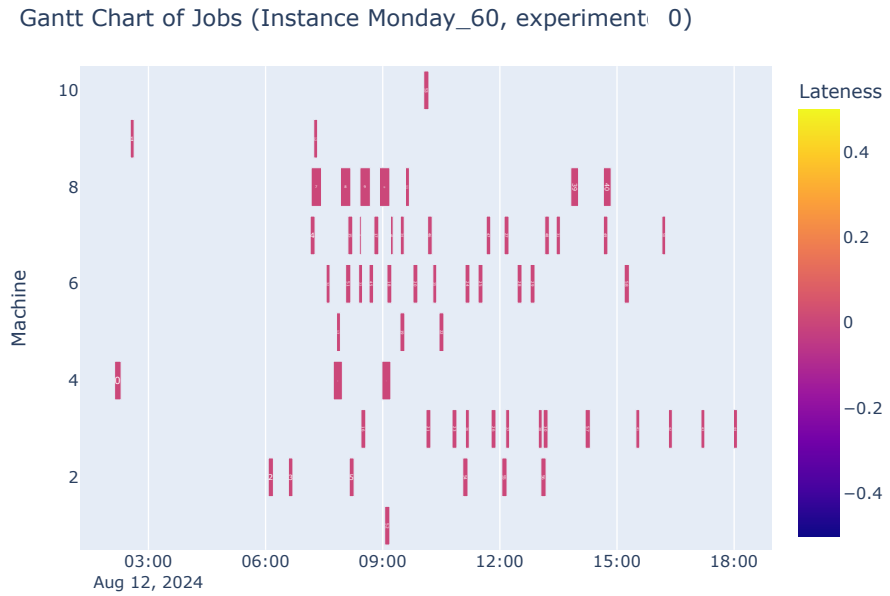
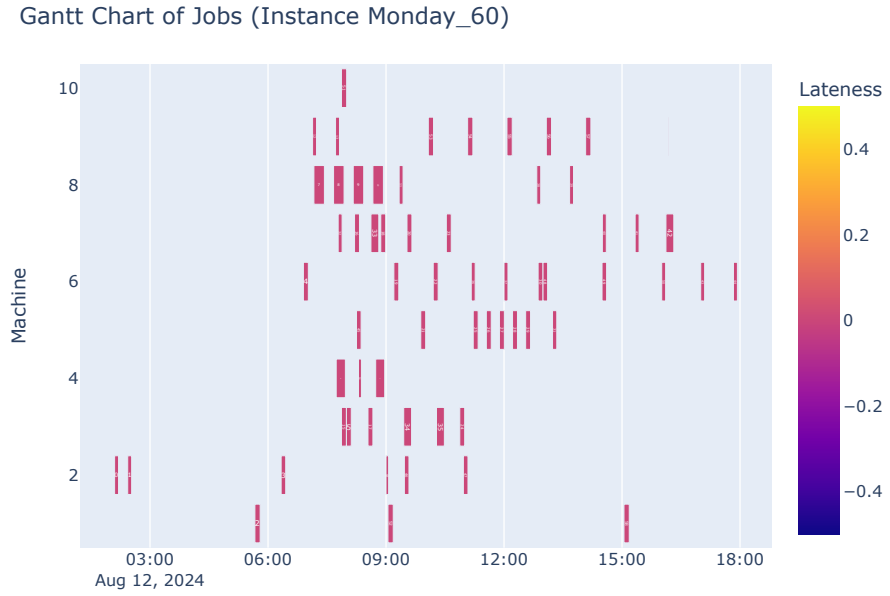


Figure 3.13: Example of Gantt Chart of Jobs from the client's perspective, obtained for *Tuesday*_50 using the Metaheuristic

Moreover, the three type of charts are also useful for comparing the different methods. In fact, it is possible to notice that the optimization model and the metaheuristic lead to similar schedules. An example is presented in Figure 3.14.



(a) Gantt Chart of Jobs obtained for $Monday\_60$ using the optimization model



(b) Gantt Chart of Jobs obtained for $Monday\_60$ using the GRASP metaheuristic

Figure 3.14: Comparison between optimization model and metaheuristic schedules, obtained for instance $Monday\_60$

From the comparison between the model in Figure 3.14.a and the metaheuristic in Figure 3.14.b, it is possible to observe that tasks are scheduled to be processed at almost the same points in time. For instance, jobs 7, 8, 9, 10, 11 and 39 are assigned to machine 8 in both cases, while job 52 to machine 1. Furthermore, all tasks in both schedules have zero lateness, resulting in the same objective value in both cases.

3.4.2. Lateness

The results of the total lateness for each instance, obtained from the execution of the optimization model and heuristics, are presented in Table 3.10 below. These values were computed as the sum of the lateness of every task in the proposed schedule and then averaged across all instances corresponding to the same day.

In the first place, it can be observed that the optimization model's values are the lowest for each day, compared with the other methods. Conversely, the random heuristic has the highest values. Both results were expected with how both methods were defined.

With respect to the other construction heuristics, $RH1$ and $RH2$, there is no clear trend indicating which of the two approaches performs better across all instances. In fact, $RH1$ yields better results on average for the instances of Monday, Wednesday, and Thursday, whereas $RH2$ achieves lower values on average for Tuesday and Friday. It is also possible to observe that both $RH1$'s and $RH2$'s values lie between the optimization model and the random heuristic for all cases, aligning with their intended role as upper and lower reference bounds, respectively. Furthermore, the construction heuristics produce solutions that are close enough to the optimal values given by the model, but at the same time, not too tight. Specifically, when measuring the relative gap between the optimal solution and the random heuristic, $RH1$ and $RH2$ achieve approximately 17.71% and 20.87% of that gap, respectively. This is necessary to ensure that the improvement heuristics have sufficient space to enhance the solution further.

The improvement heuristics have two rows of results each, corresponding to the construction heuristic used to build the initial feasible solution. By comparing these values, it can be observed that the previously identified pattern between the behavior of $RH1$ and $RH2$ across the days remains consistent for each improvement heuristic: they have better results when using $RH1$ on the days where $RH1$ outperforms $RH2$, and vice versa.

Regarding the comparison between improvement heuristics for the same input, $IH1$ consistently achieves results on average closer to the optimum than $IH2$, for all days. However, $IH2$ also improves upon its input values. It is important to mention that these results correspond to a single iteration of each method. Consequently, both improvement heuristics were integrated into GRASP, since the recursive nature of the metaheuristic may alter their relative contributions.

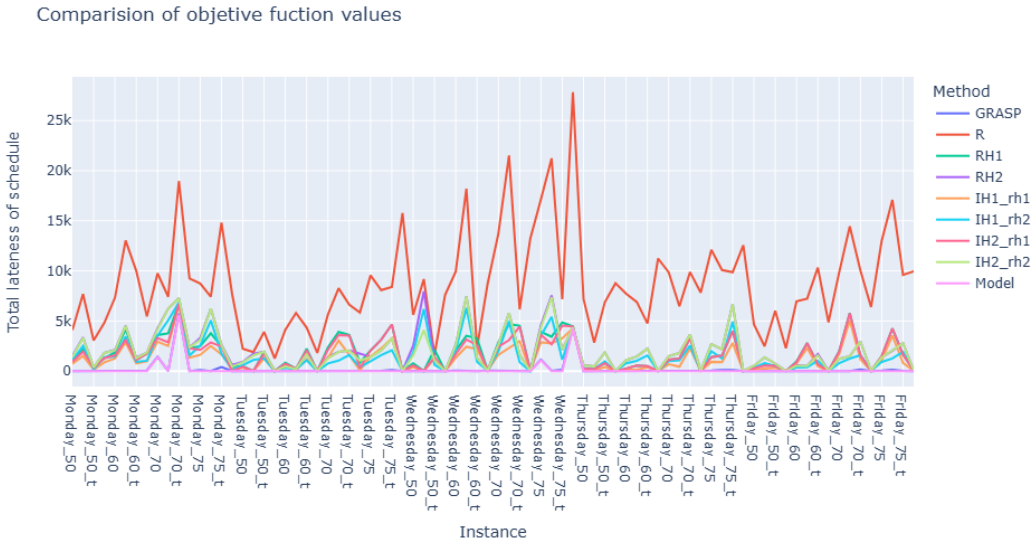
Finally, the values in the GRASP row are the closest to the optimal solutions of the optimization model. In fact, the metaheuristic improved the solution by reducing lateness by approximately 89.57% compared to the improvement heuristics, and by 91.50% compared to the construction heuristics. As a result, the average gap between the GRASP metaheuristic

and the optimization model was only 8.3%, representing an average absolute improvement of 2968.8 percentage points over the random heuristic R.

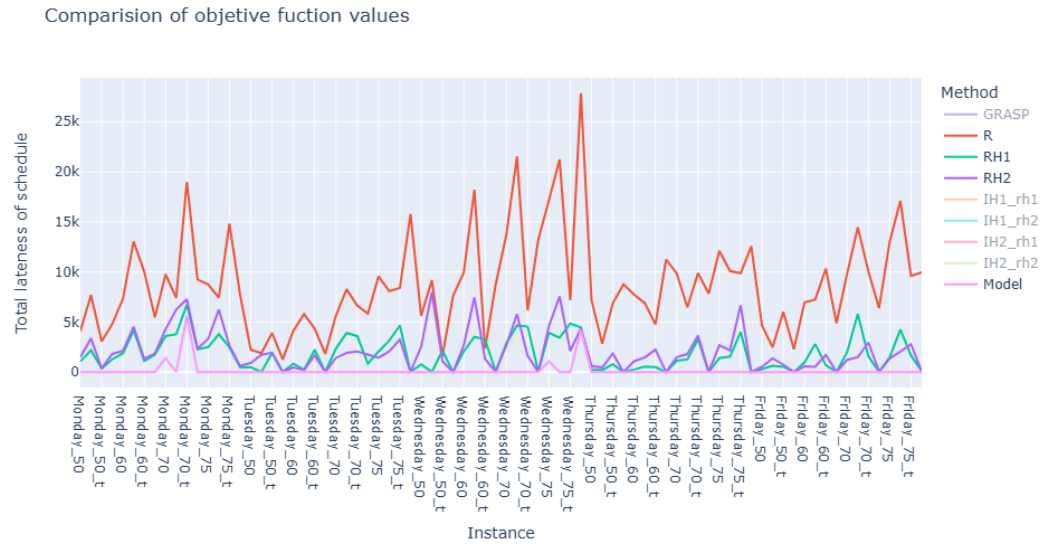
Method/Day	Monday	Tuesday	Wednesday	Thursday	Friday
Model	351.7812	0.0000	288.0781	0.0000	0.0000
R	7477.7656	5389.9844	11600.3125	9209.7969	8620.6094
RH1	2252.1250	1429.3438	2400.5625	753.9219	1182.7031
RH2	2798.6562	1052.3906	3109.2344	1459.2812	915.1875
IH1_rh1	1626.6250	893.8125	1542.6562	458.4062	920.5312
IH1_rh2	2147.5469	661.0625	2333.6406	1014.3594	594.2188
IH2_rh1	1974.0625	1336.0000	2073.3750	753.9219	1137.9844
IH2_rh2	2681.9375	997.9219	2783.6094	1453.6562	904.5938
GRASP	387.0000	8.5625	306.9531	16.7188	18.1875

Table 3.10: Average total lateness obtained for each day after applying the optimization model and heuristics

To compare the heuristics at a deeper level, a series of comparative graphs is presented. The first ones align with the information reported in Table 3.10: as expected, the random heuristic R yields the worst results in terms of the objective value, while the metaheuristic $GRASP$ is the closest to the optimum given by the model. The remaining heuristics perform within an intermediate range, as shown in Figure 3.15.a. In particular, Figure 3.15.b provides a more detailed view of the construction heuristics, showing that $RH1$ outperforms $RH2$ in some instances, whereas $RH2$ performs better in others. Nevertheless, both remain within the bounds established by R and the model. Finally, Figure 3.15.c highlights how close the values of the objective function reached by the metaheuristic $GRASP$ are to the optimal solution: in fact, in most cases, they are identical.

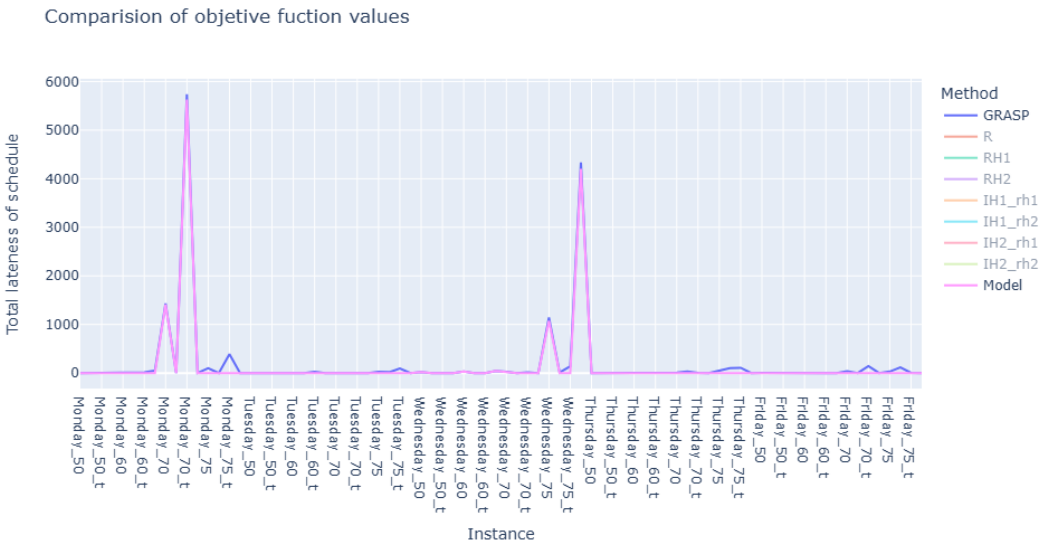


(a) Comparison of the Total Lateness objective function value obtained using different methods on each instance



(b) Comparison of the Total Lateness of the construction heuristics, after being executed on each instance

Figure 3.15: Comparison of the objective function value, Total Lateness, obtained after executing the different methods on each instance



(c) Comparison of the Total Lateness between the metaheuristic *GRASP* and optimization model, after being executed on each instance

Regarding the minimum lateness of tasks in the different proposed schedules, all methods achieve zero for all instances. However, the maximum varies across methods. In fact, as shown in Figure 3.16, both construction and improvement heuristics have maximum values of lateness in instances where the optimization model achieves zero. In contrast, the metaheuristic *GRASP* follows the pattern of the optimal values. On the other hand, the graph shows that in peak cases, such as instance *Monday*_70_*t*, all methods reach a similar maximum value.

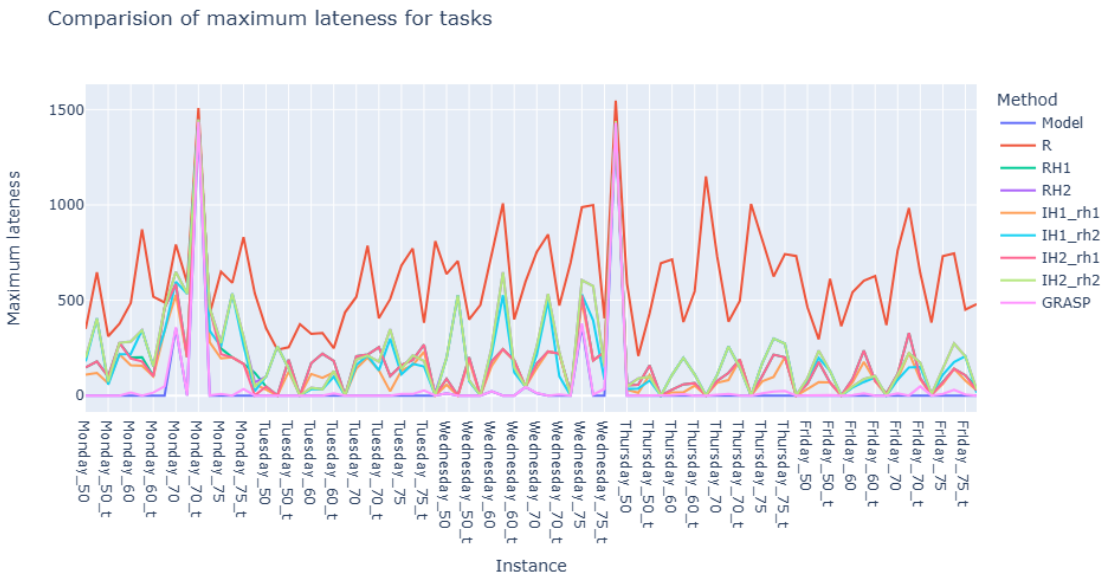


Figure 3.16: Comparison of the Maximum Lateness of tasks in schedules obtained after executing the different methods on each instance

It is important to mention that while the results for all instances are included in Figures 3.15 and 3.16, only half of their labels are displayed on the X-axis to enhance readability.

It is worth noting the presence of peaks for lateness values in all the previous graphs, which warrants further analysis. To study this, an aggregated representation was constructed and is presented in Figure 3.17 below.

(a) Comparison of the average Total Lateness grouped by subset range, for all methods

(b) Comparison of the average Total Lateness grouped by subset range, for all methods except *R*

Figure 3.17: Comparison of the average Total Lateness, obtained after executing the different methods on each instance, grouped by subset range

From Figures 3.17 and 3.17.b, it can be observed that all heuristics, except $GRASP$, reach their highest average total lateness in instances built as subsets of tasks with dispatch times concentrated around midday, while the lowest values are found in the randomly generated subsets. Additionally, the improvement heuristics $IH1$ and $IH2$ inherit the behavioral pattern of the construction heuristic used as their initial solution. For example, $RH2$ follows a similar trend to $IH2\_rh2$ and $IH1\_rh2$. Conversely, the metaheuristic $GRASP$ and the optimization model display a closely aligned trend, distinct from the rest of the methods.

3.4.3. Execution Time

The average runtime per day for each method is presented in Table 3.11 below. Each value represents the mean execution time across the 16 instances corresponding to each weekday.

	Monday	Tuesday	Wednesday	Thursday	Friday
Model	916.5498	1956.6524	1661.8729	877.1646	1207.0398
R	0.0469	0.0443	0.0511	0.0186	0.0178
RH1	0.2729	0.2370	0.2610	0.1362	0.1332
RH2	0.1829	0.1651	0.1812	0.0950	0.0891
IH1_rh1	0.7764	0.7950	0.7240	0.3474	0.3581
IH1_rh2	0.7658	0.8303	0.7735	0.3525	0.3517
IH2_rh1	4.6157	5.1896	6.4608	3.2903	3.1656
IH2_rh2	4.5481	5.2472	6.5151	3.1851	3.1504
GRASP	138.9968	128.4543	170.6291	116.3058	136.1083

Table 3.11: Average runtime in seconds for each method, after being executed in all instances of each day

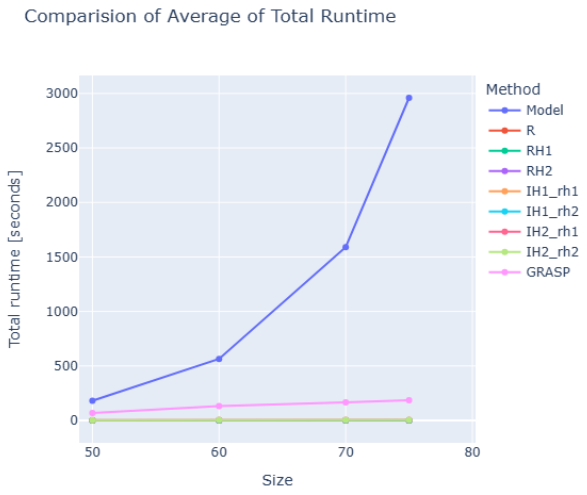
As expected, the optimization model has the largest execution time on average per day. In particular, its runtime ranges from a minimum of approximately 15 minutes on average for instances belonging to Thursday to a maximum of around 33 minutes on average for instances corresponding to Tuesday. Conversely, the random heuristic R achieves the lowest execution time, requiring less than 0.1 seconds on average per day.

Regarding the construction heuristics, their average runtime per day remains below 1 second, specifically under 0.3 seconds, which is three times higher than R but still significantly lower than the execution time of the model. When comparing $RH1$ and $RH2$, the latter demonstrates superior computational efficiency, requiring approximately 30% less time on average. In other words, $RH2$ takes only 70% of the time needed by $RH1$ to be executed.

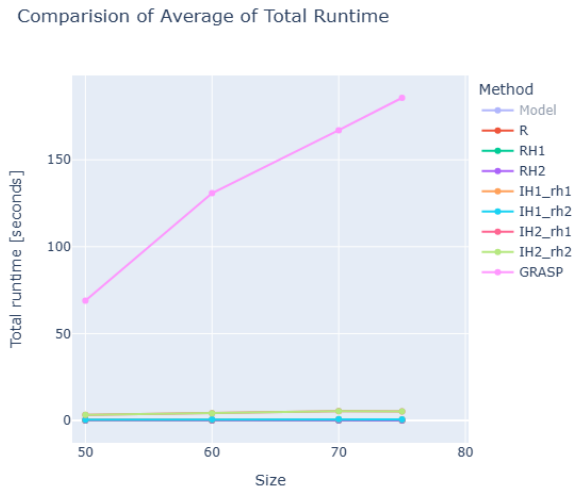
The average runtime of $IH1$ remains under 1 second, whereas $IH2$ is around 4.53 seconds across all instances. Although there are variations in their execution times according to which construction heuristic was used to build the input, these differences are not significant. In fact, the variation between the two execution cases of $IH1$ is approximately 0.83%, while for $IH2$, it is merely 0.22%.

With respect to the metaheuristic GRASP, its runtime is higher compared to the other heuristics. Specifically, it is approximately 30 times the runtime of $IH2$ and 810 times the runtime of the construction heuristics. However, despite this increase, its execution time represents only about 10.43% of that required by the optimization model.

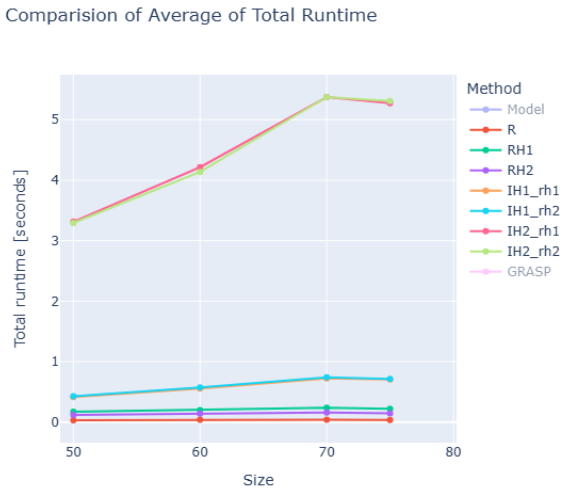
Using the execution time data of the methods, the average runtime was analyzed as a function of instance size to explore possible correlations. These results are presented next in Figure 3.18.



(a) Average runtime grouped by instance size, for all methods



(b) Average runtime grouped by instance size, for all methods except the model



(c) Average runtime grouped by instance size, for all methods except the optimization model and meta-heuristic *GRASP*

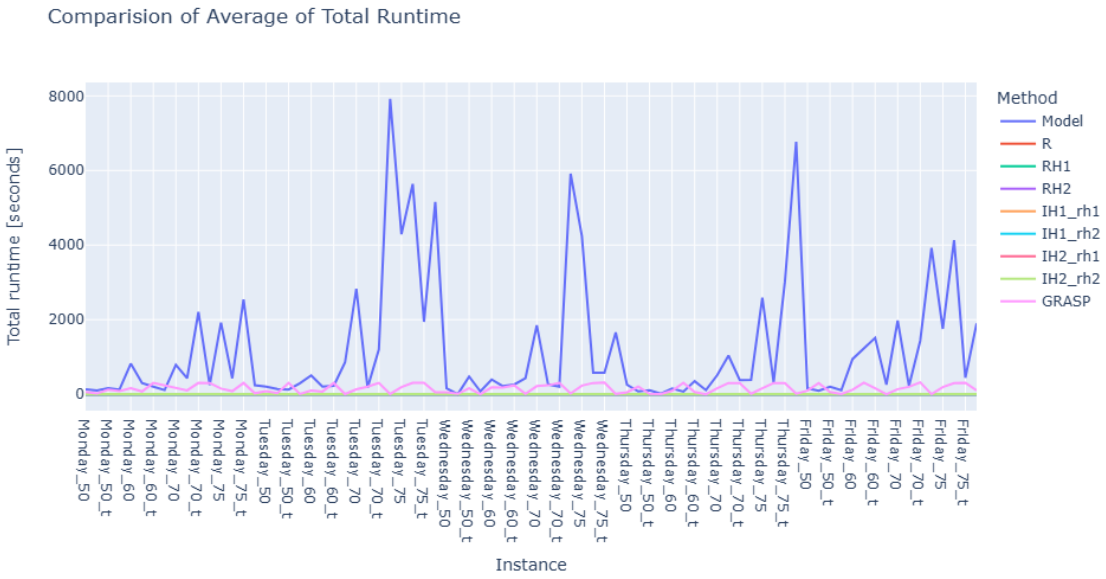
Figure 3.18: Comparison of the average runtime, obtained after executing the different methods on each instance, grouped by instance size

In Figure 3.18.a, it is evident that the optimization model exhibits significantly higher average runtime compared to the other methods. Moreover, its execution time follows an exponential growth pattern as the instance size increases.

Excluding the optimization model, Figure 3.18.b shows that the metaheuristic *GRASP* also exhibits an increasing trend in its execution time. However, its growth rate is less pronounced than that of the optimization model. Additionally, this heuristic requires more computational time on average than the others.

Referring to Figure 3.18.c, it can be observed that improvement heuristics have a longer runtime compared to construction heuristics, although within each type, both methods demonstrate similar values. Also, as expected, the random heuristic R achieves the lowest average runtime in each case. Furthermore, a general upward trend in runtime is observable as instance sizes increase from 50 to 70 tasks. However, for instances with 75 tasks, the execution time does not follow this increasing pattern.

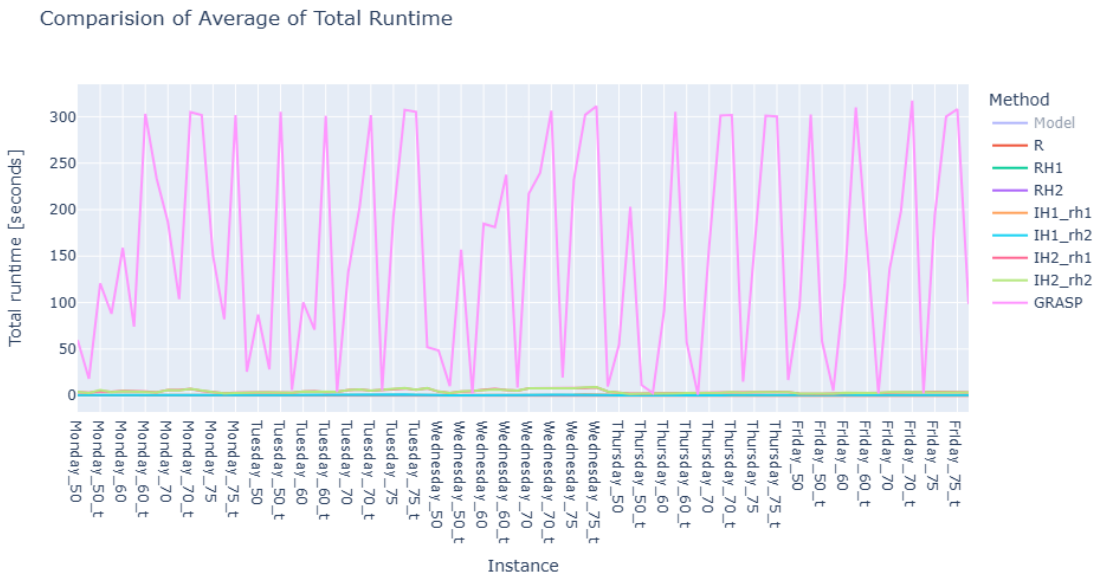
It is important to mention that the graphs and the table presented in this section summarize the runtime in aggregate terms. In the following analysis, the detailed execution time for each of the 80 instances across all methods will be presented in Figure 3.19.



(a) Runtime obtained using different methods on each instance

Figure 3.19: Comparison of the runtime obtained after executing the different methods on each instance

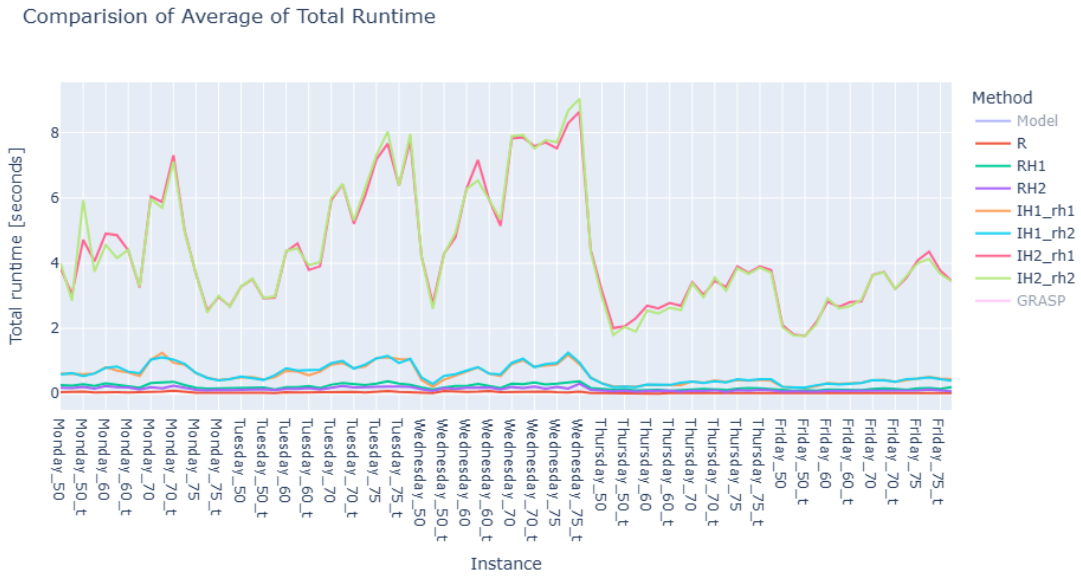
In the first place, Figure 3.19.a clearly illustrates that the optimization model has the highest runtime on average among all methods. In specific cases, its execution time is lower than that of *GRASP*, but it still remains closer to *GRASP* and consistently higher than all other heuristics.



(b) Runtime obtained using different methods on each instance, excluding the optimization model

Figure 3.19: Comparison of the runtime obtained after executing the different methods on each instance

Figure 3.19.b provides a more detailed view of the runtime behavior of the metaheuristic *GRASP*. It is possible to observe that it presents significant fluctuations, with peak execution times reaching values of around 300 times bigger than those of other heuristics, while its minimum runtime values remain close to those of the other heuristic methods.



(c) Runtime obtained using different methods on each instance, excluding the optimization model and metaheuristic

Figure 3.19: Comparison of the runtime obtained after executing the different methods on each instance

Finally, in Figure 3.19.c the relationships between the remaining heuristic methods can be observed: $IH2$ exhibits longer execution times than $IH1$, $IH1$ surpasses the construction heuristics $RH1$ and $RH2$, and both $RH1$ and $RH2$ take longer than R. Furthermore, the choice of construction heuristic used as input for the improvement heuristic does not significantly impact execution time. This is evidenced by the fact that $IH1\_rh1$ closely follows the behavior of $IH1\_rh2$, just as $IH2\_rh1$ closely aligns with $IH2\_rh2$ behavior.

3.4.4. Lateness versus runtime

In the previous sections, the analysis of the objective function and execution time metrics were addressed separately. But how are these two performance aspects related? In Figure 3.20, a summarized analysis of Lateness versus Runtime is shown, for each optimization method presented in this chapter.

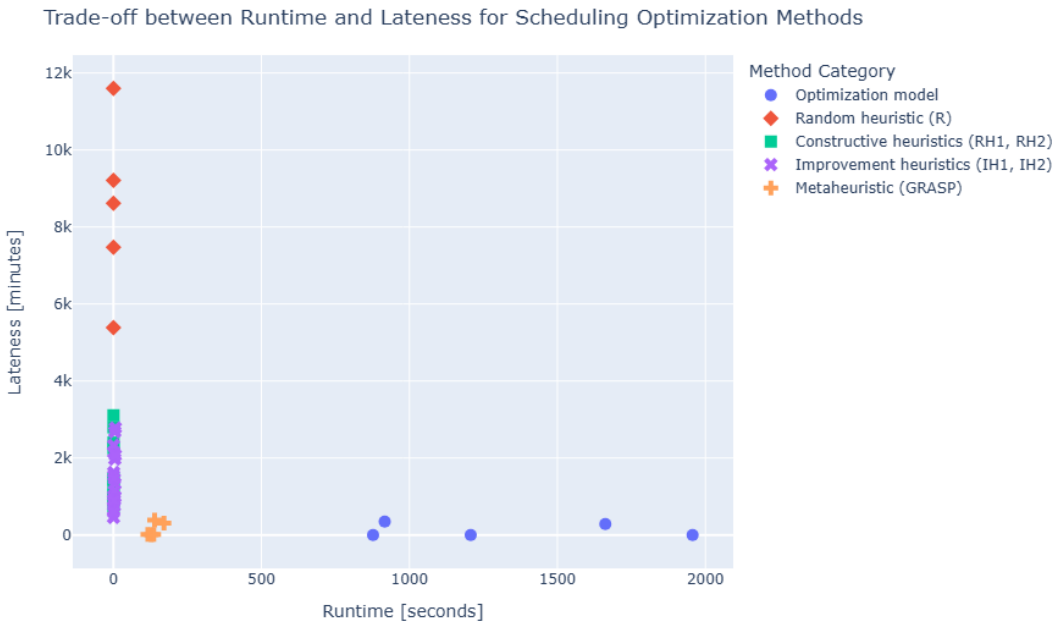


Figure 3.20: Lateness versus Runtime performance for each method

As shown in Figure 3.20, the expected relationships can be observed: while construction heuristics have the smallest execution time but achieve the worst values for the objective function, the model has the biggest execution time but reaches the optimal solution. More specifically, this analysis positions the optimization methods in a two-dimensional space, where each point represents the average lateness across instances of one weekday as a function of the corresponding average runtime.

Indeed, the heuristics exhibit the smallest execution time, in contrast to the optimization model, which has the largest. At the same time, the random assignment yields the largest value for the objective function. Additionally, it can be noted that the metaheuristic $GRASP$ corresponds to the best located point in the space: it achieves values for the objective function closer to the optimum of the model and, at the same time, it has a significantly smaller execution time, closer to the heuristics.

3.5. Conclusions of the Optimization Solution

In this chapter, an optimization model and a series of different heuristics were proposed to solve the UPMS problem presented previously. In particular, a MIP model was developed to be used as a lower referential bound, while a random construction heuristic served as an upper referential bound. Their implementation confirmed the expected behavior: the optimization model achieved the optimal solution in terms of total lateness but required the longest execution time, whereas the random heuristic produced the worst objective function value but with the lowest runtime.

Using the same input, two construction heuristics were designed and implemented. Their total lateness values averaged approximately 17.71% and 20.87% of the range between the optimal solution and the random heuristic. As expected, their results remained within both bounds, leaning towards the lower. These results show that the construction heuristics are sufficiently close to be an alternative for the optimization model, while still leaving enough room for further improvement by the other heuristics, as intended.

The two construction heuristics, $RH1$ and $RH2$, were designed with a random assignment criterion weighted by processing time, differing in their ordering criteria: $RH1$ prioritizes due dates, while $RH2$ prioritizes release times. Results showed that neither consistently outperformed the other; each performed better in different instances, in terms of lateness values. However, $RH2$ was computationally more efficient, requiring only 70% of the execution time of $RH1$. Thus, both approaches were required to build different but high-quality initial solutions.

Two improvement heuristics were then proposed, each with distinct approaches: $IH1$ evaluates swapping and non-swapping movements of tasks within the same machine sequence, whereas $IH2$ evaluates the possibility of relocating tasks from their original sequence to another in a different machine by doing an insertion. Results showed that when an initial solution with higher lateness was used, the improved solution retains this relative disadvantage. Thus, both improvement heuristics are not independent of their inputs. However, their runtime remained stable regardless of which construction heuristic was used, varying only with the characteristics of their inputs and not with the heuristics that generated them. This insight confirms the need to have the randomized component of construction heuristics, which gives variability to the initial solutions used as input for the improvement heuristics.

The metaheuristic follows a $GRASP$ approach that integrates both construction and improvement heuristics, recursively. As expected, it has the highest runtime among the heuristics; in fact, it amounts to around 30 times the runtime of $IH2$ and 810 times the runtime of the construction heuristics. Nevertheless, its execution time remains significantly lower than the runtime of the optimization model: the metaheuristic $GRASP$ requires only about 10.43% of it. In terms of the objective value, it achieved an average absolute improvement of 2968.8 percentage points over the random heuristic R, meaning an average gap of only 8.3% compared to the exact optimal value obtained by the model. This performance is consistent with its left bottom corner location in the Lateness versus Runtime space, implying that it is significantly closer to the exact value obtained by the optimization model, and at the same time, computationally efficient, allowing its use for the Machine Learning section.

All outputs were validated, confirming feasible and coherent schedules. However, some instances exhibited peaks in lateness and runtime, likely influenced by the nature of the subsets, as observed in both detailed and aggregated analyses. Additionally, the presence of idle times in Gantt charts suggests that optimizing only total lateness as the objective function might have influenced task allocation. Thus, extending the study to a multi-objective framework may be an alternative for future work.

Regarding runtime scalability, all methods exhibit exponential growth as instance size increased, but at different rates: the optimization model ranged from around 200 seconds to almost 3000, $GRASP$ from around 70 seconds to almost 200, and the other heuristics vary by less than 2 seconds. This behavior aligns with expectations, as the nature of the problem makes the optimization model NP-hard, leading to an exponential growth in their execution time. Conversely, the construction heuristics maintained low runtimes thanks to their simplicity and efficient design.

In conclusion, heuristic-based approaches effectively balance solution quality and runtime, making them suitable for practical use. Moreover, the proposed metaheuristic provides near-optimal results with significantly lower execution times, validating its applicability in the next stage of this thesis.

Chapter 4

Machine Learning Solution

Optimization models can achieve exact solutions, but the runtime required grows exponentially as problem complexity increases. Heuristics provide faster approximations but may lead to sub-optimal solutions. However, there is no method that balances quality and efficiency enough to enable its usage in real-time responsive applications.

This chapter introduces a Machine Learning approach with the aim of studying the possibility to predict the results of an optimization model, obtaining outputs in significantly shorter execution times with an acceptable loss of solution quality. For that purpose, a hybrid approach combining classification and regression models is proposed, as illustrated in Figure 4.1 and explained in the following sections.

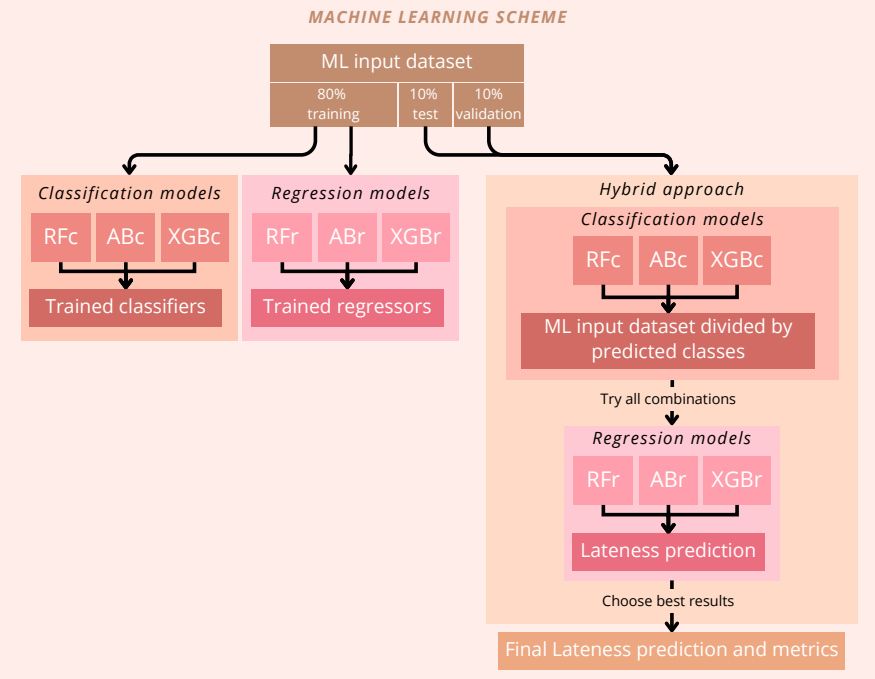


Figure 4.1: Diagram of the proposed Machine Learning solution

4.1. Input Dataset

The input for the Machine Learning section was built with the results obtained after executing the metaheuristic iteratively. In the next subsections, a detailed description of the process to obtain the dataset will be presented.

4.1.1. Rows of the Dataset: Experiments

As shown in Figure 4.2, the process starts by applying a specific experiment that introduces a controlled modification to the original data. Then, the metaheuristic was applied to that modified dataset, and the output obtained was used to construct one row of the new dataset. Then, these steps were applied iteratively to create sufficient rows to use the new dataset as input for the Machine Learning models.

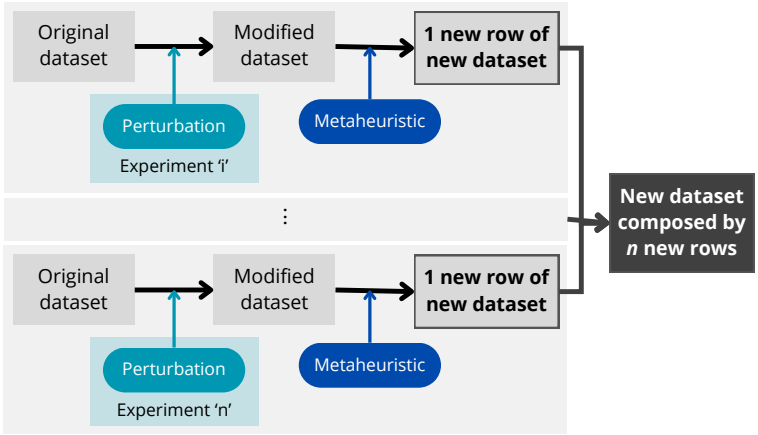


Figure 4.2: Diagram of the dataset creation process for the ML stage

The experiments that defined the perturbations of the original dataset are described in Table 4.1.

Experiment	Perturbation applied to the original dataset	Rows generated
0	No perturbation; it uses the original dataset	80
1	Removal of machines 4 and 10 from the concrete plant	80
2	Removal of machines 4, 7 and 10 from the concrete plant	80
3	Decrease of 10 minutes in the spacing time of each task	80
4	Randomly removing one machine from the feasible set of each task	80
5	Randomly removing two machines from the feasible set of each task	79
6	Increase of 15 minutes in the due date of the tasks belonging to the dispatch train with the largest lateness	27
11	Increase of 5 minutes in the due date of the tasks belonging to the dispatch train with the largest lateness	24
12	Increase of 10 minutes in the due date of the tasks belonging to the dispatch train with the largest lateness	24
13	Increase of 20 minutes in the due date of the tasks belonging to the dispatch train with the largest lateness	27

Continued on the next page

Experiment	Perturbation applied to the original dataset	Rows generated
14	Increase of 15 minutes in the due date of the tasks belonging to the two dispatch trains with the largest lateness	27
15	Increase of 5 minutes in the due date of the tasks belonging to the two dispatch trains with the largest lateness	24
16	Increase of 10 minutes in the due date of the tasks belonging to the two dispatch trains with the largest lateness	24
17	Increase of 20 minutes in the due date of the tasks belonging to the two dispatch trains with the largest lateness	24
18	Decrease of 20 minutes in the spacing time of each task	80
19	Removal of machines 4 and 7 from the concrete plant	80
20	Removal of machines 7 and 10 from the concrete plant	80
3 modified	Decrease of 5 minutes in the spacing time of each task	6
	Decrease of 8 minutes in the spacing time of each task	6
	Decrease of 8 minutes in the spacing time of each task	6
4 modified	Randomly removing one machine from the feasible set of each task, with another random seed	33
5 modified	Randomly removing two machines from the feasible set of each task, with another random seed	34
14 modified	Decrease of 15 minutes in the due date of the tasks belonging to the two dispatch trains with the largest lateness	17
15 modified	Decrease of 5 minutes in the due date of the tasks belonging to the two dispatch trains with the largest lateness	13
16 modified	Decrease of 10 minutes in the due date of the tasks belonging to the two dispatch trains with the largest lateness	13
17 modified	Decrease of 20 minutes in the due date of the tasks belonging to the two dispatch trains with the largest lateness	13
18 modified	Decrease of 15 minutes in the spacing time of each task	6
	Decrease of 18 minutes in the spacing time of each task	6
	Decrease of 22 minutes in the spacing time of each task	6
Total number of rows composing the ML input dataset		*1079*

Table 4.1: Experiments used to create rows of the ML input dataset

4.1.2. Columns of the Dataset: Features

The dataset has 29 columns, where the first specifies the instance (dataset modified with the experiment) and the others correspond to metrics designed to serve as features for the Machine Learning model. In particular, there are macro, meso, and micro level metrics, all described in Table 4.2.

Feature	Type	Description
`trains_quantity`	Integer	Number of dispatch trains in the instance
`tasks_quantity`	Integer	Number of tasks in the instance
`Dist_tasks`	Continuous	`tasks_quantity` / $(dd_{max} - dd_{min})$
`Day`	Categorical	Due date day of tasks in the instance
`Size`	Categorical	Size of the instance (total number of tasks)
`Subset_range`	Categorical	Time window corresponding to the due date of tasks in the instance

Continued on the next page

Feature	Type	Description
Machines_quantity	Integer	Number of machines working in the plant
Mean_spacing	Continuous	Average spacing time of all tasks in the instance
Min_spacing	Integer	Minimum spacing time of all tasks in the instance
Mean_tproc	Continuous	Average processing time of all tasks in the instance
dd_first_task	Integer	Earliest due date of the instance, corresponding to the first task
dd_last_task	Integer	Latest due date of the instance, corresponding to the last task
tasks_quantity_morning	Integer	Number of tasks with due dates in the 'Morning' window
tasks_quantity_afternoon	Integer	Number of tasks with due dates in the 'Afternoon' window
tasks_quantity_last	Integer	Number of tasks with due dates in the 'Last' window
Disp_med_orders	Continuous	Median of $(dd_{i+1} - dd_i)$, for all pair of consecutive tasks in the instance
Disp_prom_orders	Integer	Average of $(dd_{i+1} - dd_i)$, for all pair of consecutive tasks in the instance
Mean_dispatch_train_length	Float	Average length of all dispatch trains belonging to the instance
Max_dispatch_train_length	Integer	Maximum length of all dispatch trains belonging to the instance
Min_dispatch_train_length	Integer	Minimum length of all dispatch trains belonging to the instance
Max_spacing	Integer	Maximum spacing time of all tasks in the instance
Max_tproc	Float	Maximum processing time of all tasks in the instance
Min_tproc	Float	Minimum processing time of all tasks in the instance
trains_quantity_1_tasks	Integer	Number of dispatch trains in the instance composed of a single task
trains_quantity_2_5_tasks	Integer	Number of dispatch trains in the instance composed of two to five tasks
trains_quantity_6_9_tasks	Integer	Number of dispatch trains in the instance composed of six to nine tasks
trains_quantity_10_14_tasks	Integer	Number of dispatch trains in the instance composed of ten to fourteen tasks
trains_quantity_15_o_mas_tasks	Integer	Number of dispatch trains in the instance composed of fifteen or more tasks

Table 4.2: Features included in the ML input dataset

4.2. Machine Learning Model

This section details the Machine Learning hybrid approach developed in this thesis, which integrates classifiers and regressors.

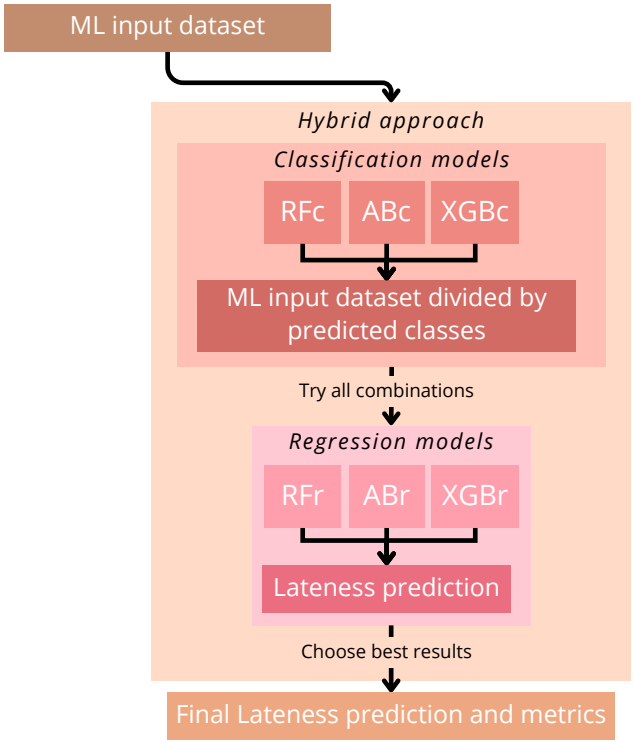


Figure 4.3: Flow diagram of the ML hybrid approach

4.2.1. Classification and Regression Models

Three different models were used: Random Forest (RF), Adaboost (AB), and XGboost (XGB), all of them applied both as classifiers and regressors.

To execute these models in their classifier form, three classes were defined using the ML input dataset. These were used to label the data and then predict the corresponding class. Thus, the output in each case corresponds to the predicted class labels for each instance.

With this information, it was possible to subdivide the ML input dataset into three parts, each one corresponding to a predicted class. The subset corresponding to class 1 was used as input for the regressors, whose output is the prediction of the total lateness.

It is worth mentioning that every possible classifier-regressor combination was executed to construct the output. Additionally, the regressors were also implemented using a weighted (pondered) result of classifiers, defined as 'POND' with the following rule: if two or more classifiers predict the same class x, then the predicted class of the element is x.

4.2.2. Building the Output

The output of the Machine Learning model is the predicted total lateness, corresponding to the objective value of the optimization section. To construct this, as shown in Figure 4.4, three different flows were followed, depending on which class was predicted by the classifiers:

- Class 0: if an element was predicted as class 0, the total lateness was set in the output as 0.
- Class 1: if an element was predicted as class 1, the total lateness was set in the output with the value obtained by the regressor.
- Class 2: if an element was predicted as class 2, the total lateness was set in the output as 300.

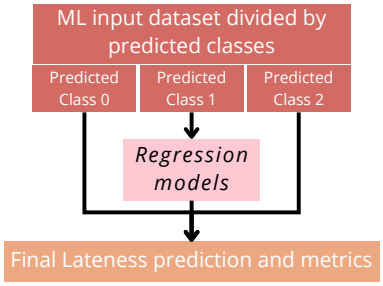


Figure 4.4: Diagram of the ML output construction

Additionally, if an element was misclassified, the error was calculated using the following rules:

⟶ If an element of real class 0 or 1 was predicted as class 2, the total lateness was set to 300.

⟶ If an element of real class 2 was predicted as class 0 or 1, the 'real' total lateness was set to 300.

With these completed output values, it was possible to evaluate the performance metrics of the proposed approach, computing lateness as shown in Table 4.3.

Table 4.3: Values for total lateness in ML output

	Predicted classes			
Real classes		Class 0	Class 1	Class 2
	Class 0	L_{pred}=0	$L_{pred} = L_{pred}$	$L_{pred} = 300$
	Class 1	L_{pred}=0	$L_{pred} = L_{pred}$	$L_{pred} = 300$
	Class 2	$L_{real} = 300$	$L_{real} = 300$	$L_{pred} = 300$

4.3. Implementation

4.3.1. Data Processing

Data Splitting

The complete ML input dataset was divided into training, test, and validation subsets, with the proportions shown in Table 4.4.

Table 4.4: Division of ML input dataset into training, test, and validation subsets

Subset	Splitting percentage	Quantity of rows
Train	80	863
Test	10	108
Validation	10	108

Definition of Classes and Balanced Data

The ML input dataset was divided into three classes:

- Class 0: tasks with total lateness equal to 0.
- Class 1: tasks with total lateness different from 0, but within an 'acceptable' range.
- Class 2: tasks with 'very large' lateness.

Given the context of the problem, a cut-off point equal to 300 minutes (5 hours) was defined. This value was used to define the group of tasks with 'very large' lateness. In other words, this threshold separated data with lateness different from 0, into classes 1 and 2. It is also worth mentioning that a version of the dataset with balanced classes was generated through random selection when fitting the classification models. The number of rows is specified in Table 4.5.

Table 4.5: Number of rows per class in the original (unbalanced) and balanced datasets

Class	Unbalanced dataset	Balanced dataset
0	484	230
1	365	230
2	230	230

Uses of Balanced and Unbalanced Data

Balanced data was used exclusively for training classifiers, as mentioned in Section 4.3.1, whereas unbalanced data was used in all other cases: training regressors and evaluating the entire method with the test and validation datasets, as shown in Table 4.6.

Table 4.6: Use of balanced and unbalanced datasets during training and evaluation

Dataset	Balanced data	Unbalanced data
Train	Classifiers	Regressors
Test	-	Classifiers and regressors
Validation	-	Classifiers and regressors

4.3.2. Model Fitting

The classification models were defined and trained using the balanced ML input dataset, while the regressors used the unbalanced ML input dataset filtered by class 1. In both cases, feature and parameter selection was performed beforehand to ensure the models were fitted using the best values between the given alternatives.

Features Selection

The Random Feature Elimination with Cross-Validation (RFECV) method was applied to select a subset from the possible features (columns of ML input dataset) for each model. To this end, a recursive search was performed, executing RFECV for each possible value of $kfold$ in the set $\{5, 8, 12, 15\}$. The optimization criterion was accuracy for the classifiers and root mean squared error for the regressors. The results are summarized in Table 4.7 presented next, while a detailed version with the selected features and metrics associated is available in Annex C.

Feature	**RFC**	**ABC**	**XGBC**	**RFR**	**ABR**	**XGBR**
trains_quantity	✓	✓	✓	✓	✓	✓
tasks_quantity	✓	✓	✓	✓	✓	✓
Dist_tasks	✓	✓	✓	✓	✓	✓
Day			✓			
Size	✓	✓	✓	✓		
Subset_range			✓			
Machines_quantity	✓	✓	✓	✓	✓	✓
Mean_spacing	✓	✓	✓	✓	✓	
Min_spacing	✓	✓	✓	✓	✓	
Mean_tproc	✓	✓	✓	✓	✓	
dd_first_task	✓	✓	✓			
dd_last_task	✓	✓	✓	✓	✓	✓
tasks_quantity_morning			✓			
tasks_quantity_afternoon	✓	✓	✓	✓	✓	✓
tasks_quantity_last	✓		✓	✓	✓	
Disp_med_orders		✓	✓			
Disp_prom_orders	✓	✓	✓	✓	✓	
Mean_dispatch_train_length	✓	✓	✓	✓	✓	
Max_dispatch_train_length	✓	✓	✓			
Min_dispatch_train_length			✓			
Max_spacing		✓	✓			
Max_tproc	✓	✓	✓	✓	✓	
Min_tproc	✓	✓	✓	✓	✓	

Feature	RFC	ABC	XGBC	RFR	ABR	XGBR
trains_quantity_1_tasks	✓		✓	✓	✓	✓
trains_quantity_2_5_tasks	✓	✓	✓	✓	✓	
trains_quantity_6_9_tasks		✓	✓			
trains_quantity_10_14_tasks		✓	✓	✓		✓

Table 4.7: Variables selected using Random Feature Elimination with Cross-Validation (RFECV) for the Machine Learning model

Parameter Tuning

For each classifier and regressor, a series of recursive searches was performed to evaluate the model for every possible value of a predefined parameter grid, shown in Table 4.8.

Table 4.8: Parameter tuning grid for each ML model

Model	Parameters
Random Forest (classifier)	{'n_estimators': [10,50,100,200], 'max_depth': [1,2,3,4,5,None], 'min_samples_split': [2,5,10], 'min_samples_leaf': [1,2,5,10], 'criterion': [gini, entropy, log_loss]}
Random Forest (regressor)	{'n_estimators': [10,50,100,200], 'max_depth': [1,2,3,4,5,None], 'min_samples_split': [2,5,10], 'min_samples_leaf': [1,2,5,10]}
Adaboost (classifier)	{'n_estimators': [10,50,100,200], 'learning_rate': [0.01,0.1,0.3,1]}
Adaboost (regressor)	{'n_estimators': [10,50,100,200], 'learning_rate': [0.01,0.1,0.3,1], 'loss': [linear,square,exponential]}
XGboost (classifier)	{'booster': [gbtree,gblinear,dart], 'max_depth': [1,2,3,4,5,None], 'learning_rate': [0.01,0.1,0.3,1]}, 'tree_method': [auto,exact,approx,hist]}
XGboost (regressor)	{'booster': [gbtree,gblinear,dart], 'max_depth': [1,2,3,4,5,None], 'learning_rate': [0.01,0.1,0.3,1]}, 'tree_method': [auto,exact,approx,hist]}

Additionally, inside the recursion, each model was trained and evaluated using Cross-Validation. Then, parameters were finally selected by optimizing F1-score for classifiers and root mean squared error for the regressors. The results are detailed in Table 4.9.

Model	Parameters
Random Forest (classifier)	{'n_estimators': 200, 'max_depth': None, 'min_samples_split': 5, 'min_samples_leaf': 1, 'criterion': gini}
Random Forest (regressor)	{'n_estimators': 50, 'max_depth': None, 'min_samples_split': 2, 'min_samples_leaf': 1}
Adaboost (classifier)	{'n_estimators': 10, 'learning_rate': 1}
Adaboost (regressor)	{'n_estimators': 100, 'learning_rate': 1, 'loss': exponential}
XGboost (classifier)	{'booster': gbtree, 'max_depth': None, 'learning_rate': 0.3}, 'tree_method': hist}
XGboost (regressor)	{'booster': gblinear, 'max_depth': 2, 'learning_rate': 0.01}, 'tree_method': approx}

Table 4.9: Parameter tuning resulting values for each part that integrates Machine Learning model

In addition to the grid search and cross-validation results reported in this section, a detailed sensitivity analysis of the parameter tuning process for each ensemble method is

provided in Annex D. This Annex presents the effect of varying key hyperparameters on model performance, including accuracy and robustness across different training subsets. In particular, sensitivity plots and tables are included for XGBoost, AdaBoost, and Random Forest classifiers and regressors, illustrating how parameter adjustments influence predictive stability. These results not only validate the robustness of the selected configurations but also provide transparency regarding the trade-offs considered when defining the final tuned models.

4.3.3. Test and Validation

The proposed hybrid approach was executed using the test and validation subsets of the unbalanced data separately, to then measure its performance. Each model was implemented to predict the classes or total lateness values, as previously defined and fitted.

4.4. Results

4.4.1. Performance Metrics of the Predictions

Performance metrics were calculated using the real total lateness values of each row in the input ML dataset with the predicted values from every classifier-regressor combination, following the algorithm explained in Section 4.2. The regression metrics Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and the coefficient of determination (R2) were chosen.

Table 4.10 reports the average regression performance metrics for every possible classifier-regressor combination applied to both the test and validation subsets of data.

Table 4.10: Performance metrics for each Regressor–Classifier combination

Regressor_Classifier	Test metrics				Validation metrics			
	RMSE	MAE	MAPE	R2	RMSE	MAE	MAPE	R2
RFr_RFc	68.04	22.67	14.78%	99.58%	56.10	18.64	16.68%	99.56%
RFr_ABc	51.28	14.95	9.66%	99.76%	54.62	17.76	16.17%	99.58%
RFr_XGBc	57.43	17.20	12.82%	99.70%	46.30	15.84	16.24%	99.70%
RFr_PONDc	49.42	13.89	10.13%	99.80%	40.53	13.03	14.27%	99.77%
ABr_RFc	72.66	27.98	20.95%	99.52%	57.06	22.02	20.11%	99.54%
ABr_ABc	53.81	19.69	16.26%	99.74%	56.99	21.55	19.57%	99.55%
ABr_XGBc	61.16	22.76	18.85%	99.66%	49.24	19.88	19.63%	99.60%
ABr_PONDc	51.98	18.41	16.34%	99.76%	43.52	16.73	17.63%	99.73%
XGBr_RFc	79.42	37.17	26.44%	99.43%	67.37	31.41	24.31%	99.36%
XGBr_ABc	65.90	30.54	21.72%	99.61%	65.29	28.93	22.88%	99.40%
XGBr_XGBc	73.15	34.17	24.61%	99.52%	61.41	28.61	23.57%	99.47%
XGBr_PONDc	65.33	29.86	22.10%	99.61%	54.27	24.77	21.53%	99.59%

It is possible to notice from Table 4.10 that MAE values remain between approximately 13 and 37 minutes, while MAPE ranges between 10% and 26%. At the same time, R2 remains close to 100% in all the cases, across the different models (combinations of Regressor_Classifier) and subsets of data (test and validation). As for the RMSE metric, it ranges between approximately 40 and 80 minutes, which implies that the method has an average

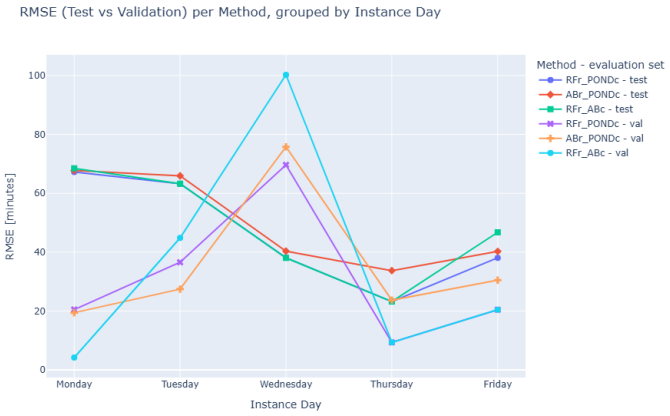
error that goes from 40 minutes to less than one and a half hours. For all mentioned metrics, there are some cases where the values increased from the test to the validation set; however, the difference are not large enough to be a sign of overfitting. Referring to the combinations of models, all achieved acceptable results, with RFr_PONDc (Random Forest regressor with Pondered classifier) performing best, closely followed by ABr_PONDc (Adaboost regressor with Pondered classifier) and RFr_ABc (Random Forest regressor with Adaboost classifier).

It is worth noting that the values in the previous analysis are calculated as averages for all the instances in the test and validation subsets of the ML input data. To explore the results at a deeper level, an analysis of average results grouping the instances by 'Day' and 'Size' is conducted.

Table 4.11: Performance metrics for each Regressor–Classifier combination, grouping instances by 'Day' and 'Size'

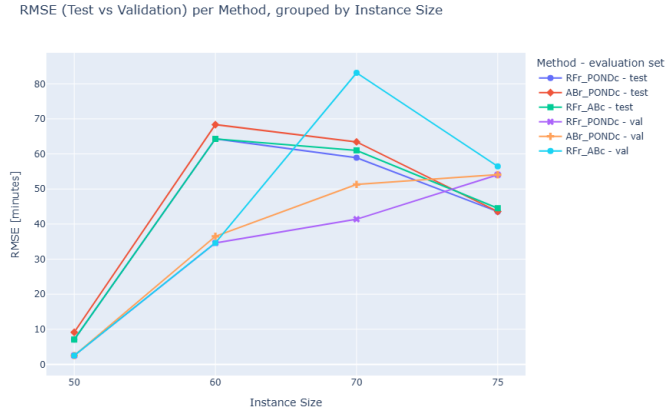
Regressor_Classifier	Instances grouped by 'Day'				Instances grouped by 'Size'			
	RMSE test	R2 test	RMSE val	R2 val	RMSE test	R2 test	RMSE val	R2 val
RFr_RFc	61.50	91.06%	41.03	98.52%	61.09	86.83%	41.64	99.37%
RFr_ABc	47.94	96.17%	35.81	99.03%	44.97	87.53%	44.18	99.40%
RFr_XGBc	53.03	96.57%	37.65	98.35%	49.02	87.50%	39.30	99.00%
RFr_PONDc	45.97	96.66%	31.29	99.31%	43.46	87.53%	33.13	99.49%
ABr_RFc	67.26	90.41%	45.50	98.39%	65.99	85.07%	43.17	99.29%
ABr_ABc	51.61	95.61%	40.89	98.88%	47.62	85.55%	46.09	99.33%
ABr_XGBc	58.31	95.41%	41.62	97.86%	52.86	85.85%	41.53	98.97%
ABr_PONDc	49.59	96.08%	35.36	99.17%	46.12	85.90%	36.09	99.43%
XGBr_RFc	75.29	87.76%	58.30	96.94%	71.38	77.56%	55.83	97.86%
XGBr_ABc	63.36	93.54%	53.20	97.45%	60.48	77.26%	56.19	98.05%
XGBr_XGBc	70.67	93.12%	54.17	96.09%	65.62	77.98%	52.74	97.72%
XGBr_PONDc	62.22	94.02%	47.98	97.76%	59.95	78.03%	46.92	98.15%

From Table 4.11, it can be observed that the overall behavior of the methods is consistent with the results obtained for the complete dataset input, achieving a good performance. In fact, the RMSE values in all cases (across all methods and with test and validation data) decreased compared to the case when the metrics were calculated without grouping the instances (Table 4.10). Referring to R2, it remains near to 100% in most of the cases, except for the ones that use XGboost regressor, where slightly lower performance is observed. Another important finding is that R2 values, calculated for instances grouped by size in the test set, are also lower, being around 77% for the ones using XGboost regressor and around 85-87% for the other methods. As for the Regressor_Classifier combinations, RFr_PONDc keeps being the one with best performance, followed by RFr_ABc and ABr_PONDc. Figure 4.5 presents RMSE values (for both test and validation sets) corresponding to these three top-performing combinations.



(a) Instances group by 'Day'

Figure 4.5: Average RMSE calculated using test and validation sets



(b) Instances group by 'Size'

Figure 4.5: Average RMSE calculated using test and validation sets

Figure 4.5.a shows that the three combinations of Regressor_Classifier have the same behavior: the validation RMSE is lower than in the test dataset in all cases, except for instances of Wednesday. Among them, RFr_PONDc is the one with the best performance in terms of magnitude of RMSE for all instances, of both sets of data (test and validation), which is consistent with the results shown in Table 4.10 and 4.11. From the chart is also possible to notice that RFr_ABc in the validation data achieves both the lowest and highest RMSE values for Monday, Tuesday, and Wednesday instances.

In Figure 4.5.b, the same trends of Figure 4.5.a are present: it shows a similar general performance of all methods for the test dataset, in addition to the fact that RFr_PONDc is the best-performing approach. With respect to the validation set, all methods have the same behavior for most of the cases, excluding instances of size 70. In that case, RFr_ABc reaches the largest RMSE.

4.4.2. Execution Time

The total execution time of the proposed Machine Learning method is presented in Table 4.12, along with the average execution time per instance (i.e. per row of data) and per set (test or validation data, each consisting of 108 rows).

Table 4.12: Runtime of proposed Machine Learning approach, in seconds

Total runtime	Average runtime per instance	Average runtime per set
13.9824 [s]	0.0647 [s]	6.9912 [s]

Table 4.12 shows a small and thus acceptable execution time for the proposed ML method, compared with the state-of-the-art applications.

4.4.3. Comparison with Optimization Models

To evaluate the performance of the proposed Machine Learning method, a comparative analysis against the optimization model and heuristics was conducted. Thus, in Table 4.13, the average value for total lateness and runtime were computed for: the optimization model (upper bound reference), random assignment R (lower bound reference), metaheuristic $GRASP$, and the Machine Learning model. Additionally, the average lateness of the input was also included as a reference. It is important to mention that the average Lateness for the optimization methods were computed using the 80 original instances explained in section 3.1.2, whereas for the Machine Learning model the instances with and without perturbations were considered (the test and validation subsets of the ML input dataset). Thus, the average lateness for the ML input column was computed using the real data, while for the ML model column, it was computed using the predictions.

Table 4.13: Performance of ML model compared to optimization and heuristic methods

	Optimization model	R	$GRASP$	Input ML	ML model
Lateness	127.9718	8459.6937	147.4843	367.5949	371.0771
Runtime	1323.8559	0.0354	138.0988	-	0.0647

From Table 4.13, it is possible to observe that ML model predictions have a gap of only 0.94% on average with the original data values presented in the input. In terms of solution quality, the table shows that the average of the ML model predictions is almost three times (2.89) the value of the optimization model, while the metaheuristic GRASP is 1.15 times and the random heuristic is 66.10 times its value.

Regarding runtime, the ML model is the second fastest approach, closely following the random heuristic R: in both cases, it takes less that a second on average to execute an instance. Compared to the other optimization methods, the ML model takes only 0.0468% of the execution time required by the metaheuristic $GRASP$ and 0.0048% of the time required by the optimization model.

4.5. Conclusions of the Machine Learning Solution

In this chapter, a Machine Learning approach was proposed. The method takes as input a dataset constructed using optimization solutions, while its output corresponds to predicted total lateness. Essentially, the algorithm consists of two stages: first, it uses classifiers to predict lateness classes, defined as zero (class 0), acceptable (class 1), and very large (class 2). Then, regressors are applied only to data predicted as class 1. Final predictions are constructed by all possible regressor–classifier combinations.

Since the final predictions of the method are continuous total lateness values, regression metrics were calculated. On average, the method achieved low errors across all regressor–classifier combinations and datasets (test and validation): MAE ranged from 13 to 37 minutes, RMSE from 40 to 80 minutes (which shows that on average the error is around half an hour and with a deviation of less than one hour and a half), and MAPE from 10% to 26%. R2 remained consistently close to 100%. All of these values show a good performance for the context of the problem, in terms of their error magnitude and explanatory power.

It is important to mention that the differences in metrics between test and validation datasets are not a sign of overfitting, due to their magnitude (neither underfitting). Precisely, there are some cases in which MAPE values increased or decreased between training and test data. This is not necessarily a problem, as it is expected given the asymmetry of the metric and its sensitivity to zeros in the real values.

Among the regressor–classifier combinations, three achieved the best overall performance: RFr_PONDc (Random Forest regressor with Pondered classifier), ABr_PONDc (Adaboost regressor with Pondered classifier), and RFr_ABc (Random Forest regressor with Adaboost classifier). Specifically, RFr_PONDc is the best one, achieving more than 99% of R2 and errors near 45 minutes, in all different analysis cases. Thus, if a single combination had to be selected, the recommendation would be to apply the Random Forest regressor to the data divided by the predicted classes obtained using the POND classifier.

The robustness of the ML method is confirmed by additional analyses grouping instances by 'Day' and 'Size', which revealed the same trends as the global averages. This suggests that good performance is not merely an artifact of averaging, but rather a consistent property across different subsets of the problem.

About execution time, the method also achieves low values, compared to state-of-the-art revision: it takes less than one second on average (indeed, less than 0.1 seconds) to compute the prediction of total lateness for one instance. Thus, as expected, it is suitable for applications that need real-time updates.

Regarding the Machine Learning model performance, compared to the optimization methods, the average of the total lateness increased, when compared to the metaheuristic $GRASP$, passing from 1.15 times the value of the optimization model to 2.89 times. However, this value of the objective function remains far from the one obtained by the random heuristic R (which is around 66 times the value of the optimization model), meaning that it is still a close approximation to the optimum. Additionally, its predicted values have a gap of only

0.94% on average with real data of test and validation subsets, achieving a good prediction of real values.

On the other hand, the ML model runtime performance is the second best, after the random assignment R, requiring only 0.0468% of the execution time required by the meta-heuristic $GRASP$ and 0.0048% of the time required by the optimization model, showing the computational efficiency expected for the method and demonstrating its ability not only to predict solutions but also to achieve a balance between quality and efficiency.

In conclusion, the proposed ML method effectively predicts the objective function of an optimization model, achieving good quality solutions in a short execution time.

Chapter 5

Discussion and Conclusions

The aim of this chapter is to synthesize the main findings from both approaches presented in this thesis, Optimization and Machine Learning, highlighting their individual and shared contributions. Additionally, general conclusions are drawn and directions for future research are outlined.

5.1. Discussion

The purpose of the first part of this thesis was to create solutions using optimization methods for solving the problem, which were sufficiently efficient and close to the optimal, to be used as input for the Machine Learning model. This was achieved, as the proposed metaheuristic obtained results with only 8.3% of average gap with the exact value obtained by the optimization model, requiring only about 10,43% of its execution time. This makes high-quality solutions, considering that the problem is NP-hard with exponential runtimes. Likewise, the computational efficiency of the proposed metaheuristic is given by its GRASP design: it applies a search in a reduced space, building initial feasible schedules and then exploring focused improvement opportunities, which brings the objective function values close to the reference bound. Although this method could be further enhanced in terms of solution quality and runtime, such refinements were beyond the scope of this research. For the purposes of this thesis, the achieved results are good enough to be used for the second stage.

Referring to the ML section, the focus was on studying the possibility of obtaining predictions of the optimization solutions, using the results given by the metaheuristic implementation. This was accomplished using a hybrid approach: first, a classification problem is solved, and then it uses the predicted classes as input for a regression problem. The proposed method succeeds in predicting the total lateness (the objective function of the optimization problem) with errors that average around 30 minutes for MAE and R2 close to 100%. These are good performance values, considering the state of the art and the context of the problem: predicting the total sum of lateness of an entire schedule of orders with only 30 minutes of error. Furthermore, the high-quality solutions are accompanied by low execution time: it takes less than 0.1 seconds on average to compute the total lateness prediction for one instance. This good performance is coherent with the design of the proposed method: the regressors are applied only to class 1, given by the classifiers, which reduces the total runtime of the method.

It is worth noting that both the optimization and Machine Learning methods use approaches that combine different models. While this could have increased the overall error in principle, this was not the case: the error of each algorithm was individually controlled, so the improvement of the solutions was greater than the total error in the integrated form.

As discussed in previous chapters, due to computational constraints, it was not possible to execute the optimization model and heuristics using the entire original dataset. Instead, subsets were used as instances. It could be interesting to try other subsets with a larger size to compare those results with the ones obtained by the model. This could also enable the design of more features to train the ML model. However, the various instances constructed were sufficient to, for example, analyze the exponential growth that characterizes this NP-hard problem and to observe the efficient performance of the final method. Specifically, while solving some large instances with the optimization model was impossible, the metaheuristic was able to compute solutions, albeit with long execution times. Moreover, the Machine Learning model was able to compute solutions with a very small runtime for these large instances. This highlights an advantage of the ML approach: while metaheuristics could still be improved in quality, their runtime remains too high for real-time applications. In contrast, the ML approach offers high quality solution in a significantly lower time, enabling practical use in industrial applications.

Although the ML model produced a higher average total lateness than the metaheuristic $GRASP$, this cannot be attributed to the ML model itself, because its predictions have an average gap of just 0.94% with the real input data used by the ML model. Thus, there is a possibility that the quality of its solutions could increase by improving the input dataset; for example, designing a dataset with perturbations of the instances that keep the average total lateness of its rows closer to the optimal value before applying the ML model.

One potential industrial application of this work is a decision-support tool for daily operations management of a concrete plant. Specifically, plant managers could rely on the metaheuristic and ML surrogate to rapidly evaluate the impact of incoming orders, schedule adjustments, or disruptions, enabling near real-time rescheduling decisions without depending on the computationally expensive MILP solver. This type of tools often requires overcoming several implementation challenges, such as procuring quality data in a timely manner and integration with existing production planning systems. Addressing these challenges is necessary for transitioning the proposed methodology from a research prototype to a fully operational industrial tool.

5.2. General Conclusions

When solving the problem in the first section, as expected, the optimization model achieves the best solution for the objective function, while the random heuristic achieves the lowest execution time. Excluding these extremes, construction heuristics were the ones with the best execution time, while the metaheuristic achieved the best quality solution, in terms of distance to the optimum value achieved by the optimization model. In fact, the results demonstrated that the proposed metaheuristic provides near-optimal results with significantly lower execution times, validating its use for the Machine Learning stage.

In the Machine Learning stage, a proposed method was effectively implemented. Although all the classifier-regressor combinations had good performance, applying the Random Forest regressor to data divided by the predicted classes obtained using the POND classifier was the best one. In fact, its predictions achieve an average value for the total lateness of 2.89 times the one obtained by the optimization model, but only requiring 0.0048% of its execution time.

In conclusion, this thesis demonstrated the feasibility of training and implementing a Machine Learning model to predict the objective function of a Mixed-Integer Linear Programming optimization model, achieving good quality solutions within a short execution time. This enables its use in applications that require real-time updates, and in particular, in the concrete industry.

5.3. Future Work

As this thesis represents a first approach to study the possibility of predicting solutions of an optimization model using Machine Learning tools, future research could explore the use of more advanced ML models to obtain better results, such as neural networks.

Additionally, several variations of the problem could be studied to enhance the applicability of the proposed methodology to the industry. For example:

- Multi-objective optimization: rather than only considering total lateness as the objective function of the problem, future models could also incorporate other variables. For example, minimizing total lateness and idle times could maximize the service level for the client and at the same time the efficiency of the concrete plant.
- Limited number of trucks: in real-world scenarios, trucks are also a limited resource. Thus, there could be more than one bottleneck source, given by the availability of either machines or trucks.
- Heterogeneous machines: to extend the model, it is enough to define groups of machine types in the formulation of the optimization problem. For the ML part, the type of the machine could be a feature to be defined and used.
- Weights: Assigning weights to specific orders, reflecting their relative importance, and incorporating these as features in both optimization and ML models.
- Stochastic variations: taking into consideration that the demand is uncertain in real-world scenarios, the optimization problem could be modeled as a stochastic problem.

Finally, practical applications in the industry should be explored. For example, integrating the algorithm into a dynamic pricing grid system, that adjusts in real time based on dispatch demand fluctuations.

Bibliography

[1] Balin, S. (2011). Parallel machine scheduling with fuzzy processing times using a robust genetic algorithm and simulation. *Information Sciences, Elsevier*(181), 3551-3569. doi: 10.1016/j.ins.2011.04.010

[2] Beek, O., Roa, B., Dullaert, W., & Vigo, D. (2018). An efficient implementation of a static move descriptor-based local search heuristic. *Computer & Operations Research, Elsevier*(94), 1-10. doi: 10.1016/j.cor.2018.01.006

[3] Bertsimas, D., & Tsitsiklis, J. N. (1997). *Introduction to linear optimization*. Athena Scientific and Dynamic Ideas.

[4] Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G., & Weglarz, J. (2001). *Scheduling computer and manufacturing processes*. Berlin Heidelberg: Springer-Verlag.

[5] Boyd, S., & Vandenberghe, L. (2009). *Convex optimization*. Cambridge University Press.

[6] Brucker, P. (2007). *Scheduling algorithms*. Springer Berlin.

[7] Forel, A., Parmentier, A., & Vidal, T. (2023). Explainable data-driven optimization: From context to decision and back again. *Proceedings of Machine Learning Research*, *202*, 10170-10187. doi: 10.48550/arXiv.2301.10074

[8] Ghatak, A. (2019). *Deep learning with r*. Springer Singapore.

[9] Korupolu, M. R., & Plaxton, C. G. (2000). Analysis of a local search heuristic for facility location problems. *Journal of Algorithhms, Elsevier*(37), 146-188. doi: 10.1006/jagm.2000.1100

[10] Kotu, V., & Deshpande, B. (2019). *Data science: Concepts and practice*. Morgan Kaufmann.

[11] Kuhn, M., & Johnson, K. (2013). *Applied predictive modeling*. Springer New York.

[12] Kumar, A., & Jain, M. (2020). *Ensemble learning for ai developers*. Apress Berkeley.

[13] Lenstra, J., Kan, A. R., & Brucker, P. (1997). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, *1*, 343-362.

[14] Lombardi, M., Milano, M., & Bartolini, A. (2016). Empirical decision model learning. *Artificial Intelligence, Elsevier*(244), 343-367. doi: 10.1016/j.artint.2016.01.005

[15] López-Rojas, A. D., & Cruz-Villar, C. A. (2024). Neural networks as an approximator for a family of optimization algorithm solutions for online applications. *Neural Computing and Applications*, *36*, 3125-3140. doi: 10.1007/s00521-023-09203-7

[16] Maghrebi, M., Periaraj, V., Waller, S. T., & Sammut, C. (2014a). Solving ready mixed concrete delivery problems: Evolutionary comparison between column generation and

robust generation algorithm. *Computer in Civil and Building Engineering*, 1417-1424.

[17] Maghrebi, M., Periaraj, V., Waller, S. T., & Sammut, C. (2014b). Using benders descomposition for solving ready mixed concrete dispatching problems. *International Symposium on Automation and Robotics in Construction and Mining*.

[18] Maghrebi, M., Periaraj, V., Waller, S. T., & Sammut, C. (2016). Column generation-based approach for solving large-scale ready mixed concrete delivery dispatching problems. *Computer-Aided Civil and Infrastructure Engineering*(31), 145-159. doi: 10.1111/mice.12182

[19] Marinakis, Y., Migdalas, A., & Pardalos, P. M. (2006). A new bilevel formulation for vehicle routing problem and a solution method using a genetic algorithm. *Journal of Global Optimization, Springer Science+Business Media*(38), 555-580. doi: 10.1007/s10898-006-9094-0

[20] Martí, R., Pardalos, P. M., & Resende, M. G. C. (2018). *Handbook of heuristics*. Springer Cham.

[21] Pinedo, M. L. (2012). *Scheduling: Theory, algorithms, and systems*. Springer Cham.

[22] Rebala, G., Ravi, A., & Churiwala, S. (2019). *An introduction to machine learning*. Springer Nature Switzerland AG.

[23] Resende, M. G., & Ribeiro, C. C. (2013). *Grasp: Greedy randomized adaptive search procedures*. Boston: Springer.

[24] scikit learn. (2025). *Rfecv*.

[25] Spieckermann, C., Minner, S., & Schiffer, M. (2025). Reduce-then-optimize for the fixed-charge transportation problem. *Transportation Science*, *59*(3), 540-564. doi: 10.1287/trsc.2023.0407

[26] Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*. John Wiley & Sons.

[27] Teodoro, G. D., Monaci, M., & Palagi, L. (2024). Unboxing tree ensembles for interpretability: A hierarchical visualization tool and a multivariate optimal re-built tree. *EURO Journal on Computational Optimization, Elsevier*, *12*(100084), 2192-4406. doi: 10.1016/j.ejco.2024.100084

[28] Xu, K., Shen, L., & Liu, L. (2025). Enhancing column generation by reinforcement learning-based hyper-heuristic for vehicle routing and scheduling problems. *Computers & Industrial Engineering*, *206*. doi: 10.1016/j.cie.2025.111138

[29] Zhou, Z.-H. (2021). *Machine learning*. Springer Nature Singapore.

Annexes

Annex A. Preliminary Analysis

In a preliminary stage of this research, three type of Machine Learning (ML) models were tested: Linear Regression, Support Vector Machine (SVM) and Random Forest (RF).

Multivariate Linear Regressions were trained and evaluated as part of a preliminary exploratory analysis. The main results are summarized in Table A.1.

Table A.1: Linear Regression R^2 results across variable sets

Variable Set	$\mathbf{R^2}$
tasks_quantity, Subset_range	0.056
tasks_quantity, Dist_tasks, Subset_range	0.112
tasks_quantity, Min_spacing	0.081
Min_spacing, Size	0.060
tasks_quantity, Dist_tasks	0.088
Dist_tasks, Min_spacing	0.048
Dist_tasks, Size	0.067

A comparative analysis of the performance of SVM and RF classifiers is presented in Table A.3 and Figure A.1, where their results are evaluated using the F1-score. Analogously, the performance of SVM and RF regressors is shown in Table A.4 and Figure A.2, evaluated by the R^2 metric. In both cases, the variable sets used are described in Table A.2.

Table A.2: Variable Sets Used in the Experiments

Variable Set	**Variables**
A	trains_quantity, Mean_tproc, dd_last_task
A2	trains_quantity, Min_spacing, dd_last_task, Subset_range
B	Size, Min_spacing, dd_last_task, Subset_range
C	Mean_tproc, Size, dd_last_task, Subset_range
C2	Mean_tproc, tasks_quantity, dd_last_task, Subset_range
D	Dist_tasks, Size, Subset_range, Min_spacing, dd_first_task
D2	Dist_tasks, Size, Mean_spacing, Mean_tproc, dd_first_task

Table A.3: F1-scores for Random Forest and SVM across variable sets

Model	Variable Set	Class 0	Class 1	Class 2	Accuracy	Macro Avg	Weighted Avg
RF	A	0.81	0.69	0.82	0.78	0.78	0.78
RF	A2	0.81	0.67	0.77	0.76	0.75	0.76
RF	B	0.81	0.64	0.77	0.75	0.74	0.75
RF	C	0.82	0.67	0.78	0.76	0.76	0.76
RF	C2	0.82	0.59	0.74	0.74	0.72	0.73
RF	D	0.82	0.71	0.80	0.78	0.77	0.78
RF	D2	0.84	0.68	0.79	0.78	0.77	0.78
SVM	A	0.68	0.49	0.54	0.60	0.57	0.59
SVM	A2	0.65	0.45	0.48	0.56	0.53	0.55
SVM	B	0.63	0.34	0.58	0.55	0.51	0.53
SVM	C	0.62	0.24	0.43	0.49	0.43	0.46
SVM	C2	0.61	0.26	0.43	0.49	0.43	0.46
SVM	D	0.64	0.39	0.54	0.55	0.52	0.54
SVM	D2	0.68	0.24	0.50	0.54	0.47	0.50

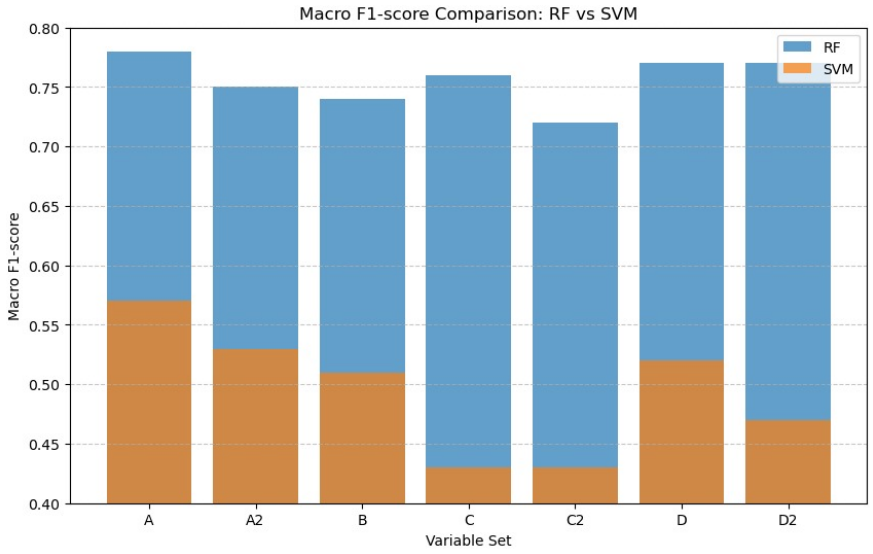


Figure A.1: Comparison of Random Forest (RF) and Support Vector Machine (SVM) classifiers performance, using F1-score

Table A.4: R^2 performance for SVR and Random Forest across variable sets

Model	**Variable Set**	**R^2**
SVR	A	0.0134
SVR	A2	0.8015
SVR	B	0.8121
SVR	C	0.3906
SVR	C2	0.5819
SVR	D	0.9350
SVR	D2	0.1016
RF	A	0.6134
RF	A2	0.8985
RF	B	0.8960
RF	C	0.9462
RF	C2	0.9696
RF	D	0.9711
RF	D2	0.9515

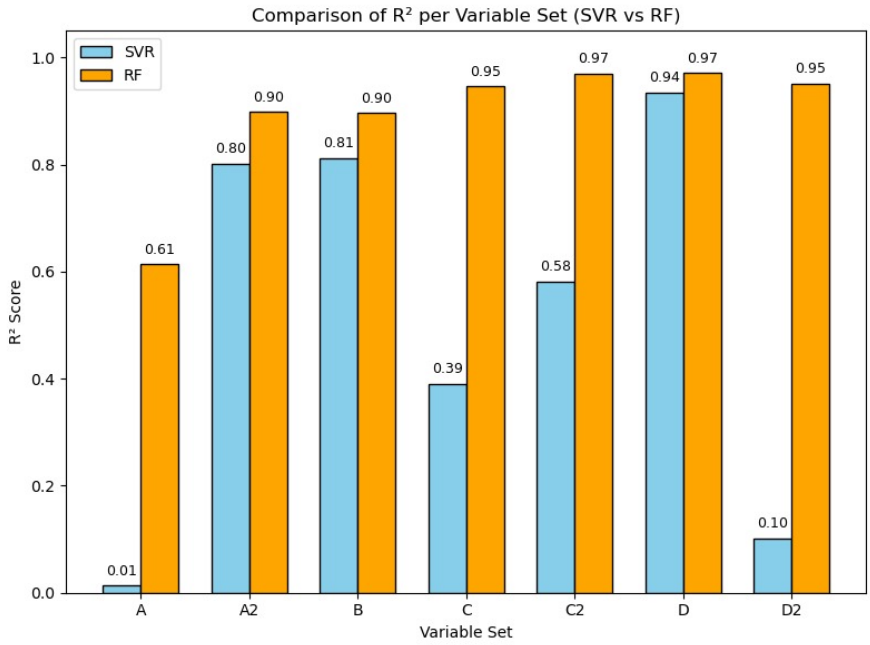


Figure A.2: Performance comparison of Random Forest (RF) and Support Vector Machine (SVM) regressors, using RMSE

Annex B. Dataset used for the Optimization Model and Heuristics

As described in Chapter 3, the original dataset was used for the implementation of both the optimization model and the heuristics. The complete list of attributes, along with their respective characteristics, is presented in Table B.1.

Attribute	Description	Non-null count	Type
`order_id`	Order identification number	3366	object
`dispatch_id`	Dispatch identification number	3366	object
`block`	identifier of the time block in which the order is required to be received	3366	object
`due_date_day`	Day of the week when the order is due for delivery to the client	3366	object
`due_date_time`	Time of the day when the order is due for delivery to the client	3366	object
`spacing`	time between the dispatches received by the customers	3366	int64
`dispatch_train_length`	number of dispatches of the dispatch train	3366	int64
`full_order_size`	size of the complete order placed by the customer	3366	float64
`load_size`	size of the dispatch	3366	float64
`unload_time`	unloading time	3366	float64
`travel_time_p1`	travel time from the machine $p = 1$ in the concrete plant to the customer, using a truck	2371	float64
`return_time_p1`	return time of the truck from the customer to the machine $p = 1$ of the concrete plant	2371	float64
`travel_time_p2`	travel time from the machine $p = 2$ in the concrete plant to the customer, using a truck	1835	float64
`return_time_p2`	return time of the truck from the customer to the machine $p = 2$ of the concrete plant	1835	float64
`travel_time_p3`	travel time from the machine $p = 3$ in the concrete plant to the customer, using a truck	2605	float64
`return_time_p3`	return time of the truck from the customer to the machine $p = 3$ of the concrete plant	2605	float64
`travel_time_p4`	travel time from the machine $p = 4$ in the concrete plant to the customer, using a truck	2009	float64
`return_time_p4`	return time of the truck from the customer to the machine $p = 4$ of the concrete plant	2009	float64

Continued on the next page

Attribute	Description	Non-null count	Type
travel_time_p5	travel time from the machine $p = 5$ in the concrete plant to the customer, using a truck	1903	float64
return_time_p5	return time of the truck from the customer to the machine $p = 5$ of the concrete plant	1903	float64
travel_time_p6	travel time from the machine $p = 6$ in the concrete plant to the customer, using a truck	2698	float64
return_time_p6	return time of the truck from the customer to the machine $p = 6$ of the concrete plant	2698	float64
travel_time_p7	travel time from the machine $p = 7$ in the concrete plant to the customer, using a truck	2982	float64
return_time_p7	return time of the truck from the customer to the machine $p = 7$ of the concrete plant	2982	float64
travel_time_p8	travel time from the machine $p = 8$ in the concrete plant to the customer, using a truck	1108	float64
return_time_p8	return time of the truck from the customer to the machine $p = 8$ of the concrete plant	1108	float64
travel_time_p9	travel time from the machine $p = 9$ in the concrete plant to the customer, using a truck	2321	float64
return_time_p9	return time of the truck from the customer to the machine $p = 9$ of the concrete plant	2321	float64
travel_time_p10	travel time from the machine $p = 10$ in the concrete plant to the customer, using a truck	1796	float64
return_time_p10	return time of the truck from the customer to the machine $p = 10$ of the concrete plant	1796	float64
load_time_p1	load time of each task in the machine $p = 1$	3366	float64
load_time_p2	load time of each task in the machine $p = 2$	3366	float64
load_time_p3	load time of each task in the machine $p = 3$	3366	float64
load_time_p4	load time of each task in the machine $p = 4$	3366	float64
load_time_p5	load time of each task in the machine $p = 5$	3366	float64
load_time_p6	load time of each task in the machine $p = 6$	3366	float64
load_time_p7	load time of each task in the machine $p = 7$	3366	float64

Continued on the next page

Attribute	Description	Non-null count	Type
`load_time_p8`	load time of each task in the machine $p = 8$	3366	float64
`load_time_p9`	load time of each task in the machine $p = 9$	3366	float64
`load_time_p10`	load time of each task in the machine $p = 10$	3366	float64

Table B.1: Attributes of the original dataset, used to implement the optimization model and heuristics

Annex C. Random Feature Selection Results for Each Model with Metrics

As part of the Machine Learning model fitting process explained in Chapter 4, a feature selection method was applied. Specifically, Recursive Feature Selection with Cross-Validation (RFSCV) was implemented for both the regressors and classifiers. The models were optimized using RMSE for regressors and F1-score for classifiers. The corresponding results are summarized in Table C.1 and Table C.2, respectively.

Model	Kfold	RMSE	RMSE Variables
RFR	12	64.171963	trains_quantity, tasks_quantity, Dist_tasks, Machines_quantity, Mean_spacing, Min_spacing, Mean_tproc, dd_last_task, tasks_quantity_afternoon, tasks_quantity_last, Disp_prom_orders, Mean_dispatch_train_length, Max_tproc, Min_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks, trains_quantity_10_14_tasks
	15	64.484659	tasks_quantity, Dist_tasks, Machines_quantity, Mean_spacing, Mean_tproc, dd_last_task, tasks_quantity_afternoon, Mean_dispatch_train_length, Max_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks
	5	65.351787	trains_quantity, tasks_quantity, Dist_tasks, Machines_quantity, Mean_spacing, Min_spacing, Mean_tproc, dd_last_task, tasks_quantity_afternoon, Disp_prom_orders, Mean_dispatch_train_length, Max_tproc, Min_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks
	8	65.631521	trains_quantity, tasks_quantity, Dist_tasks, Day, Size, Machines_quantity, Mean_spacing, Min_spacing, Mean_tproc, dd_first_task, dd_last_task, tasks_quantity_afternoon, tasks_quantity_last, Disp_prom_orders, Mean_dispatch_train_length, Max_dispatch_train_length, Max_spacing, Max_tproc, Min_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks, trains_quantity_6_9_tasks, trains_quantity_10_14_tasks
ABR	12	70.223272	trains_quantity, tasks_quantity, Dist_tasks, Machines_quantity, Mean_spacing, Mean_tproc, dd_last_task, tasks_quantity_afternoon, Disp_prom_orders, Mean_dispatch_train_length, Max_tproc, Min_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks
	8	70.238979	tasks_quantity, Dist_tasks, Machines_quantity, Mean_spacing, Mean_tproc, tasks_quantity_afternoon, Max_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks
	15	70.308864	trains_quantity, tasks_quantity, Dist_tasks, Machines_quantity, Mean_spacing, Mean_tproc, dd_last_task, tasks_quantity_afternoon, Disp_prom_orders, Max_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks

Continued on the next page

Model	Kfold	RMSE	RMSE Variables
XGBR	5	70.581900	tasks_quantity, Dist_tasks, Machines_quantity, Mean_spacing, Mean_tproc, tasks_quantity_afternoon, Disp_prom_orders, Max_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks
	15	66.319125	trains_quantity, tasks_quantity, Dist_tasks, Machines_quantity, dd_last_task, tasks_quantity_afternoon, trains_quantity_1_tasks, trains_quantity_10_14_tasks
	12	67.252617	trains_quantity, tasks_quantity, dd_last_task, tasks_quantity_afternoon, trains_quantity_1_tasks, trains_quantity_10_14_tasks
	8	67.827890	trains_quantity, tasks_quantity, Dist_tasks, dd_last_task, tasks_quantity_afternoon, trains_quantity_1_tasks, trains_quantity_10_14_tasks
	5	70.490898	trains_quantity, tasks_quantity, dd_last_task, tasks_quantity_afternoon, trains_quantity_1_tasks, trains_quantity_10_14_tasks

Table C.1: Cross-validation RMSE results for regression models

Model	Kfold	F1-score	F1-score Variables
RFC	5	0.98	trains_quantity, tasks_quantity, Dist_tasks, Size, Machines_quantity, Mean_spacing, Min_spacing, Mean_tproc, dd_first_task, dd_last_task, tasks_quantity_afternoon, tasks_quantity_last, Disp_prom_orders, Mean_dispatch_train_length, Max_dispatch_train_length, Max_tproc, Min_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks
	8	0.98	trains_quantity, tasks_quantity, Dist_tasks, Size, Machines_quantity, Mean_spacing, Min_spacing, Mean_tproc, dd_first_task, dd_last_task, tasks_quantity_afternoon, tasks_quantity_last, Disp_prom_orders, Mean_dispatch_train_length, Max_dispatch_train_length, Max_tproc, Min_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks
	12	0.98	trains_quantity, tasks_quantity, Dist_tasks, Day, Size, Subset_range, Machines_quantity, Mean_spacing, Min_spacing, Mean_tproc, dd_first_task, dd_last_task, tasks_quantity_morning, tasks_quantity_afternoon, tasks_quantity_last, Disp_med_orders, Disp_prom_orders, Mean_dispatch_train_length, Max_dispatch_train_length, Max_spacing, Max_tproc, Min_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks, trains_quantity_6_9_tasks, trains_quantity_10_14_tasks

Continued on the next page

Model	Kfold	F1-score	F1-score Variables
	15	0.98	trains_quantity, tasks_quantity, Dist_tasks, Day, Size, Subset_range, Machines_quantity, Mean_spacing, Min_spacing, Mean_tproc, dd_first_task, dd_last_task, tasks_quantity_morning, tasks_quantity_afternoon, tasks_quantity_last, Disp_med_orders, Disp_prom_orders, Mean_dispatch_train_length, Max_dispatch_train_length, Min_dispatch_train_length, Max_spacing, Max_tproc, Min_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks, trains_quantity_6_9_tasks, trains_quantity_10_14_tasks, trains_quantity_15___tasks
ABC	5	0.96	trains_quantity, tasks_quantity, Dist_tasks, Size, Machines_quantity, Mean_spacing, Min_spacing, Mean_tproc, dd_first_task, dd_last_task, tasks_quantity_afternoon, Disp_med_orders, Disp_prom_orders, Mean_dispatch_train_length, Max_dispatch_train_length, Max_spacing, Max_tproc, Min_tproc, trains_quantity_2_5_tasks, trains_quantity_6_9_tasks, trains_quantity_10_14_tasks
	8	0.96	trains_quantity, tasks_quantity, Dist_tasks, Machines_quantity, Min_spacing, Mean_tproc, Mean_dispatch_train_length, Max_dispatch_train_length, trains_quantity_2_5_tasks
	12	0.96	trains_quantity, tasks_quantity, Dist_tasks, Day, Size, Subset_range, Machines_quantity, Mean_spacing, Min_spacing, Mean_tproc, dd_first_task, dd_last_task, tasks_quantity_morning, tasks_quantity_afternoon, tasks_quantity_last, Disp_med_orders, Disp_prom_orders, Mean_dispatch_train_length, Max_dispatch_train_length, Min_dispatch_train_length, Max_spacing, Max_tproc, Min_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks, trains_quantity_6_9_tasks, trains_quantity_10_14_tasks, trains_quantity_15___tasks
	15	0.96	trains_quantity, tasks_quantity, Dist_tasks, Size, Machines_quantity, Mean_spacing, Min_spacing, Mean_tproc, dd_first_task, dd_last_task, tasks_quantity_morning, tasks_quantity_afternoon, tasks_quantity_last, Disp_med_orders, Disp_prom_orders, Mean_dispatch_train_length, Max_dispatch_train_length, Max_spacing, Max_tproc, Min_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks, trains_quantity_6_9_tasks, trains_quantity_10_14_tasks

Continued on the next page

Model	Kfold	F1-score	F1-score Variables
XGBC	5	0.98	trains_quantity, tasks_quantity, Dist_tasks, Day, Size, Subset_range, Machines_quantity, Mean_spacing, Min_spacing, Mean_tproc, dd_first_task, dd_last_task, tasks_quantity_morning, tasks_quantity_afternoon, tasks_quantity_last, Disp_med_orders, Disp_prom_orders, Mean_dispatch_train_length, Max_dispatch_train_length, Min_dispatch_train_length, Max_spacing, Max_tproc, Min_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks, trains_quantity_6_9_tasks, trains_quantity_10_14_tasks
	8	0.98	trains_quantity, tasks_quantity, Dist_tasks, Day, Size, Subset_range, Machines_quantity, Mean_spacing, Min_spacing, Mean_tproc, dd_first_task, dd_last_task, tasks_quantity_afternoon, tasks_quantity_last, Disp_med_orders, Disp_prom_orders, Mean_dispatch_train_length, Max_dispatch_train_length, Min_dispatch_train_length, Max_spacing, Max_tproc, Min_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks, trains_quantity_6_9_tasks, trains_quantity_10_14_tasks
	12	0.94	trains_quantity, tasks_quantity, Dist_tasks, Day, Subset_range, Machines_quantity, Mean_spacing, Min_spacing, dd_first_task, dd_last_task, tasks_quantity_afternoon, tasks_quantity_last, Disp_med_orders, Disp_prom_orders, Mean_dispatch_train_length, Max_dispatch_train_length, Max_spacing, Max_tproc, Min_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks, trains_quantity_6_9_tasks, trains_quantity_10_14_tasks
	15	0.94	trains_quantity, tasks_quantity, Dist_tasks, Day, Machines_quantity, Mean_spacing, Min_spacing, dd_first_task, dd_last_task, tasks_quantity_afternoon, tasks_quantity_last, Disp_med_orders, Disp_prom_orders, Mean_dispatch_train_length, Max_dispatch_train_length, Max_spacing, Max_tproc, Min_tproc, trains_quantity_1_tasks, trains_quantity_2_5_tasks, trains_quantity_6_9_tasks, trains_quantity_10_14_tasks

Table C.2: Cross-validation RMSE results for classification models

Annex D. Parameter Tuning and Sensitivity Analysis Results

As part of the Machine Learning model fitting process, explained in Chapter 4, a parameter tuning method was applied. The results corresponding to the top five parameter combinations for classifiers, ranked by their F1-score performance, are presented in the following tables. Namely, Table D.1 for XGboost, Table D.2 for AdaBoost and D.3 for Random Forest.

Table D.1: Parameter tuning results for XGBoost Classifier

Booster	**Max Depth**	**Learning Rate**	**Tree Method**	**F1-score**
gblinear	3.0	0.01	hist	0.530645
gblinear	3.0	0.01	approx	0.530645
gblinear	3.0	0.01	auto	0.530645
gblinear	3.0	0.01	exact	0.530645
gblinear	4.0	0.01	exact	0.530645

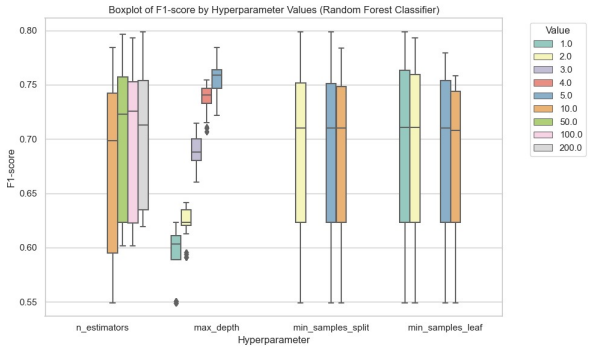
Table D.2: Parameter tuning results for AdaBoost Classifier

n_estimators	**learning_rate**	**F1-score**	**Accuracy**
100	0.01	0.742835	0.744619
100	0.30	0.748072	0.748141
200	0.30	0.748199	0.748174
100	0.10	0.748408	0.750008
50	0.01	0.748764	0.750008

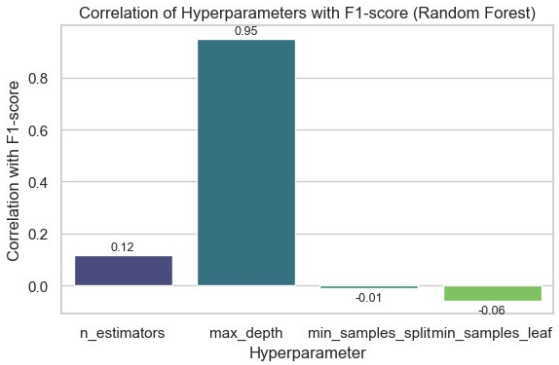
Table D.3: Parameter tuning results for Random Forest Classifier

ID	**n_estimators**	**max_depth**	**min_samples_leaf**	**criterion**	**F1-score**
4	10	1.0	2	entropy	0.549212
5	10	1.0	2	log_loss	0.549212
1	10	1.0	1	entropy	0.549212
2	10	1.0	1	log_loss	0.549212
25	10	1.0	10	entropy	0.549212

Additionally, a sensitivity analysis was performed for each model, in both its regressor and classifier forms. In particular, boxplots and correlation charts were generated to evaluate the impact of hyperparameters on performance (F1-score for classifiers and RMSE for regressors): D.1 for Random Forest classifier, in Figure D.2 for AdaBoost classifier, in Figure D.3 for XGBoost classifier, in Figure D.4 for Random Forest regressor, in Figure D.5 for AdaBoost regressor, in Figure D.6 for XGBoost regressor.

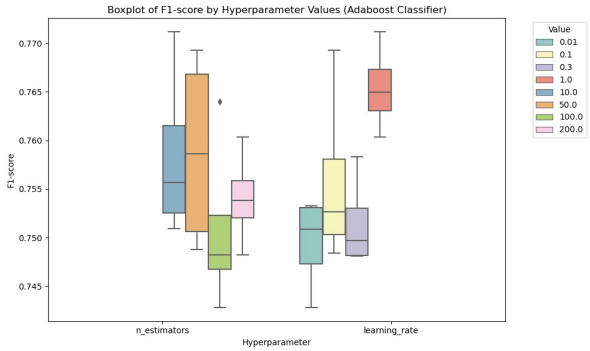


(a) Boxplot of F1-score values by hyperparameter value

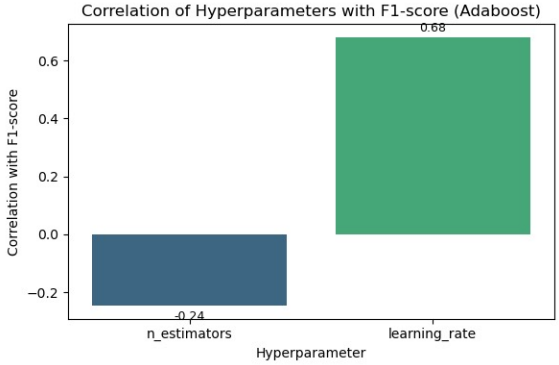


(b) Correlation between hyperparameters and F1-score

Figure D.1: Random Forest Classifier Sensitivity Analysis

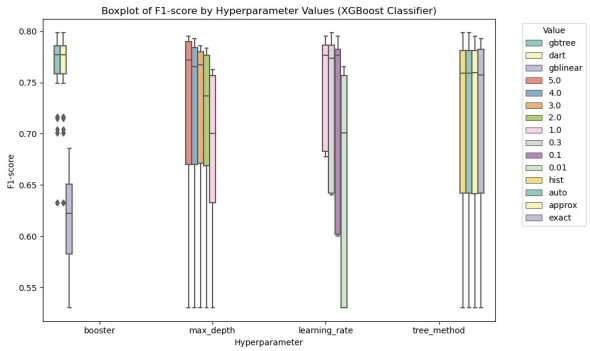


(a) Boxplot of F1-score values by hyperparameter value

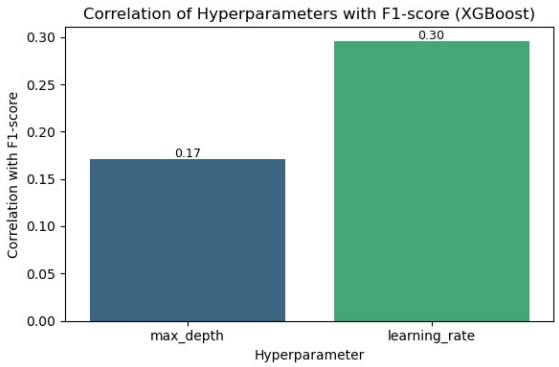


(b) Correlation between hyperparameters and F1-score

Figure D.2: AdaBoost Classifier Sensitivity Analysis

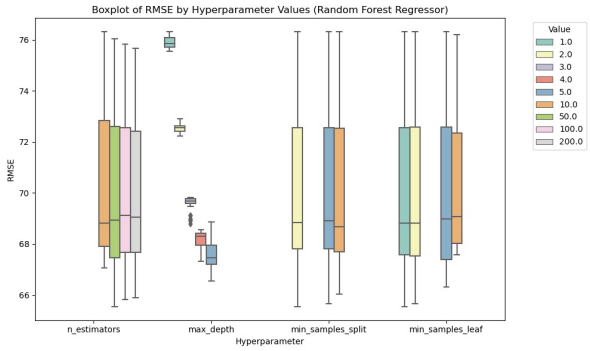


(a) Boxplot of F1-score values by hyperparameter value

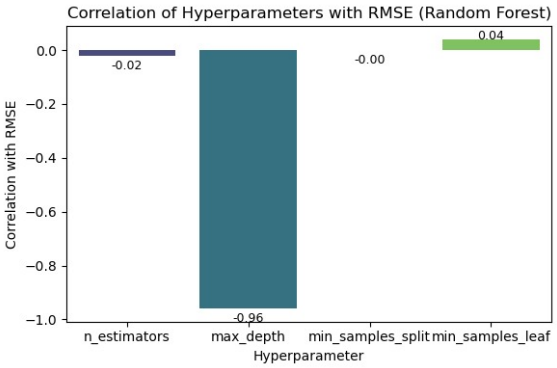


(b) Correlation between hyperparameters and F1-score

Figure D.3: XGBoost Classifier Sensitivity Analysis

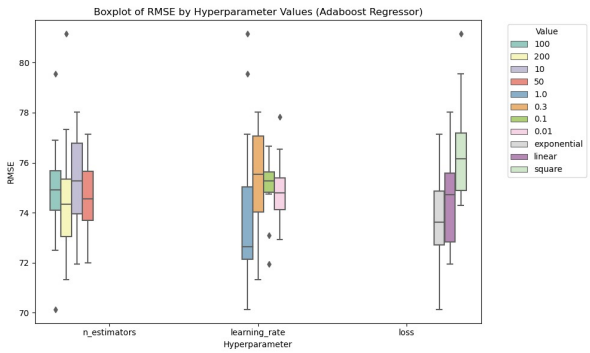


(a) Boxplot of RMSE values by hyperparameter value

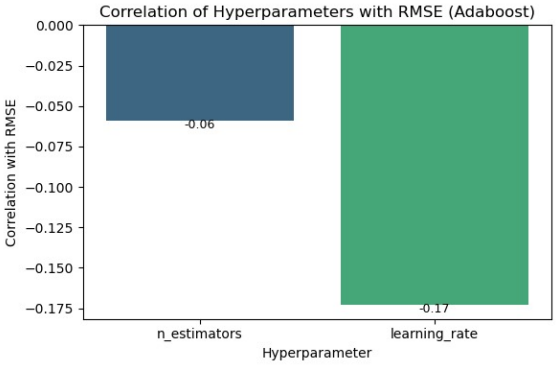


(b) Correlation between hyperparameters and RMSE

Figure D.4: Random Forest Regressor Sensitivity Analysis

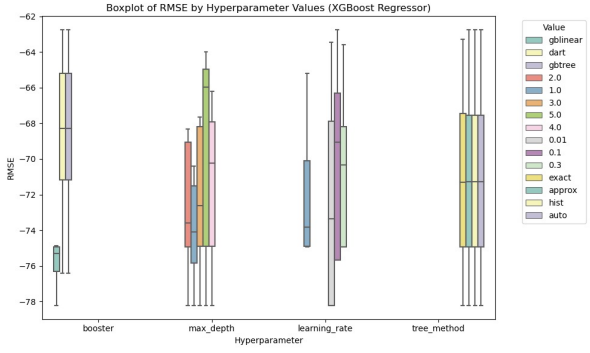


(a) Boxplot of RMSE values by hyperparameter value

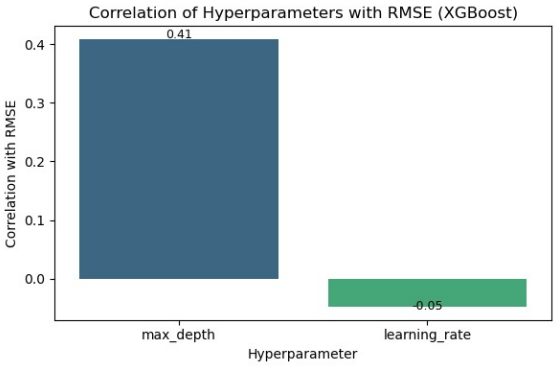


(b) Correlation between hyperparameters and RMSE

Figure D.5: AdaBoost Regressor Sensitivity Analysis



(a) Boxplot of RMSE values by hyperparameter value



(b) Correlation between hyperparameters and RMSE

Figure D.6: XGBoost Regressor Sensitivity Analysis