HEURISTICS FOR SIMULTANEOUS LOT SIZING AND SCHEDULING PROBLEM

CEVDET UTKU ŞAFAK

ÖZYEĞİN UNIVERSITY

JUNE, 2025

HEURISTICS FOR SIMULTANEOUS LOT SIZING AND SCHEDULING PROBLEM

by
Cevdet Utku Şafak

Dissertation
Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy

in
Industrial Engineering

Advisor: Asst. Prof. Erinç Albey

Graduate School of Science and Engineering
Özyeğin University
İstanbul

June, 2025

HEURISTICS FOR SIMULTANEOUS LOT SIZING AND SCHEDULING PROBLEM

Approved by:

Asst. Prof. Erinç Albey, Advisor
Department of Artificial Intelligence and Data Engineering
Özyeğin University

Prof. Mehmet Güray Güler
Department of Data Science and Analytics
İstanbul Technical University

Asst. Prof. Görkem Yılmaz, Co-advisor
Department of Industrial Engineering
İzmir University of Economics

Asst. Prof. Mehmet Önal
Department of Industrial Engineering
Özyeğin University

Assoc. Prof. İhsan Yankıoğlu
Department of Industrial Engineering
Özyeğin University

Asst. Prof. Umman Mahir Yıldırım
Department of Industrial Engineering
İstanbul Bilgi University

Approval Date: May 20, 2025

To my dear daughter Hayal;

and my advisors and family - thank you for your unwavering support.

DECLARATION OF ORIGINALITY

I hereby declare that I am the sole author of this thesis and that this is the true copy of my thesis, including the final revisions, approved by my thesis committee. All data and information have been obtained, produced, and presented in accordance with the rules of research ethics and principles of academic honesty. As required by these rules, to the best of my knowledge I have acknowledged ideas, thoughts, and any copyrighted material in accordance with the standard referencing rules. I certify that any part of this thesis has not been submitted for a degree or diploma in another educational institution.

Cevdet Utku Şafak

ABSTRACT

This study tackles the Capacitated lot-sizing problem (CLSP) enriched with industrial complexities such as sequence-dependent setup times, setup carryovers, secondary resource constraints, and parallel non-identical machines. These characteristics pose significant challenges for traditional optimisation techniques, especially when scaling to large, real-world problem instances.

To address these challenges, we develop a novel column generation heuristic framework. The method decomposes the problem into a Restricted master problem (RMP) and pricing subproblems, which dynamically generate production patterns based on dual information. Unlike classical exact methods or generic heuristics, our framework incorporates a problem-specific neighbourhood search that guides pattern generation heuristically. This results in high-quality solutions within computational time frames that are acceptable for industrial applications.

The proposed Column generation neighbourhood search (CGNS) algorithm is validated on synthetic benchmarks and real-world data from a plastic injection facility. Results show that our method outperforms conventional fix-and-relax heuristics and achieves performance comparable to commercial solvers, while maintaining superior scalability. The modular structure of the algorithm also enables its integration with machine learning-based decision support systems, paving the way for hybrid optimisation approaches implemented as an improvement to CGNS algorithm as Column generation neighbourhood search with neural networks (CGNSNN).

Keywords: lot-sizing and scheduling, sequence-dependent setup times and costs, setup carryover, column generation, neighbourhood search heuristic, neural networks, machine learning, machine learning in optimisation, plastic injection

ÖZET

Bu çalışma, ardışık-bağımlı kurulum süreleri, kurulum taşıma, ikincil kaynak kısıtları ve paralel ve birbirinden farklı makineler gibi endüstriyel karmaşıklıklarla zenginleştirilmiş CLSP problemini ele almaktadır. Bu özellikler, özellikle büyük ölçekli ve gerçek dünya problem ölçeğinde, geleneksel optimizasyon teknikleri için önemli zorluklar oluşturur.

Bu zorlukların üstesinden gelebilmek için, yenilikçi bir kolon üretimi sezgisel (heuristic) algoritması geliştirilmiştir. Önerilen yöntem, problemi bir kısıtlı ana probleme ayırarak, ikincil problem bilgilerine dayalı olarak üretim desenlerini dinamik biçimde oluşturur. Klasik karma tam sayılı doğrusal problem eniyileme yöntemlerden veya genel sezgisel yaklaşımlardan farklı olarak, desen üretimini sezgisel olarak yönlendiren probleme özgü bir komşuluk arama (neighbourhood search) mekanizması içerir. Bu yapı, endüstriyel uygulamalarda kabul edilebilir süreler içinde yüksek kaliteli çözümler elde edilmesini sağlamaktadır.

Önerilen CGNS algoritması, sentetik kıyaslamalar (benchmark) ve bir plastik enjeksiyon tesisine ait gerçek dünya verileri üzerinde doğrulanmıştır. Elde edilen sonuçlar, geliştirilen yöntemin klasik fix-and-relax sezgisel yaklaşımlardan daha başarılı olduğunu ve ticari çözücülerle benzer performans sergilerken üstün ölçeklenebilirlik göstermektedir. Algoritmanın modüler yapısı, makine öğrenimi tabanlı karar destek sistemleriyle entegrasyonuna da olanak tanımaktadır. Bu da, CGNS algoritmasının geliştirilmiş bir versiyonu olarak sunulan yapay sinir ağları ile desteklenmiş hibrit optimizasyonun (CGNSNN) yolunu açmaktadır.

Anahtar Kelimeler: parti boyutu belirleme ve çizelgeleme, ardışık-sıra-bağımlı kurulum süreleri ve maliyetleri, kurulum taşıma, kolon türetme, komşuluk arama sezgiseli, yapay sinir ağları, makine öğrenimi, eniyilemede makine öğrenimi, plastik enjeksiyon

ACKNOWLEDGEMENTS

The work reported in this study was supported by the The Scientific and Technological Research Council of Turkey (TÜBİTAK) 2244 Industrial Ph.D. Fellowship Programme under grant number 118C138.

TABLE OF CONTENTS

DECLARATION OF ORIGINALITY iv

ABSTRACT v

ÖZET vi

ACKNOWLEDGEMENTS vii

LIST OF TABLES x

LIST OF FIGURES xii

LIST OF ACRONYMS AND ABBREVIATIONS xiii

1 INTRODUCTION 1

2 PROBLEM DEFINITION 5

3 LITERATURE REVIEW 8

3.1 Mathematical model of the General Lot-sizing and Scheduling Problem (GLSP) 8

3.2 Classification of Lot Sizing and Scheduling Problems 11

3.3 Literature Review on Big-Bucket Formulations 13

3.4 Machine Learning Approaches for Solving Mixed Integer Linear Programming (MILP)s 16

3.5 Gaps and Motivation 20

4 SOLUTION ALGORITHMS 22

4.1 Exact Formulation for Capacitated lot-sizing with sequence-dependent setups (CLSD) 22

4.2 CGNS Algorithm 25

4.2.1 Restricted master problem 28

4.2.2 Restricted linear problem 33

4.2.3 Column search heuristics 34

4.3 Neural Network-Based Column Pool Generation 41

4.3.1 Feature Selection and Proposed Neural Network Structures 43

4.3.2 Integration of Neural Network Predictions into Column Generation and Experimental Results 48
4.3.3 Neural Network Training and Performance Evaluation 51
5 NUMERICAL STUDIES 54
5.1 Numerical results I: validation and comparison with benchmark algorithms 54
5.2 Numerical results II: A real-life case 62
6 CONCLUSIONS 70
REFERENCES 72
APPENDICES 77
APPENDIX A — MIQP FORMULATION 78
APPENDIX B — HDFO ALGORITHM SUMMARY 80
APPENDIX C — SODBS ALGORITHM SUMMARY 83
APPENDIX D — MACHINE LEARNING APPROACHES 85
VITA 88

LIST OF TABLES

Table 3.1: Parameters of the GLSP Model. 9

Table 3.2: Decision Variables of the general GLSP Model. 10

Table 3.3: Literature Review Summary. 15

Table 3.4: Summary of Contributions from Key Papers. 19

Table 4.1: Indices. 22

Table 4.2: Model Parameters. 23

Table 4.3: Decision Variables. 23

Table 4.4: RMP Model Parameters. 29

Table 4.5: RMP Model Decision Variables. 31

Table 4.6: Neural Network Input Features: Searched Part i^*. 44

Table 4.7: Neural Network Input Features: Parts in the Searched Set A. 45

Table 4.8: Confusion Matrices and Performance Metrics for Full and Balanced Datasets (Multi Layer Perceptron (MLP) Model). 53

Table 4.9: Confusion Matrices and Performance Metrics for Full and Balanced Datasets (Recurrent Neural Network (RNN) Model). 53

Table 5.1: Number of Columns Generated by the CGNS and CGNSNN Algorithms. 56

Table 5.2: Validation and Comparison of Column Search Heuristics. 57

Table 5.3: Overall Comparison of the Proposed Algorithms and Benchmarks. 59

Table 5.4: Paired t-test Results Comparing Proposed Algorithms and Gurobi. 60

Table 5.5: Comparison of Benchmark Algorithms on Large Datasets with Extended Initialization Time Limits. 61

Table 5.6: Overall Comparison of the Proposed Algorithm and the Company's Original Plan. 65

Table 5.7: Cost Comparisons of the Proposed Algorithm and the Company's Original Plans Regarding Setup, Inventory, Backlog, and Production Costs... 66

Table 5.8: Variation of the Costs Due to the Change in Unit Backlogging Cost Comparison of the Company's Original Plan and the Proposed Algorithm. 67

Table 5.9: Paired t-test Comparison of CGNS and CGNSNN Total Costs on Real World Instances. 69

Table D.1: Comparison of Neural Network-Based Learning Approaches. 85

LIST OF FIGURES

Figure 2.1: Plastic Injection Factory Production Process Overview showing machines, moulds, feeding systems, and versioning inserts. 6

Figure 3.1: Classification Lot Sizing Scheduling Problem [1]. 12

Figure 4.1: Demonstration of Patterns, Columns, and Sequences. 26

Figure 4.2: Algorithm Flowchart. 27

Figure 4.3: Setup Carryover Column Generation. 36

Figure 4.4: Temporal CPU times of the overall algorithm. 42

Figure 4.5: Neural network structure used in the column generation process RNN version. 47

Figure 4.6: Neural network structure used in the column generation process MLP version. 48

Figure 5.1: Inventory Comparison. 64

Figure 5.2: Backlog Comparison. 64

Figure 5.3: Setup Comparison. 64

Figure 5.4: Solution Time, Objective Value Evaluation. Sensitivity Analysis Over Backlogging Costs. 68

Figure D.1: Approach 1: Overview. 85

Figure D.2: Approach 2: Overview. 86

Figure D.3: Approach 3: Overview. 86

Figure D.4: Approach 4: Overview. 87

Figure D.5: Approach 5: Overview. 87

LIST OF ACRONYMS AND ABBREVIATIONS

CLSP	Capacitated lot-sizing problem
RMP	Restricted master problem
RLP	Restricted linear problem
CGNS	Column generation neighbourhood search
CGNSNN	Column generation neighbourhood search with neural networks
GLSPCS	General Lot-Sizing and Scheduling Problem with Conservation of Setup Stages
GLSPLS	General Lot-Sizing and Scheduling Problem with Loss of Setup Stages
CLSD	Capacitated lot-sizing with sequence-dependent setups
HDFO	Hybrid decomposition fix and optimise
SODBS	Set-Oriented Data-Driven Branching and Selection
GLSP	General Lot-sizing and Scheduling Problem
SD	Sequence-dependent setups
SR	Secondary Resources
SCo	Setup Carryovers
SCr	Setup Crossovers
PLSP	Proportional Lot-sizing and Scheduling Problem
DLSP	Discrete Lot-sizing and Scheduling Problem
CSLP	Continuous Lot-sizing and Scheduling Problem

SM	Single Machine
PI	Parallel Identical Machines
PN	Parallel Non-identical Machines
ML	Multi Level Machines
MLP	Multi Layer Perceptron
RNN	Recurrent Neural Network
NN	Neural Network
GNN	Graph Neural Network
GCN	Graph Convolution Network
RL	Reinforcement Learning
RELU	Rectified Linear Unit
LG-GNN	Local and Global Information Aware Graph Neural Network
GCNN	Graph Convolution Neural Network
ECO-DQN	Exploratory Combinatorial Optimization - Deep Q Network
CP	Constraint Programming
DRL	Deep Reinforcement Learning
ALNS	Adaptive Large Neighbourhood Search
RH	Rolling Horizon
MDP	Markov Decision Process
MILP	Mixed Integer Linear Programming

1. INTRODUCTION

Production planning encompasses a range of complex decision-making activities such as lot-sizing, production scheduling, and inventory control [2]. These tasks are intricately linked, and their interdependencies often pose considerable challenges in real-world settings. One such challenge arises from the trade-off between lot-sizing and setup operations: planners frequently prefer larger lot sizes to minimise the frequency of setups [3], yet this can lead to inefficiencies in inventory and resource utilisation. This balance becomes increasingly challenging, especially in large-scale manufacturing systems with high setup variability and multiple product types. Hence, the need for scalable, efficient solution methodologies becomes critical.

This study is motivated by the need to provide an effective and scalable approach for solving large instances of the lot-sizing and scheduling problem, particularly in environments with intricate constraints such as sequence-dependent setups and secondary resource requirements. Existing literature has explored numerous variants of the General Lot-Sizing and Scheduling Problem with Conservation of Setup Stages (GLSPCS), which models the scheduling of multiple products across a defined planning horizon while conserving setup states. As highlighted by [4], GLSPCS problems are typically categorised into big-bucket and small-bucket formulations, each offering unique trade-offs in modelling granularity and computational complexity. The big-bucket formulations, particularly the CLSP and CLSD models, segment the planning horizon into macro-periods, enabling more flexible scheduling across time while accounting for complex constraints [5].

Sequence-dependent setups are particularly critical in industrial applications such as plastic injection manufacturing. Mould changes, material switches, and tooling adjustments can incur substantial setup costs and time losses. The CLSD formulation, as an extension of CLSP, integrates these sequence dependencies and allows for variable lot sizes,

multiple setup transitions, and the carryover of setup states into future periods [6]. This makes CLSD especially suitable for environments involving shared secondary resources, such as moulds [7], and for operations requiring fine-grained production planning in high demand variability [8].

However, as emphasised in [9], the problem complexity significantly increases in realistic settings where tactical-level decisions such as workforce planning, shift scheduling, and overtime management must be integrated into the operational scheduling framework. In practical environments like plastic injection plants, machines must be paired with compatible moulds, and machine-level decisions are constrained by the available workforce. These additional dimensions elevate the classical CLSD problem into a much richer decision environment, where setup, capacity, and workforce constraints interact dynamically across the planning horizon.

Nevertheless, solving large-scale CLSD instances is particularly challenging due to the exponential growth in decision variables, especially when incorporating setup carryovers and crossover constraints [10]. Traditional exact optimisation methods and commercial solvers often falter under the weight of these complexities. Consequently, the research community has increasingly adopted heuristic and metaheuristic approaches. Decomposition heuristics [11, 12], fix-and-relax methods [13, 10], and rounding-based strategies [14] have shown promise. Notably, fix-and-optimise algorithms have demonstrated strong performance in large-scale scenarios by breaking down the problem and iteratively refining decision subsets [15].

Building on this foundation, we propose a CGNS algorithm that addresses the full spectrum of CLSD complexities—including secondary resource management, setup carryovers, and crossovers—in a unified framework. Our approach introduces a novel column-based formulation that effectively reduces the combinatorial space and supports a scalable pricing procedure guided by neighbourhood heuristics. Furthermore, we propose

a neural network inserted into the CGNS to improve quality and solution time for the column generation phase, which has also been shown to improve the algorithm's results further.

The primary contributions of this study are as follows:

(i) **Innovative Column-Based Formulation:** We propose a compact column structure to mitigate the exponential growth of variables and constraints in CLSD models. This structure simplifies the integration of side constraints and supports efficient column generation.

(ii) **CGNS Algorithm Development:** The CGNS algorithm iteratively solves a RMP and Restricted linear problem (RLP), using dual prices from the RLP to guide the generation of promising production patterns via neighbourhood search. This enhances classical column generation's scalability and solution quality.

(iii) **CGNSNN Algorithm Development:** The column selection procedures are replaced by a novel neural network structure trained on multiple instances to increase the column selection quality and improve the algorithm's overall results.

(iv) **Scalability to Industrial Instances:** The algorithm is validated on real-world data sets and demonstrates robust performance across large-scale cases. It consistently outperforms commercial solvers and benchmark heuristics, including the Hybrid decomposition fix and optimise (HDFO) and Set-Oriented Data-Driven Branching and Selection (SODBS) algorithms.

By efficiently integrating multiple industrial constraints into a unified column generation framework, our approach significantly improves on prior methods regarding scalability, flexibility, and solution quality. Experimental results confirm that the CGNS algorithm delivers superior performance within practical computation times and can serve

as a decision support tool for production environments characterised by high complexity and variability.

The remainder of this thesis is organised as follows. First, we explain in detail the problem studied. Next, we review the literature on CLSP, CLSD, and associated solution strategies. We then present the mathematical formulation of the exact and proposed models, followed by a detailed description of the CGNS and CGNSNN algorithms. Subsequently, we evaluate its performance using synthetic and real-world data, comparing it with benchmark algorithms. Finally, we conclude with a discussion of the results and practical implications.

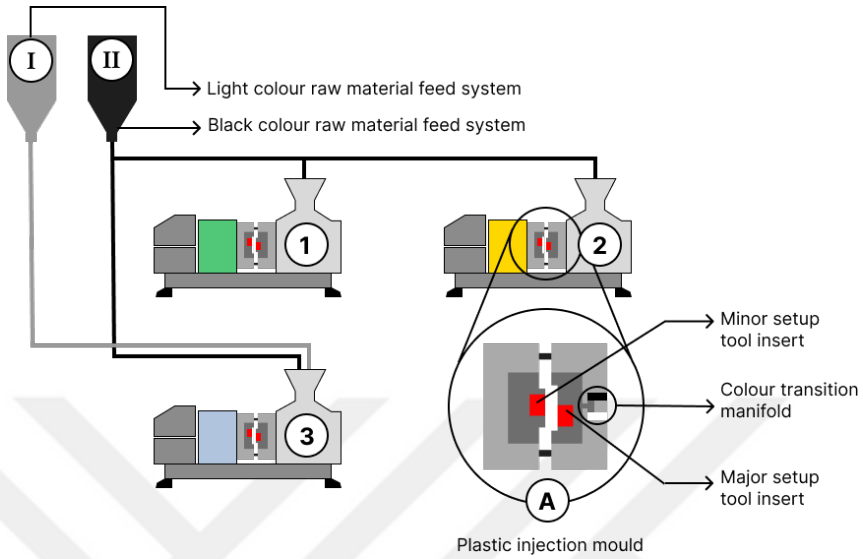
2. PROBLEM DEFINITION

This study addresses a highly complex and large-scale instance of the CLSD encountered in a real-world plastic injection manufacturing facility. The core objective is to minimise the total operational cost, which includes sequence-dependent setup costs, production costs, backlog penalties, and inventory holding costs. What distinguishes this case from conventional CLSD formulations is the considerable scale of data and the additional constraints related to mould-machine assignments, acting as secondary resource requirements.

The problem environment involves 91 parallel non-identical plastic injection machines and over 300 moulds, producing more than 300 distinct parts. Within a 14-day planning horizon, the system must respond to approximately 3,000 monthly final product orders, as determined by the downstream assembly process. This translates into over 10,000 semi-finished part SKUs and nearly three million parts produced per month, following a detailed bill-of-materials structure.

Figure 2.1 illustrates the high-level structure of the production environment. Production occurs via machine and mould capabilities, where mould compatibility is determined by tonnage, injection sleeve size, raw material type, material colour, and steam injection capability. For instance, mould-machine compatibility must ensure that the clamping force, dictated by machine tonnage, aligns with the mould size. At the same time, injection sleeves are matched to specific part geometries to prevent overheating and material degradation.

Figure 2.1: *Plastic Injection Factory Production Process Overview showing machines, moulds, feeding systems, and versioning inserts.*



The plant also includes a centralised raw material feeding system responsible for routing different types of raw materials to designated machine clusters. This system is visualised as zones I and II in Figure 2.1, and it plays a crucial role in efficiently distributing white and coloured raw materials. Switching from black to white raw material requires manifold cleaning due to colour contamination risks. This process may take up to 24 hours unless the mould includes a secondary manifold (as highlighted in the figure), circumvents this issue by isolating white material flows.

Certain parts require high surface gloss, which is achieved via steam injection. Only machines equipped with steam generators (also depicted in Figure 2.1) can handle these moulds. Additionally, mould versions can be adapted using technician-installed inserts, as shown in point A in the figure. While most mould duplications accommodate all part versions, some omit specific inserts, adding to the complexity of mould selection.

Sequence-dependent setups are a major contributor to scheduling difficulty. Mould

changes typically take around three hours, while switching inserts can range from 15 minutes to eight hours, depending on part complexity. Transitioning between raw material colours, especially from black to white, introduces further setup time, between three and 24 hours, again depending on the availability of a dedicated white manifold. These setup processes constrain machine availability and add to the cumulative cost structure.

Energy consumption, which scales with machine tonnage, constitutes a significant portion of the production cost. Machines with higher clamping forces consume more energy per cycle, and mould allocation to such machines must consider this trade-off. Part-specific cycle times, demand distributions, and accumulated setup delays dynamically influence backlog and inventory costs.

Explaining the critical aspects of the studied problem, the next section is reserved for the literature review of the Simultaneous Lot Sizing and Scheduling Problem.

3. LITERATURE REVIEW

This section makes a brief introduction to GLSP variants and dives into an extensive literature review on CLSD, with a particular emphasis on industrially relevant characteristics such as Sequence-dependent setups (SD), Secondary Resources (SR), Setup Carryovers (SCo), and Setup Crossovers (SCr). Given the combinatorial complexity introduced by these features, many real-world problems remain computationally intractable using classical exact approaches. Thus, various decomposition and heuristic methods have been proposed to handle such formulations. This section reviews significant contributions, identifying research trends and methodological gaps.

3.1 Mathematical model of the GLSP

We begin our literature review with the mathematical model of the GLSP proposed by [4] and represent different model variants based on this formulation. Finally, we will dive into the details of the specific CLSD formulation, which is most suitable for the studied problem.

Table 3.1 shows the details of the parameters used in the GLSP model, while Table 3.2 summarizes the decision variables. As detailed below, the mathematical formulation consists of an objective function and several constraints.

Equation 3.1 defines the objective function, which minimizes the total cost comprising setup costs, inventory holding costs, and standby costs for preserving the setup state. The production resource capacity is limited by Equation 3.2, ensuring that the total time spent on production, setup conservation, and changeovers does not exceed the available capacity in each macro period. Inventory balance is maintained through Equation 3.3, which updates the inventory level of each product at the end of each macro period based on the previous inventory, production quantity, and demand. Equation 3.4 guarantees that exactly one setup state is active during each micro period, corresponding

to a physical product or the neutral state. Equation 3.5 links the production quantities and standby times with the setup state, ensuring that production or standby occurs if the setup is active. Equation 3.6 modelled changeovers between setup states, where the variable z_{iks} indicates whether a transition from product i to product k occurs in a given micro period. Finally, Equation 3.7 enforces minimum lot size requirements, which ensures that if a product starts production, it satisfies its minimal batch size constraints.

The model provides an integrated lot-sizing and scheduling framework suitable for environments with sequence-dependent setups, finite production capacities, and inventory considerations.

Table 3.1: *Parameters of the GLSP Model.*

Symbol	Description
i, k	Product indices, $i, k = 0, 1, \ldots, K$, where 0 is the neutral state
s	Micro period index, $s = 1, \ldots, S$
t	Macro period index, $t = 1, \ldots, T$
S_t	Set of micro periods within macro period t
sc_{ik}	Setup cost from product i to k
h_{ck}	Holding cost per unit of product $k > 0$ per macro period
pc_k	Standby cost per time unit for preserving the setup of product k
a_k	Production time per unit of product k (with $a_0 = 1$)
st_{ik}	Setup time from product i to product k
C_t	Available capacity in macro period t (time)
I_{k0}	Initial inventory of product $k > 0$
d_{kt}	Demand of product k in macro period t
ω_{k0}	1 if the resource is set up for product k at start, 0 otherwise
$q_k^{\min}$	Minimum production quantity for product $k > 0$; for $k = 0$, minimum neutral time

Table 3.2: *Decision Variables of the general GLSP Model.*

Variable	Description
$q_{ks} \geq 0$	Production quantity of product $k > 0$ in micro period s (time in neutral state if $k = 0$)
$\tilde{q}_{ks} \geq 0$	Time preserving the setup state of product k in micro period s (with $\tilde{q}_{0s} = 0$)
$I_{kt} \geq 0$	Inventory of product $k > 0$ at the end of macro period t
$\omega_{ks} \in \{0, 1\}$	1 if the resource is set up for product k in micro period s
$z_{iks} \in \{0, 1\}$	1 if a changeover from product i to k occurs in micro period s

$$\min \sum_{s=1}^{S} \sum_{i=0}^{K} \sum_{k=0}^{K} sc_{ik} \cdot z_{iks} + \sum_{k=1}^{K} \sum_{t=1}^{T} h_{ck} \cdot I_{kt} + \sum_{k=0}^{K} \sum_{s=1}^{S} pc_k \cdot \tilde{q}_{ks} \tag{3.1}$$

$$\sum_{k=0}^{K} \sum_{s \in S_t} (a_k \cdot q_{ks} + \tilde{q}_{ks}) + \sum_{i=0}^{K} \sum_{k=0}^{K} \sum_{s \in S_t} st_{ik} \cdot z_{iks} = C_t \quad (\forall t \in T) \tag{3.2}$$

$$I_{kt} = I_{k,t-1} + \sum_{s \in S_t} q_{ks} - d_{kt} \quad (\forall k > 0, \forall t \in T) \tag{3.3}$$

$$\sum_{k=0}^{K} \omega_{ks} = 1 \quad (\forall s = 1, \ldots, S) \tag{3.4}$$

$$a_k \cdot q_{ks} + \tilde{q}_{ks} \leq C_t \cdot \omega_{ks} \quad (\forall k = 0, \ldots, K, \forall s \in S_t) \tag{3.5}$$

$$z_{iks} \geq \omega_{i,s-1} + \omega_{ks} - 1 \quad (\forall i = 0, \ldots, K, \forall k = 0, \ldots, K, \forall s = 1, \ldots, S) \tag{3.6}$$

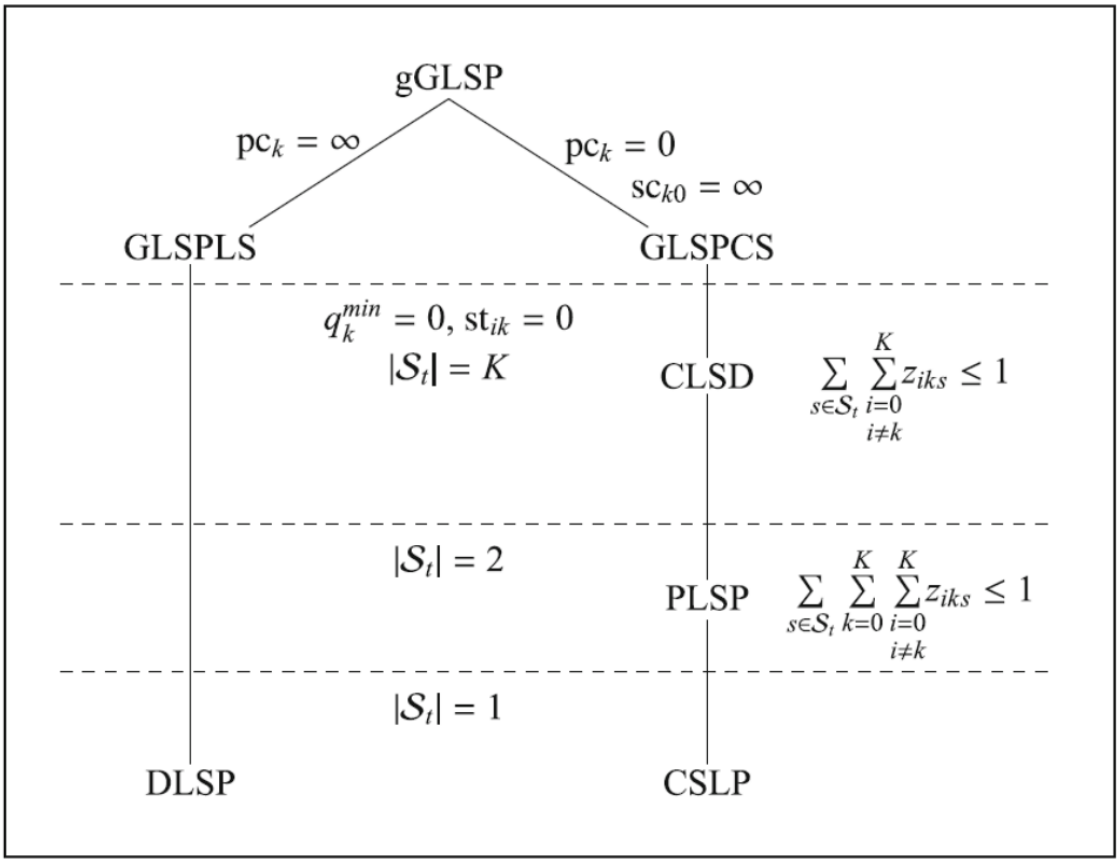
$$q_{ks} \geq q_k^{\min} (\omega_{ks} - \omega_{k,s-1}) \quad (\forall k = 1, \ldots, K, \forall s = 1, \ldots, S) \tag{3.7}$$

3.2 Classification of Lot Sizing and Scheduling Problems

Figure-3.1 represents a general overview of the lot sizing and scheduling problem. [1] classifies the problem in multiple subproblems first by categorizing them based on conservation of setups; General Lot-Sizing and Scheduling Problem with Loss of Setup Stages (GLSPLS) where the Standby costs for preserving the setup state of product k on the production resource (per time unit) is $pc_k = \infty$. On the other hand, GLSPCS is that the pc_k is set to zero, so idle periods can conserve the setup state inherited from the previous period.

The following classification is based on the quantity of micro periods present within macro periods. In the CLSD problem, the microperiod can be set to an arbitrary value K. Notably, the capacitated lot sizing and scheduling problem CLSP is one variant of CLSD where the setups are sequence-independent and are called big bucket models. In Proportional Lot-sizing and Scheduling Problem (PLSP), this value is fixed to two. Finally, the Continuous Lot-sizing and Scheduling Problem (CSLP) has no micro periods similar to the GLSPLS variant, the Discrete Lot-sizing and Scheduling Problem (DLSP).

Figure 3.1: *Classification Lot Sizing Scheduling Problem [1].*



The evolution of lot-sizing and scheduling research has seen a progressive shift from idealized formulations toward more realistic and complex settings. Notable early work focused on single-machine and small-bucket models. More recently, big-bucket formulations of CLSP and CLSD have become dominant due to their flexibility in modelling complex production environments over extended planning horizons. Several studies have extended traditional CLSP models by incorporating sequence dependency and resource constraints. For instance, [16] and [17] introduced part-mould-machine compatibility constraints in plastic injection settings, addressing SR and machine eligibility constraints.

Similarly, [18] formulated a CLSD with secondary tools as resource constraints and proposed an MILP model.

3.3 Literature Review on Big-Bucket Formulations

Although the primary focus of this study lies in solving large-scale instances of simultaneous lot-sizing and scheduling problems with sequence-dependent setups, it is critical to acknowledge relevant extensions found in the literature. For example, [16] investigates a plastic injection production setting involving moulds as secondary resources and explicitly models part-mould-machine interactions. Similar interactions are explored in [17] and [19]. Secondary resources, such as tools and setup operators, are further addressed in [18] and [20], respectively.

Despite these developments, existing formulations and solution strategies still require enhancement to address the challenges of large-scale, real-world applications. A significant trend in the literature is the preference for big-bucket formulations, particularly in sequence-dependent setups. Noteworthy contributions include studies employing the big-bucket CLSP and CLSD formulations. Nevertheless, some alternatives such as the PLSP [21, 22, 23] and the CSLP [24] utilise small-bucket formulations.

Various heuristics have been employed to tackle the inherent complexity of these problems, especially under sequence-dependent setups. These include Lagrangian-based and column generation-based approaches [25, 26, 24, 27], decomposition heuristics [10, 17], and fix-and-relax approaches [20, 28, 15]. Notable extensions of these strategies include the HDFO [13], SODBS [10], and rolling horizon algorithms [15].

Big-bucket CLSP and CLSD models offer scheduling flexibility by accommodating multiple products within macro periods—, a crucial advantage when managing sequence-dependent setups and dynamic demand profiles. For example, [29] addresses

setup carryovers in a single-machine context using a big-bucket formulation and introduces variable neighbourhood and tabu search heuristics. Similarly, [12] and [30] consider setup carryovers in single-machine and parallel identical machine settings with hierarchical and two-stage heuristics. In contrast, [13] applies a fix-and-optimise heuristic to a CLSD problem on parallel non-identical machines with setup carryovers. Furthermore, [31] formulates a CLSP model with setup carryovers and crossovers, omitting sequence dependence. In contrast, [20] examines practical extensions such as shelf-life constraints using fix-and-relax and fix-and-optimise heuristics.

To refine our scope, we categorise the literature based on machine configurations: Single Machine (SM), Parallel Identical Machines (PI), Parallel Non-identical Machines (PN), and Multi Level Machines (ML) systems. Focusing on PN systems due to their industrial relevance, we highlight six key works: [10, 32, 27, 13, 18, 24] Each addresses sequence-dependent setups and setup carryovers from distinct angles.

Among these, [32] and [18] lack scalable solution algorithms for large datasets. Conversely, [24] uses a CSLP small-bucket model with Lagrangian relaxation in a glass container application but suffers from scalability issues due to the extensive number of macro periods required. For benchmark comparison, we rely on [13] and [10], which employ big-bucket CLSD formulations and demonstrate robust performance using HDFO and SODBS heuristics.

Solving large-scale CLSD problems under big-bucket formulations is particularly challenging because of the inclusion of complex side constraints such as setup carryovers, crossovers, and secondary resources. Column generation methods offer a promising avenue by efficiently handling models with vast decision spaces. Nevertheless, the pricing phase remains a bottleneck, especially with large instances. To mitigate this, neighbourhood search heuristics have been integrated into column generation frameworks, enabling targeted exploration of promising solution regions.

Our approach builds upon this synergy by proposing a novel pattern-based column generation algorithm. It combines adaptive neighbourhood search to construct candidate columns with column generation to evaluate reduced costs, resulting in a compact representation of production sequences and a significant reduction in decision variables.

Table 3.3: *Literature Review Summary.*

	Reference	SD[1]	SR[2]	SCo[3]	SCr[4]	Machine	Solution method
CLSP	[16]		✓			PN	MIP Solver sequential heuristic
	[10]	✓		✓		PN	Data-driven decomposition heuristic
	[32]	✓		✓		PN	MIP Solver
	[27]	✓		✓		PN	Lagrangian-simulated annealing
	[13]	✓		✓		PN	Fix-and-relax heuristics
	[18]	✓	✓	✓		PN	MIP Solver
	[24]	✓	✓	✓		PN	Lagrangian relaxation
	[11]	✓	✓			PN	Decomposition heuristic
	[33]					PI	Genetic programming
	[26]					PI	Lagrangian heuristics
	[20]	✓	✓	✓		PI	Fix-and-relax heuristics
	[30]	✓		✓		PI	MIP heuristic
	[31]			✓	✓	SM	MIP Solver
	[12]	✓		✓		SM	Decomposition heuristic
	[29]	✓		✓		SM	Variable neighborhood–Tabu search
	[34]					SM	Branch and price
	[35]			✓		ML	Kernel search
	[36]	✓		✓		ML	MIP Solver
	[28]			✓		ML	Fix and optimise heuristic
PLSP	[37]			✓		SM	Valid inequalities
	[22]		✓	✓		SM	MIP Solver
	[23]	✓		✓		SM	MIP Solver

3.4 Machine Learning Approaches for Solving MILPs

MILP models form the foundation of many combinatorial optimization problems. Traditional exact methods, such as branch-and-bound and branch-and-cut, have been the dominant techniques for solving MILPs; however, these methods often struggle to scale with increasing problem complexity. Recently, machine learning approaches, particularly Reinforcement Learning (RL), deep learning, and Graph Neural Network (GNN)s have been explored as promising avenues for enhancing or replacing parts of classical MILP solution strategies.

This section reviews the literature on integrating machine learning into MILP solving, highlighting advancements in reinforcement learning, robust optimization, neural-enhanced decomposition algorithms, and graph-based learning approaches.

Several studies have focused on reinforcement learning to improve optimization processes. [38] provides a comprehensive survey on using RL for combinatorial optimization, modelling MILP solving as a Markov decision process. [39] explores deep reinforcement learning applications in transportation network optimization. [40] proposes the ECO-DQN framework, allowing iterative refinement during RL solution construction. Additionally, [41] advocates combining deep reinforcement learning with constraint programming to leverage dynamic programming structures. [42] presents a reinforcement learning framework to enhance the selection of cutting planes in integer programming.

In robust optimization, [43] introduces a data-driven method using deep neural networks to define uncertainty sets, improving robust MILP formulations.

Regarding decomposition methods, [44] uses neural predictors to accelerate the pricing phase in branch-and-price algorithms. [45] applies graph neural networks to drive columns to preprocess large-scale problems, reducing computation times. Furthermore, [46] propose a machine learning-based column selection technique using GNN and Graph Convolution Network (GCN) to improve the efficiency of column generation processes.

For heuristic search enhancements, [47] develops a Graph Reinforcement Learning approach to optimize operator selection in adaptive large neighbourhood search methods.

Work on graph-based learning includes [48], who develop a Local and Global Information Aware Graph Neural Network (LG-GNN) that improves feature aggregation in graph structures. Similarly, [49] and Gasse et al. [50] examine the potential of GNNs for direct reasoning and solution construction in combinatorial optimization problems.

In meta-learning, Chen et al. [51] introduce a recurrent neural network-based optimizer for black-box optimization tasks, illustrating how meta-learned strategies can generalize across problems.

Specific enhancements to branch-and-bound processes are also evident. [52] propose Neural Diving and Neural Branching to guide search strategies efficiently. [53] employ learning-to-rank methods to improve branching decisions, while [54] explore tree parameterization for better generalization. [55] design a hybrid GNN-MLP architecture enabling CPU-efficient branching.

The surveyed literature demonstrates a rapidly growing interest in applying machine learning techniques to MILP solving. Key advancements include modelling solution construction and improvement as sequential decision processes via reinforcement learning, using deep neural networks to construct data-driven uncertainty sets for robust MILP formulations, enhancing traditional decomposition techniques such as branch-and-price with neural predictors, employing graph neural networks to capture complex structural dependencies in MILP models, leveraging meta-learning and hybrid architectures for solver integration, and hybridizing constraint programming with reinforcement learning to combine completeness and efficiency.

Despite promising results, challenges remain, including generalization across diverse MILP instances, computational overhead during training, and integration into industrial solvers. Future research is expected to explore hybrid models combining traditional optimization expertise with learned strategies to balance reliability and adaptability.

Table 3.4 presents a structured overview of the main advances categorized by their core innovation to summarise the contemporary contributions discussed above.

Table 3.4: *Summary of Contributions from Key Papers.*

Paper	Year	Contribution
[48]	2024	Local-global GNNs for feature enhancement in graph structures.
[44]	2024	Neural predictors for dual values in branch-and-price.
[47]	2024	Graph RL for Adaptive Large Neighbourhood Search (ALNS) operator selection.
[49]	2023	GNN review for combinatorial optimization tasks.
[43]	2023	Data-driven uncertainty sets for robust optimization.
[41]	2021	Hybridization of Deep Reinforcement Learning (DRL) and Constraint Programming (CP) using dynamic programming.
[46]	2021	ML-based column selection for improving column generation.
[54]	2021	Parameterizing B&B trees to enhance branching generalization.
[38]	2021	Survey of RL for combinatorial optimization, MILP framing as Markov Decision Process (MDP).
[52]	2021	Neural Diving and Neural Branching integrated into MIP solvers.
[40]	2020	Exploratory Combinatorial Optimization - Deep Q Network (ECO-DQN) enabling exploration during RL combinatorial search.
[42]	2020	RL-based cut selection in cutting plane methods.
[55]	2020	Hybrid GNN-MLP architectures for CPU-efficient branching.
[50]	2019	Exact combinatorial optimization with Graph Convolution Neural Network (GCNN)s for branching.
[51]	2017	Meta-learned optimizers for black-box optimization.
[53]	2016	Learning-to-rank branching strategies for branch-and-bound.

The following section will explain the gaps and our motivation to handle large-scale instances, including mathematical models, solution methods, and possible machine learning adaptations to solve the CLSD problems.

3.5 Gaps and Motivation

Despite increasing efforts to incorporate realistic constraints, few studies adequately address large-scale instances involving parallel non-identical machines. Among these, [24] employs a small-bucket model with secondary resources, but its scalability is limited due to the proliferation of periods. Conversely, [13] and [10] provide scalable big-bucket CLSD formulations using fix-and-optimise heuristics, making them suitable benchmarks.

Column generation is a powerful technique for solving large-scale linear relaxations by decomposing the problem into a master and a pricing component. However, its use in CLSD remains limited. For example, [25] integrates it into a Lagrangian CLSD to reduce problem size, but pricing remains computationally intensive. We propose a hybrid framework to address this challenge that embeds neighbourhood search into the pricing phase. This targeted approach avoids exhaustive enumeration and leverages problem-specific structures for more efficient pattern generation.

Furthermore, integration of Neural Network (NN)s to column generation and neighbourhood search heuristics can be seen in recent publications on the integration of machine learning with combinatorial optimisation such as in [44] and [46] using machine learning algorithms in column selection or [47] using machine learning in neighbourhood search.

Unlike rolling horizon fix-and-optimise methods like HDFO, which iteratively fix decisions over time, our algorithm generates complete production sequences as column patterns. These patterns correspond to feasible schedules over macro periods and serve

as units in a column-generation framework. This formulation enhances scalability and provides a solid foundation for hybridisation with learning-based strategies, which is explored in the subsequent chapters of this thesis.

While several methods exist to handle sequence-dependent setups, secondary resources, and setup carryovers individually, few approaches efficiently integrate these features in large-scale PN systems. Fix-and-optimise heuristics dominate the literature due to their scalability but often neglect deeper structural exploitation. Column generation offers a promising alternative; however, its adoption in CLSD is hindered by pricing complexity.

This study fills this gap by proposing a hybrid column generation heuristic. Our method integrates a pattern-based neighbourhood search with structural decomposition, significantly reducing pricing subproblem solution time while maintaining scalability. The result is a robust and efficient solution framework tailored to industrial-scale CLSD problems. Moreover, we propose a neural network structure to increase solution quality and reduce column generation time, yielding superior results to the proposed CGNS heuristic.

The next chapter presents the exact mathematical model and proposed formulation with solution methods.

4. SOLUTION ALGORITHMS

This chapter is dedicated to the mathematical formulations of the problem under study and the proposed solution algorithms. Initially, the exact formulation of the CLSD utilized by the benchmark algorithms will be presented. Following that, the proposed CGNS algorithm, along with its mathematical formulation and solution techniques, will be introduced. Finally, the discussion will cover the CGNSNN algorithm and the proposed NN structures.

4.1 Exact Formulation for CLSD

This section introduces the exact mathematical model used by [13] and [10]. The introduced model considers sequence-dependent setups, setup carryovers, and backlogs. However, the model does not consider setup crossovers. Due to this reason, our proposed pattern-based formulation will be extended by additional constraints to match the benchmark algorithms proposed by [13] and [10].

Table 4.1: *Indices.*

Symbol	Description
$i, j \in \mathcal{I}$	Set of items (products)
$l \in \mathcal{L}$	Set of machines
$t \in \mathcal{T}$	Set of time periods

Table 4.2: *Model Parameters.*

Parameter	Description
D_{it}	Demand for item i at time period t
CT_i	Production time required per unit of item i
ST_{ij}	Setup time required to switch from item i to item j
HC_i	Inventory holding cost per unit of item i
BC_i	Backlog penalty cost per unit of item i
PC_l	Production cost coefficient for machine l
SC	Setup cost coefficient (per unit setup time)
T^{cap}	Available time capacity per machine per period
Z_l^0	Initial setup state on machine l (i.e., part assigned at $t = 0$)

Table 4.3: *Decision Variables.*

Symbol	Description
$s_{it} \geq 0$	Inventory level of item i at time t
$b_{it} \geq 0$	Backlog of item i at time t
$q_{ilt} \geq 0$	Quantity of item i produced on machine l at time t
$z_{ilt} \in \{0, 1\}$	1 if machine l is setup for item i at time t
$y_{ijlt} \in \{0, 1\}$	1 if setup from item i to j occurs on machine l at time t
$S_{ilt} \in \mathbb{Z}_+$	Sequencing helper variable

$$\begin{aligned}\min \sum_{i\in I}\sum_{t\in T} HC_i \cdot s_{it} + BC_i \cdot b_{it} \\ + \sum_{i\in I}\sum_{l\in L}\sum_{t\in T}\left(PC_l \cdot CT_i \cdot q_{ilt} + \sum_{j\in I} ST_{ij} \cdot SC \cdot y_{ijlt}\right)\end{aligned} \tag{4.1}$$

$$s_{it-1} + \sum_{l\in L} q_{ilt} - b_{it-1} + b_{it} - s_{it} = D_{it} \quad (\forall i \in I, \forall t \in T, t \geq 1) \tag{4.2}$$

$$\sum_{i \in I} CT_i \cdot q_{ilt} + \sum_{i \in I} \sum_{j \in I} ST_{ij} \cdot y_{ijlt} \leq T^{\text{cap}} \quad (\forall l \in L, \forall t \in T) \tag{4.3}$$

$$q_{ilt} - \sum_{\substack{j \in I \\ j \neq i}} \text{term3}_{ilt} \cdot y_{jilt} - \text{term3}_{ilt} \cdot z_{il,t-1} \leq 0 \quad (\forall i \in I, \forall l \in L, \forall t \in T) \tag{4.4}$$

$$\sum_{i \in I} z_{ilt} = 1 \quad (\forall l \in L, \forall t \in T) \tag{4.5}$$

$$\sum_{\substack{j \in I \\ j \neq i}} (y_{ijlt} - y_{jilt}) + z_{ilt} - z_{il,t-1} = 0 \quad (\forall i \in I, \forall l \in L, \forall t \in T) \tag{4.6}$$

$$S_{jlt} - S_{ilt} - I^{\max} \cdot y_{ijlt} \geq 1 - I^{\max} \quad (\forall i \in I, \forall j \in I, \forall l \in L, \forall t \in T) \tag{4.7}$$

$$s_{i0} = 0 \quad (\forall i \in I) \tag{4.8}$$

$$b_{i0} = 0 \quad (\forall i \in I) \tag{4.9}$$

$$z_{il0} = \begin{cases} 1 & \text{if } Z_l^0 = i \\ 0 & \text{otherwise} \end{cases} \quad (\forall i \in I, \forall l \in L) \tag{4.10}$$

The exact formulation from our benchmark [13] represents a simultaneous lot-sizing and scheduling problem with sequence-dependent setups on multiple machines and allows setup carryovers. The goal is to fulfil demand requirements while minimising total operational costs. Equation 4.1 defines the objective function, which minimises the total cost, composed of inventory holding, backlog, production, and setup costs. Equation 4.2 ensures inventory balance, guaranteeing that inventory and backlog flows meet demand

over time. Equation 4.3 imposes machine time capacity constraints, ensuring that total processing and setup times do not exceed the available time per machine. Equation 4.4 enforces setup requirements, allowing production only if a proper setup transition or existing setup is present. Equation 4.5 ensures that each machine is set up for only one item at a period. Equation 4.6 models the setup flow conservation, maintaining consistency in setup transitions across periods. Equation 4.7 introduces cycle elimination constraints using sequencing variables to prevent circular setup transitions. Equations 4.8, 4.9, and 4.10 specify the initial conditions, with zero initial inventory and backlog and a machine setup state defined by the parameter Z_l^0. This model integrates production, inventory, and sequence-dependent setup decisions to optimise the scheduling of multiple products across multiple machines and periods.

4.2 CGNS Algorithm

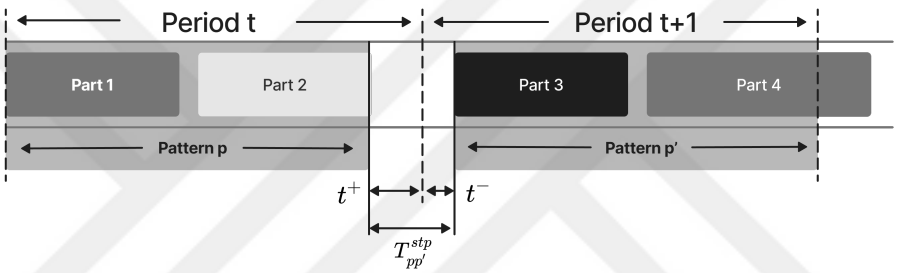
This section outlines the proposed CLSD formulation that integrates setup carryovers and secondary resources and introduces the CGNS algorithm, designed to solve large instances of such problems effectively.

A critical aspect of our approach is the novel pattern structure we propose. This structure organises patterns as lists of parts that preserve essential sequence information while also capturing total sequence-dependent setup times. Unlike traditional methodologies that rely on rigid column definitions, which often separate scheduling and sequencing decisions, our pattern structure addresses the inherent complexity of the CLSD problem. By integrating sequencing and scheduling into a unified framework, we greatly minimise the excessive constraints and decision variables that often arise in existing formulations. This approach improves the algorithm's efficiency and ensures that the essential relationships among parts, setups, and sequences are preserved throughout optimisation.

Figure 4.1 visualises the relationships among parts, patterns, and columns used

in the problem. Specifically, Pattern p represents the sequence of Part 1 and Part 2. A column denotes the assignment of Pattern p to Machine l during Period t. The setup crossover times, denoted as t^+ and t^-, represent the sequence-dependent setups between two patterns across consecutive periods. The t^+ variable signifies the setup crossover time occurring in the first period, while the t^- variable represents the setup crossover time appearing in the subsequent period.

Figure 4.1: *Demonstration of Patterns, Columns, and Sequences.*



The algorithm follows a structured flow, as illustrated in Figure 4.2. It begins with an initialisation phase, where initial sets of patterns are generated. In this phase, single-part tuples represent a simplified version of the problem, establishing a solid foundation for subsequent steps.

Following the initialisation, the algorithm solves the RMP based on the generated initial tuples. This solution is the cornerstone for further optimisation, allowing the algorithm to refine its approach.

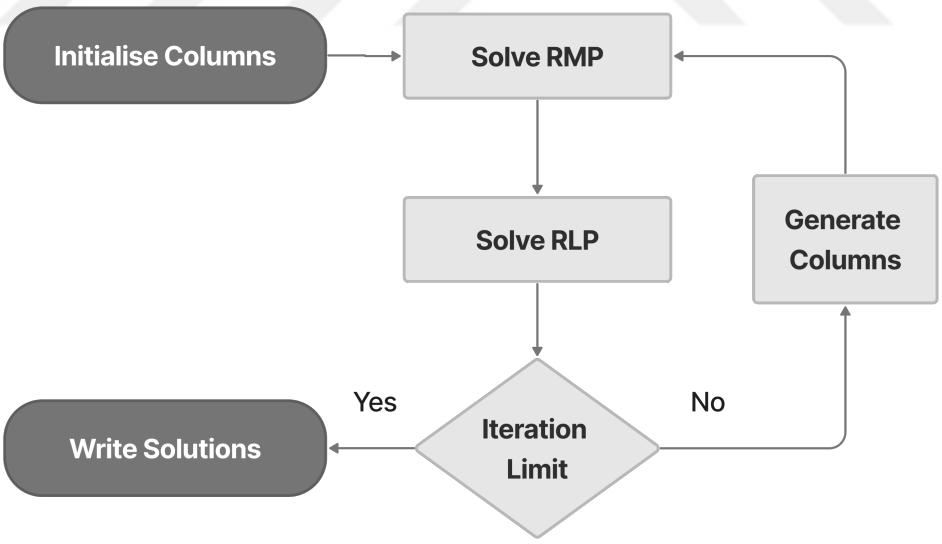
Once the RMP has been solved, the algorithm moves on to the RLP. The algorithm extracts dual information from the last incumbent solution during this stage. This dual information is crucial, as it guides the generation of new columns more closely aligned with the current solution.

The algorithm then systematically searches neighbouring columns through well-defined search procedures using the extracted dual information. These procedures facilitate the generation of a new column pool closely related to the incumbent solution, enhancing the algorithm's ability to explore the solution space effectively.

The algorithm iteratively cycles through this process, returning to solve the RMP with the newly generated columns. This iterative process continues until a predefined solution time is reached, allowing for continuous refinement and improvement.

Upon reaching the designated time limit, the algorithm halts and reports the results, summarising the solutions obtained throughout the iterative process. Further details on the RMP, RLP, and the search procedures will be elaborated in the subsequent subsections.

Figure 4.2: *Algorithm Flowchart.*



4.2.1 Restricted master problem

The CLSD model presented in this section aims to generate production schedules for the next two weeks, covering allocating machines and secondary resources. The model's objective is to minimise the total cost of production, backlog, inventory, and setup while ensuring that the solution meets the physical constraints imposed by the production plant and customer demand. The model decides on machine-part allocations, part sequences, mould machine assignments, setup crossover partitioning, and production quantities.

Table 4.4: *RMP Model Parameters.*

Parameter	Description
L	Set of machines, where l is the machine index such that $l = 1, \ldots, L^{\max}$
M	Set of moulds, where m is the mould index such that $m = 1, \ldots, M^{\max}$
I	Set of parts, where i is the part index such that $i = 1, \ldots, I^{\max}$
P	Set of patterns, where p is the pattern index such that $p = 1, \ldots, P^{\max}$
T	Set of periods, where t is the period index such that $t = 1, \ldots, T^{\max}$
P_i	Set of patterns that contain part i
I_p	Set of parts that are members of pattern p
L_i	Set of machines that are compatible with part i
C^{stp}	Setup cost per unit time
C_l^{prod}	Production cost of machine l
C_i^{back}	Backlogging cost of part i
C_i^{inv}	Inventory cost of part i
C^{wf}	Cost for free slots (unused machine-period pairs)
D_{it}	Demand of part i at period t
Π_i	Total demand of part i
T_p^{stp}	Total setup time of pattern p
$T_{pp'}^{stp}$	Setup time from pattern p to pattern p'
T^{cap}	Available production capacity per macro-period (in seconds)
T_i^{cyc}	Cycle time for part i
T_{lt}^{co}	Setup crossover time from period $t-1$ to t on machine l
Γ_{im}	Compatibility matrix: 1 if mould m can produce part i, 0 otherwise
Λ_{ml}	Compatibility matrix: 1 if mould m is compatible with machine l, 0 otherwise
Ω_{il}	Compatibility matrix: 1 if part i is compatible with machine l, 0 otherwise
B_i^0	Initial backlog of part i
S_i^0	Initial inventory of part i
A_{lt}	Pattern index assigned to machine l and period t in the last incumbent solution
$Q_{pp'}$	Setup crossover cost from pattern p to p'

The objective of the RMP is to minimise inventory holding, backlogging, production, and setup costs, including the setup crossovers. The RMP involves binary decision variables such as pattern assignment variables, setup carryover-crossover binary variables that track consecutive pattern assignments of the machines, and binary variables that penalise periods where pattern allocations are missing. Linear decision variables for production, inventory, and backlog are also included in the mathematical model, following the existing literature. Additionally, the proposed model incorporates sequence-dependent setup crossover times between consecutive periods.

A restricted number of columns generated throughout the algorithm are introduced into the RMP to enhance the existing solutions from the previous iteration. The RMP is an integer programming model with binary pattern assignment decision variables. Table 4.4 and Table 4.5 present this study's parameters and decision variables.

Table 4.5: *RMP Model Decision Variables.*

Variable	Description
$\alpha_{plt} \in \{0,1\}$	1 if machine l uses pattern p at period t
$\beta_{pp'lt} \in \{0,1\}$	1 if machine l uses pattern p at period t and pattern p' at period $t+1$
$w_{ml} \in \{0,1\}$	1 if mould m is assigned to machine l
$w^f_{lt} \in \{0,1\}$	1 if no pattern (column) is assigned to machine l at period t
$\overline{\alpha}_{plt} \in [0,1]$	Linearised (relaxed) form of α_{plt}
$\overline{w}_{ml} \in [0,1]$	Linearised (relaxed) form of w_{ml}
$\overline{w}^f_{lt} \in [0,1]$	Linearised (relaxed) form of w^f_{lt}
$\overline{\beta}_{pp'lt} \in [0,1]$	Linearised (relaxed) form of $\beta_{pp'lt}$
$q_{ilt} \in [0,1]$	Fraction of total demand of part i produced on machine l at period t
$s_{it} \in \mathbb{N}^+$	Inventory level of part i at period t
$b_{it} \in \mathbb{N}^+$	Backlog level of part i at period t
$t^-_{lt} \in \mathbb{N}^+$	Setup crossover time at the beginning of period t on machine l
$t^+_{lt} \in \mathbb{N}^+$	Setup crossover time at the end of period t on machine l
μ^5_{ilt}	Dual variable associated with constraint Eq-4.15
μ^6_{lt}	Dual variable associated with constraint Eq-4.16
μ^7_{lt}	Dual variable associated with constraint Eq-4.17
μ^9_{ilt}	Dual variable associated with constraint Eq-4.19

$$\begin{aligned}\min \sum_{i\in I}\sum_{t\in T} C^{inv}_i s_{it} + \sum_{i\in I}\sum_{t\in T} C^{back}_i b_{it} + \sum_{i\in I}\sum_{l\in L}\sum_{t\in T} C^{prod}_l T^{cyc}_i \Pi_i q_{ilt} \\ + \sum_{p\in P}\sum_{l\in L}\sum_{t\in T} C^{stp} T^{stp}_p \alpha_{plt} + \sum_{l\in L}\sum_{t\in T} C^{stp}(t^+_{lt} + t^-_{lt} + T^{cap} w^f_{lt})\end{aligned} \tag{4.11}$$

$$s_{it-1} + b_{it} + \sum_{l\in L} \Pi_i q_{ilt} - s_{it} - b_{it-1} = D_{it} \quad (\forall i \in I; \forall t \in T) \tag{4.12}$$

$$b_{i0} = B^0_i \qquad (\forall i \in I) \tag{4.13}$$

$$s_{i0} = S^0_i \qquad (\forall i \in I) \tag{4.14}$$

$$\sum_{p \in P_i} \alpha_{plt} - q_{ilt} \geq 0 \qquad (\forall i \in I; \forall l \in L; \forall t \in T) \tag{4.15}$$

$$T^{cap} - \sum_{i \in I} T_i^{cyc} \Pi_i q_{ilt} - \sum_{p \in P} T_p^{st} \alpha_{plt} - t_{lt}^{+} - t_{lt+1}^{-} \geq 0 \qquad (\forall l \in L; \forall t \in T) \tag{4.16}$$

$$\sum_{p \in P} \alpha_{plt} + w_{lt}^{f} = 1 \qquad (\forall l \in L; \forall t \in T) \tag{4.17}$$

$$\sum_{l \in L} w_{ml} \leq 1 \qquad (\forall m \in M) \tag{4.18}$$

$$\sum_{m \in M | \Gamma_{im} = 1} w_{ml} - \alpha_{plt} \geq 0 \qquad (p \in P; i \in I_p; l \in L; t \in T) \tag{4.19}$$

$$w_{ml} \leq \Lambda_{ml} \qquad (\forall m \in M; \forall l \in L) \tag{4.20}$$

$$\sum_{p \in P} \sum_{p' \in P} \beta_{pp'lt} T_{pp'}^{stp} = t_{lt}^{+} + t_{lt+1}^{-} \qquad (\forall l \in L; \forall t \in T | T^{max}) \tag{4.21}$$

$$\beta_{pp'lt} - \alpha_{plt} \leq 0 \qquad (\forall p \in P; \forall p' \in P; \forall l \in L; \forall t \in T | T^{max}) \tag{4.22}$$

$$\beta_{pp'lt} - \alpha_{p'lt+1} \leq 0 \qquad (\forall p \in P; \forall p' \in P; \forall l \in L; \forall t \in T | T^{max}) \tag{4.23}$$

$$\beta_{pp'lt} - \alpha_{plt} - \alpha_{p'lt+1} \geq -1 \qquad (\forall p \in P; \forall p' \in P; \forall l \in L; \forall t \in T | T^{max}) \tag{4.24}$$

The objective function (4.11) minimises the total inventory, production, backlogging, and setup costs, including the setup crossover costs. Equalities (4.12) define the

inventory balance constraint. Equalities (4.13) define initial backlogged quantities of the parts. Equalities (4.14) define the initial inventory quantities of the parts. Inequalities (4.15) ensure that if a chosen setup pattern does not include part *i*, then part *i* cannot be produced. Inequalities (4.16) guarantee that production plans cannot exceed available capacity. Equalities (4.17) prevent multiple choices of patterns on a machine *l* at a period *t*. Inequalities (4.18) assign moulds to only one machine throughout the planning horizon, which simplifies the problem and does not conflict with the current practice used by the company. Inequalities (4.19) ensure a pattern *p* containing part *i* can be assigned to a machine *l* if there is a compatible mould *m* assigned to machine *l* as well. Inequalities (4.20) prevent mould *m* from being assigned to an incompatible machine *l*. Equalities (4.21) set the setup crossover variables to the total setup between two consecutive patterns. Inequalities (4.22), (4.23), and (4.24) make sure that $\beta_{pp'lt}$ variable takes the value of 1 if α_{plt} and $\alpha_{p'lt+1}$ take the value of 1 at periods *t* and *t+1* respectively.

4.2.2 Restricted linear problem

The restricted linear problem is the linearised form of the RMP, where we fix the binary decision variables to the latest incumbent solution values. The problem is solved by gaining dual information on the binary columns and using this information to search for new promising columns for subsequent iterations. First, we introduce linear decision variables associated with the binary variables to gather dual information from the RMP. Later, we fix those linear variables to the latest incumbent RMP values. Finally, the dual information is collected, solving the resulting restricted linear problem. Below equalities (4.25), (4.26), (4.27), and (4.28) fix the linearised $\overline{\alpha}_{plt}$, $\overline{\beta}_{pp'lt}$, $\overline{w}_{ml}$ and $\overline{w}^{f}_{lt}$ to the latest RMP solutions α^{*}_{plt}, $\beta^{*}_{pp'lt}$, w^{*}_{ml} and w^{f*}_{lt} respectively.

$$\overline{\alpha}_{plt} = \alpha^{*}_{plt} \qquad (\forall p \in P; \forall l \in L; \forall t \in T) \tag{4.25}$$

$$\overline{\beta}_{pp'lt} = \beta^*_{pp'lt} \qquad (\forall p \in P; \forall p' \in P; \forall l \in L; \forall t \in T | T^{max}) \tag{4.26}$$

$$\overline{w}_{ml} = w^*_{ml} \qquad (\forall m \in M; \forall l \in L) \tag{4.27}$$

$$\overline{w}^f_{lt} = w^{f*}_{lt} \qquad (\forall l \in L; \forall t \in T) \tag{4.28}$$

4.2.3 Column search heuristics

The column search heuristics aim to discover neighbouring solutions to the latest incumbent and incorporate promising columns into a new column pool. The search process involves various procedures generating diverse columns, such as last solution, carryover, backlogged part, random improvement, injection, and best position columns. The algorithm utilises the dual information of the RLP to assess the generated columns, except for the procedures involving the last solution and carryover columns. We include these eligible columns in the column pool for the subsequent iteration.

The columns are evaluated based on their reduced costs within each iteration of the search procedure. The calculation of these reduced costs is explained below:

$$c^{\alpha}_{plt} = -\sum_{i \in I | i \in p} \mu^5_{ilt} + T^{stp}_p \mu^6_{lt} + \mu^7_{it} + \sum_{i \in I | i \in p} \mu^9_{ilt} + Q_p \tag{4.29}$$

$$Q_p = Q_{pp'} + Q_{p''p} \tag{4.30}$$

$$Q_{pp'} = T^{stp}_{pp'}(Min(\mu^6_{lt}, \mu^6_{lt+1}) + C^{stp}) \tag{4.31}$$

$$Q_{p''p} = T^{stp}_{p''p}(Min(\mu^6_{lt-1}, \mu^6_{lt}) + C^{stp}) \tag{4.32}$$

Equations (4.29), (4.30), (4.31), and (4.32) are used to calculate the reduced costs of the columns. μ^5_{ilt}, μ^6_{lt}, μ^7_{it} and μ^9_{ilt} are the reduced costs of inequalities (4.15), (4.16), and (4.19) and equalities (4.17). Adding a new column reduces the capacity due to the setup crossover times. The capacity loss cost of the machines could be calculated as the minimum dual costs of capacity inequalities (4.16). The Q_p term calculates the cost of

losing capacity due to the setup crossovers and setups. This calculation is conducted to set up crossovers between the previous and the next period's incumbent columns.

Six distinct sequentially operating algorithms generate a predetermined number of promising columns. Each of these proposed algorithms explores various regions within the solution space.

Algorithm 1 utilises the column pool from the RMP and generates neighbouring columns for subsequent iterations. Within the algorithm, the notation $< plt >$ represents a tuple, where *p* stands for the pattern index, *l* denotes the machine, and *t* represents the period. The set Φ' indicates the column pool, comprising these tuples derived from the latest iteration's solution. Similarly, Φ refers to the columns ($< plt >$ tuples) generated in the current search iteration.

Algorithm 1 Generate Last Solution Columns.

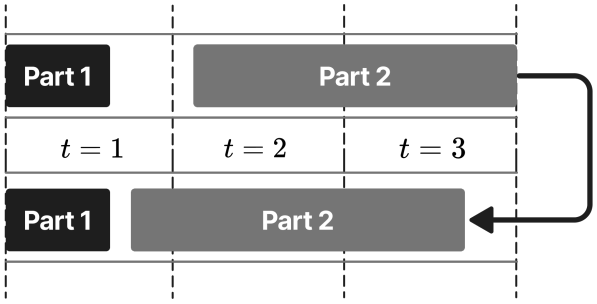
Input: Previously generated columns Φ', planning window n, time horizon $T^{\max}$

Output: Column set Φ updated with relevant patterns

$\Phi \leftarrow \{\}$
for each $\alpha_{plt'}$ in Φ' **do**
  if $\alpha_{plt'} = 1$ **then**
    for $t = \max(t' - n,\ 1)$ to $\min(t' + n,\ T^{\max})$ **do**
      Add $\langle p,\ l,\ t \rangle$ to Φ
    end for
  end if
end for

Each iteration, Φ is refreshed to encompass the incumbent solution columns from the preceding iteration. This is achieved by appending the most recent incumbent solution to Φ, spanning a duration of n periods before and after the current solution. This strategy enables the algorithm to commence from the latest solution and systematically explore

Figure 4.3: *Setup Carryover Column Generation.*



the incumbent plan across the planning horizon by navigating it forward and backward. In the numerical study, a value of $n = 5$ was selected following preliminary analyses.

We generate setup carryover columns for each machine *l* to create a continuous-time representation of the model. When a part carryover occurs between consecutive periods of a machine, new columns are produced by connecting the sequential patterns of the columns. The initial part of the subsequent column is inserted as the final part of the pattern in the first column. Similarly, the last part of the first successive column is included as the initial part in the pattern of the next consecutive column. Figure 4.3 illustrates this process.Algorithm 2 generates setup carryover columns for each set of incumbent solutions over consecutive periods. The final part of pattern *p* used at period *t*-1 is appended as the initial part of pattern p' used at period *t*. Conversely, the initial part of pattern p' used at period *t* is added as the final part to the pattern p employed at period *t*-1. The algorithm uses the *Append* operator to append the second input to the end of the first input.

Algorithm 2 Generate Carryover Columns.

for each l in L **do**
  for $t = 2$ to $T_{\max}$ **do**
    $p^1 \leftarrow A_{l,t-1}$, $p^2 \leftarrow A_{l,t}$
    iFirst = last part of p^1, iLast = first part of p^2
    $p^1 \leftarrow$ *Append*$(p^1,$ iLast$)$, $p^2 \leftarrow$ *Append*$($iFirst$, p^2)$
    Add $\langle p^1 l(t-1)\rangle$, $\langle p^2 lt\rangle$ to Φ
  end for
end for

Algorithm 3 searches for the suitable positions of the backlogged parts within the latest incumbent solution. In the algorithm's initial stage, backlogged parts are identified and aggregated into a pool named B^{list}, which essentially functions as a list of these parts. A finite number of iterations is then carried out, each involving a random selection of a backlogged part and a compatible machine. During each iteration, the focus is on searching for columns that could enhance the solution (i.e., those with $c^{\alpha}_{plt} \leq 0$) across every period in the planning horizon. The evaluation procedure for the columns generated is elaborated in detail earlier in this section.Algorithm 4 shares similarities with Algorithm 3. While Algorithm 3 focuses on locating backlogged parts, Algorithm 4 searches for random parts near the latest incumbent solution, seeking potential improvements.

Algorithm 3 Generate Backlogged Part Columns.

$List < i > B^{list}$= {}
for each i in I **do**
  for each t in T **do**
    if $b_{it} \geq 0$ **then**
      Add i to B^{list}
    end if
  end for
end for
$N^{iteration} = 1$, $N^{column} = 1$
while $N^{iteration} \leq iteration^{max}$ and $N^{column} \leq column^{max}$ **do**
  Randomly select *i* from B^{list}
  Randomly select *l* from L_i
  for each t in T **do**
    $p =< i >$
    if $c^{\alpha}_{plt} \leq 0$ and $< p, l, t > \notin \Phi$ **then**
      Add $< p, l, t >$ to Φ
      $N^{column} + +$
    end if
    $N^{iteration} + +$
  end for
end while

Algorithm 5 is designed to identify and inject small batches into the existing incumbent columns. The process begins with selecting a random backlogged part, followed by a random compatible machine. Subsequently, the chosen backlogged part is injected into every possible position within the pattern of the latest incumbent solution for each

period. In each iteration, a different place is attempted. The reduced cost of the generated pattern is then computed, and if it meets the criteria for being a promising pattern, it is added to the column pool. Algorithm 6 searches for each pattern that holds a single part *i* throughout the machine-period pairs and finds the columns with the most negative reduced costs. The column of the selected machine-period pair is added to the column pool.

Algorithm 4 Generate Random Improving Columns.

$N^{iteration} = 1$, $N^{column} = 1$

while $N^{iteration} \leq iteration^{max}$ and $N^{column} \leq column^{max}$ **do**

Randomly select *i* from I

Randomly select *l* from L_i

for each t in T **do**

$p =< i >$

if $c^{\alpha}_{plt} \leq 0$ and $< p, l, t > \notin \Phi$ **then**

Add $< p, l, t >$ to Φ

$N^{column} + +$

end if

$N^{iteration} + +$

end for

end while

Algorithm 5 Generate Backlogged Part Columns by Injection.

$List < i > B^{list}$= {}
for each i in I **do**
  for each t in T **do**
    if $b_{it} \geq 0$ **then**
      Add i to B^{list}
    end if
  end for
end for
$N^{iteration} = 1$, $N^{column} = 1$
while $N^{iteration} \leq iteration^{max}$ and $N^{column} \leq column^{max}$ **do**
  Randomly select *i* from B^{list}
  Randomly select *l* from L_i
  for each t in T **do**
    for $n = 0$ to $|p'|$ **where** p' is the incumbent pattern **do**
      p=Inject part *i* to n'th position of pattern p'
      if $c^{\alpha}_{plt} \leq 0$ and $< p, l, t > \notin \Phi$ **then**
        Add $< p, l, t >$ to Φ
        $N^{column} + +$
      end if
    end for
  end for
end while

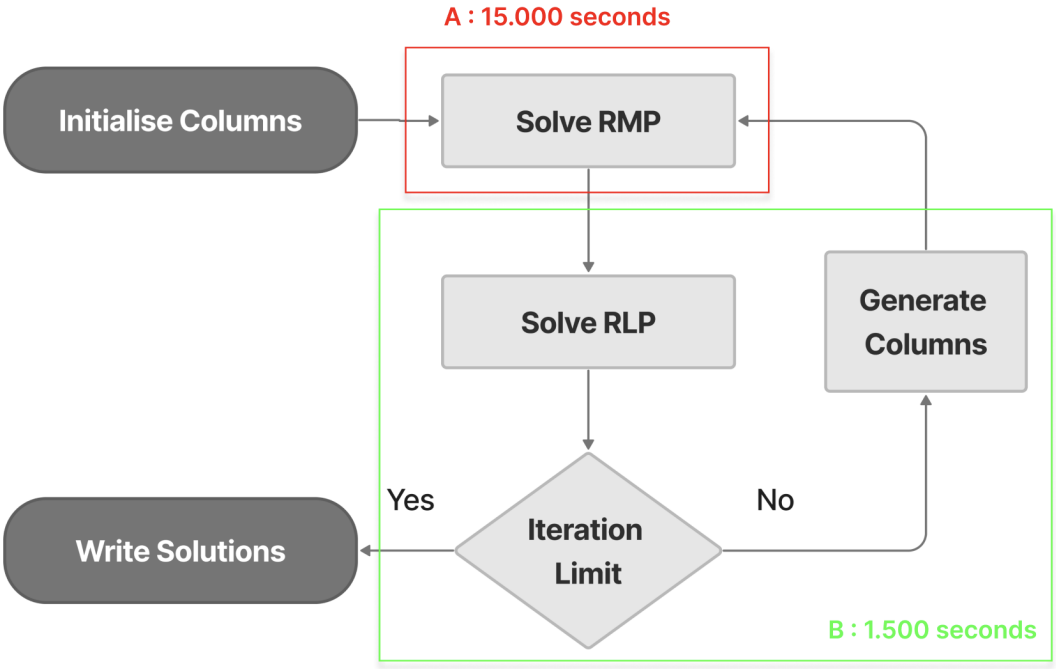
Algorithm 6 Generate Best Position Columns.

for each i in I **do**
  reducedCost=0, l'=0, t'=0
  for each l in L **do**
    for each t in T **do**
      $p =< i >$
      if $c^{\alpha}_{plt} \leq reducedCost$ and $< p, l, t > \notin \Phi$ **then**
        reducedCost=c^{α}_{plt}, l'=l, t'=t
      end if
    end for
  end for
  if $l' \neq 0$ and $t' \neq 0$ **then**
    Add $< p, l', t' >$ to Φ
  end if
end for

4.3 Neural Network-Based Column Pool Generation

The proposed CGNS algorithm uses mechanical search procedures and heuristic calculations of reduced costs to evaluate searched columns to include in the new column pool. As shown in Figure-4.4, the overall solution time for the real-world case, 15.000 seconds, is spent for RMP computation, and 1.500 seconds is consumed for column search and generation phase. The results presented in subsequent sections motivated us to investigate possible machine learning algorithms to simulate the search procedures with higher accuracy and using less computation time.

Figure 4.4: *Temporal CPU times of the overall algorithm.*



The proposed NN structures are designed to evaluate whether introducing a search for part i^* on machine l during period t leads to improvements in the objective function. Let A denote the unique set of distinct parts in the current pattern p of the incumbent column α_{plt}, the last part of the sequence of α_{plt-1} and the first part of the sequence of α_{plt+1}. We extend this set to $\bar{A} = A \cup \{i^*\}$, and define the corresponding power set as $\mathcal{P}(\bar{A}) = \{S \mid S \subseteq \bar{A}\}$. By evaluating each subset $S \in \mathcal{P}(\bar{A})$ and generating all their permutations, we enrich the column pool to explore improved configurations resulting from searching for part i^* on machine l at period t.

This column generation process allows us to assess the impact of including part i^* through the change in the objective function, which we model as a binary output, whether or not the search yields an improvement. To predict this outcome, we propose two types

of neural networks; the first one leverages a permutation-invariant feature set using aggregation functions like minimum, maximum and median and the second one uses a hybrid MLP and RNN structure which are used for sequence invariant and sequence-dependent features respectively. The subsequent subsections detail the selected features, categorized as sequence-independent and sequence-dependent.

4.3.1 Feature Selection and Proposed Neural Network Structures

The input features for the neural network are categorized into two groups: sequence-independent features, which characterize the system regardless of the order of parts, and sequence-dependent features, which capture the temporal relationships among parts within machine schedules.

Sequence-independent features encapsulate the characteristics of the production system, particularly in relation to the searched part i^* and the current schedule. These features are not influenced by the order of products and offer essential insights into key properties of the system, including demand, capacity, inventory levels, production costs, and backlog status.

Table 4.6: *Neural Network Input Features: Searched Part i^*.*

Feature	Description
μ_{lt}^6	Dual variable of Eq-4.16 for machine l at period t
μ_{lt}^7	Dual variable of Eq-4.17 for machine l at period t
PC_l	Unit production cost of machine l
CT_{i^*}	Cycle time of product i^*
b_{i^*t}	Backlog of product i^* in period t
s_{i^*t}	Inventory level of product i^* in period t
q_{i^*lt}	Quantity of product i^* produced on machine l at period t
D'_{i^*}	Average demand of product i^*
D_{i^*t}	Period-specific demand of product i^*
$\mu_{i^*t}^2$	Dual variable of Eq-4.12 for product i^* at period t
$\mu_{i^*lt}^9$	Dual variable of Eq-4.19 for product i^* on machine l at period t
B_{i^*}	Total backlog of product i^*: $\sum_{t'=1}^{T_{\max}} b_{i^*t'}$

Sequence-dependent features, in contrast, capture the dynamic interactions and temporal relationships of parts relative to each other. When considering inserting a new part i^* into an existing sequence, it interacts with preceding and following parts through setup times and resource transitions. These features are derived from the set A, which encompasses the neighbouring components within the current scheduling context.

Each part $i \in A$ is represented by the following sequence-dependent features:

Table 4.7: *Neural Network Input Features: Parts in the Searched Set A.*

Feature	Description
CT_i	Cycle time of product i
b_{it}	Backlog of product i in period t
s_{it}	Inventory level of product i in period t
q_{ilt}	Quantity of product i produced on machine l at period t
D'_i	Average demand of product i
D_{it}	Period-specific demand of product i
μ^2_{it}	Dual variable of Eq-4.12 for product i at period t
μ^9_{ilt}	Dual variable of Eq-4.19 for product i on machine l at period t
ST_{i^*i}	Sequence-dependent setup time from product i^* to product i

These features are replicated for each part i within the searched set A.

The integration of sequence-independent and sequence-dependent features creates a robust input representation for the neural network, allowing it to grasp both the static attributes and the dynamic interactions present in the scheduling environment. Sequence-independent features offer a broad contextual overview, while sequence-dependent features enable the network to evaluate transition costs when incorporating a new part into an existing schedule.

At this point, we propose and test two different neural network structures: one is the concatenation of fixed features from Table-4.6 using an MLP and sequence-dependent features from Table-4.7 using an RNN. The concatenation of both neural networks is fed to a subsequent MLP to predict if the searched part i^* on machine l at period t improves the objective.

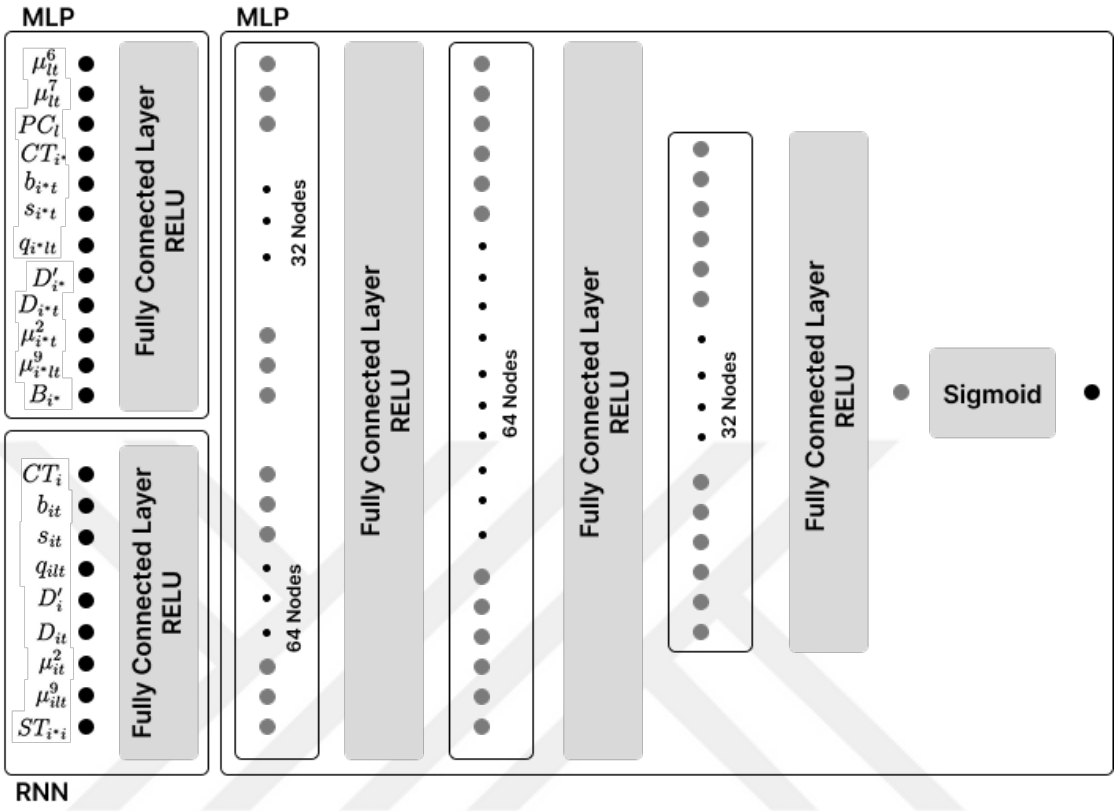
The other proposed approach uses minimum, maximum, and median aggregation functions to ensure permutation-invariance and a consistent input size for the features

from Table-4.7, for each of the nine sequence-dependent attributes, the minimum, maximum, and median values across the parts in A are computed. This aggregation results in a fixed-size feature vector independent of the cardinality of A. Consequently, the final input vector consists of 39 features: 12 features derived from the searched part i^* and 27 features resulting from the aggregated statistics of the surrounding parts.

Based on this feature representation, we propose employing a MLP architecture to predict the effectiveness of inserting part i^* into the current schedule. The objective of the neural network is to output a binary decision indicating whether the insertion is expected to improve the objective function.

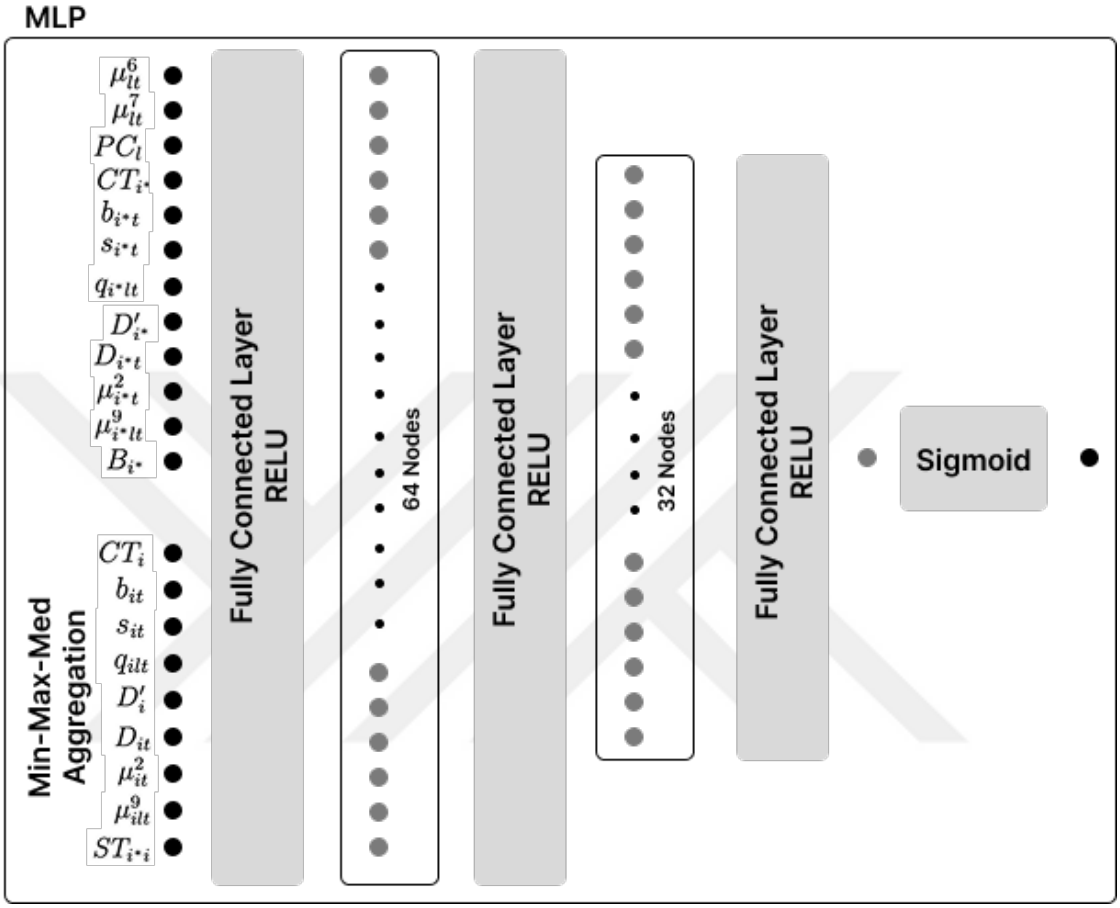
The proposed MLP architecture and the RNN architecture are illustrated in Figure 4.6 and Figure 4.5, respectively. The MLP network consists of an input layer with 39 nodes corresponding to the engineered features, followed by two fully connected hidden layers. The first hidden layer contains 64 nodes, and the second includes 32, utilizing the Rectified Linear Unit (RELU) activation function. A final output node with a Sigmoid activation function produces a probability score, which is interpreted as the binary prediction outcome.

Figure 4.5: *Neural network structure used in the column generation process RNN version.*



On the other hand, the RNN network similarly accepts 12 fixed features and a series of 9 sequence-dependent features. The fixed features are fed by a fully connected layer with RELU activation to 32 nodes. Similarly, the RNN structure is fed to 64 nodes by a fully connected layer with RELU activation. Like the MLP approach, the output is fed to 64 nodes and 32 node hidden layers and finally outputs the probability of objective improvement using a sigmoid activation.

Figure 4.6: *Neural network structure used in the column generation process MLP version.*



The experimental setup and comparison of the two approaches are detailed in the following section.

4.3.2 Integration of Neural Network Predictions into Column Generation and Experimental Results

The Algorithm 7 introduces a hybrid approach that merges neural network guidance with a traditional column generation framework to address capacitated lot-sizing and

scheduling challenges. This algorithm evaluates the parts scheduled in the previous, current, and subsequent periods for each machine across all periods, constructing an extended sequence that captures inter-period interactions.

A neural network is employed to predict the potential impact of inserting a specific product i on machine l during period t on improving the objective function. This model is trained using features derived from existing sequences, alongside static attributes and sequence-dependent characteristics of the product i under consideration. When a favorable prediction is received, the algorithm generates all permutations of subsequences that include the product and adds them to the column pool for evaluation in subsequent iterations.

By effectively narrowing the search space based on neural network predictions, the algorithm enhances the column pool with high-quality candidates. This targeted generation addresses the complexities associated with sequence-dependent setups and significantly boosts computational efficiency by circumventing unnecessary RMP evaluations for configurations with low potential.

Algorithm 7 Generate New Column Pool.

Initialize index list $\Pi \leftarrow \emptyset$ and total score accumulator $S \leftarrow 0$
for $i \in I$ such that $B_i > 0$ **do**
  for $l \in L$, $t \in T$ **do**
    if $IL_{il} = 1$ and $v_{ilt} > \tau$ **then**
      Append (i, l, t) to Π, update $S \leftarrow S + v_{ilt}$
    end if
  end for
end for
for $m = 1$ to $|\Pi|$ **do**
  Sample $(i, l, t) \in \Pi$ with probability $\propto v_{ilt}/S$
  Extract $searchSeq_{ilt}$ and convert to integer sequence σ
  for each $p \in$ *InsertMember*(σ, i) **do**
    if $|\Phi| < N^{max}$ **then**
      Add column $\langle p, l, t \rangle$ to column pool Φ
    end if
  end for
end for

This algorithm develops a refined column pool by filtering candidate triplets (i, l, t) according to neural network scores and feasibility conditions. It subsequently employs stochastic sampling, weighted by prediction scores, to prioritize high-potential configurations. Each chosen configuration is then enhanced through local insertions, adding new candidate sequences to the column pool and adhering to a maximum size constraint. This probabilistic yet guided approach effectively balances the exploration of the solution space with the exploitation of high-quality regions.

4.3.3 Neural Network Training and Performance Evaluation

A series of random instances was generated to train the neural network. These instances consist of random combinations of five machines and twenty products created based on the original dataset while maintaining a target capacity utilization of approximately 70%. Each dataset is solved using the CGNS heuristic for ten iterations. After solving the RLP at each iteration, new training samples are generated using Algorithm 8 and Algorithm 9.

Algorithm 8 Data Generation for Column Pool Update.

```
for each i ∈ I do
    for each l ∈ L do
        if Ω_il = 1 then
            for each period t ∈ T do
                Search_Columns(i, l, t)
            end for
        end if
    end for
end for
```

After regenerating the column pool, the updated RMP is solved, and the corresponding input features and prediction labels are recorded. Negative samples (0 labels) are downsampled to match the number of positive samples to generate a balanced dataset for training.

Algorithm 9 Search Columns(i, l, t).

Inputs: i, l, t

$p^1 \leftarrow A_{l,t-1}$, $p^2 \leftarrow A_{l,t}$, $p^3 \leftarrow A_{l,t+1}$

iFirst $\leftarrow$ last part of p^1, iLast $\leftarrow$ first part of p^3

$p^* \leftarrow$ *Append*(iFirst, p^2)

$p^* \leftarrow$ *Append*(p^*, iLast)

for each subset $s \in \mathcal{S}(p^*)$ **do**

    for each permutation $p \in \mathcal{P}(s)$ **do**

        Add $\langle p, l, t \rangle$ to column pool Φ

    end for

end for

Solve RMP with column pool $\mathcal{P}_t = \Phi \cup \mathcal{I}_{\text{incumbent}}$

Obtain solution S_{RMP}

if Objective value of S_{RMP} improves **then**

    Set ILT$[i, l, t] \leftarrow 1$

else

    Set ILT$[i, l, t] \leftarrow 0$

end if

The confusion matrices and performance metrics for both the whole dataset and the balanced dataset are presented in Table 4.8 and Table 4.9 for MLP and RNN NN's, respectively. The confusion matrices provide insights into true positive, true negative, false positive, and false negative counts. Weighted average F1 score, precision, recall, and accuracy metrics are reported for a comprehensive evaluation.

Initially, we establish a subset of labelled rows from 96600 to create a balanced dataset, ensuring an equal distribution of true and false instances. We then divide this balanced dataset into training and validation sets, with 30% designated for validation

purposes. We then evaluate each neural network on the complete dataset and report the findings. Our observations revealed that the MLP variant outperformed the RNN model on the full dataset, indicating superior generalisation. As a result, we utilise the MLP for further tests, which will be discussed in the subsequent section.

Table 4.8: *Confusion Matrices and Performance Metrics for Full and Balanced Datasets (MLP Model).*

Dataset	**Full Dataset**		**Balanced Dataset**	
	Negative (0)	Positive (1)	Negative (0)	Positive (1)
Negative (0)	**76.957**	18.277	**1.120**	130
Positive (1)	671	**3.495**	183	**1.067**
F1 Score	0.86		0.87	
Precision	0.96		0.88	
Recall	0.81		0.87	
Accuracy	0.81		0.87	

F1 Score, precision, recall and accuracy are calculated as weighted averages.

Table 4.9: *Confusion Matrices and Performance Metrics for Full and Balanced Datasets (RNN Model).*

Dataset	**Full Dataset**		**Balanced Dataset**	
	Negative (0)	Positive (1)	Negative (0)	Positive (1)
Negative (0)	**70.301**	24.933	**1.166**	76
Positive (1)	484	**3.682**	178	**1.080**
F1 Score	0.82		0.90	
Precision	0.96		0.90	
Recall	0.74		0.90	
Accuracy	0.74		0.90	

F1 Score, precision, recall and accuracy are calculated as weighted averages.

5. NUMERICAL STUDIES

In this section, we first validate our proposed approaches with generated data and benchmark against state-of-the-art algorithms and the Gurobi solver. Next, showing the proposed algorithms surpass the benchmarks, we use the algorithm on a set of real-world problems and report the results, showing the improvements in the quality of the production plans in various metrics.

5.1 Numerical results I: validation and comparison with benchmark algorithms

We validate the proposed algorithm using generated data sets and benchmark against a commercial solver and the HDFO algorithm proposed by [13], and the SODBS algorithm proposed by [10], both summarised in the Appendix section. We employ real-life problem data sets for validation data generation, incorporating parameters such as the number of machines and parts, capacity utilisation, and part compatibility ratios. Diverse machine and part combinations are used to generate new data sets. Additional constraints are introduced to the model to facilitate a comparison between the CGNS, CGNSNN, HDFO and SODBS algorithms. Specifically, a binary variable θ_{lt} is introduced to denote setup crossover at either the end of the first period (with a value of 1) or the start of the second period (with a value of 0). Eq. 5.1 and Eq. 5.2 enforce this condition, rendering the results directly comparable with the HDFO and SODBS algorithms. Additionally, the secondary resource constraints (Eq. 4.18, Eq. 4.19, and Eq. 4.20) are excluded from the CGNS and CGNSNN algorithms to align the problem with the HDFO and SODBS algorithms.

$$t^{+}_{lt} \leq \theta_{lt} T^{cap} \qquad (\forall l \in L; \forall t \in T | T^{max}) \tag{5.1}$$

$$t^{-}_{lt+1} \leq (1 - \theta_{lt}) T^{cap} \qquad (\forall l \in L; \forall t \in T | T^{max}) \tag{5.2}$$

The data generation process is as follows: initially, a random assortment of machines and parts is chosen, and their compatibility ratios and capacity utilisation are calculated. If these ratios and utilisation fall within the predefined range for the data set, the outcomes are recorded for subsequent use in numerical examples. The presence of compatible parts for each machine is verified for each generated dataset.

All the generated instances exhibit capacity utilisation between 0.70% and 0.75%. Problem instances, covering various problem sizes, ranging from 10 to 30 machines and 50 to 120 parts, are created and used on the tests to ensure the algorithms are versatile enough. These data sets are identified using abbreviations, with "L" representing machines and "I" representing parts.

In the numerical solution, the cost pattern employed is 0.6 for backlogging, 0.0001 for inventory, 0.00001 times the machine's tonnage (averaging around 1000 tons) for production, and 0.1 for setup costs, which are based on data from our partner company.

As outlined in the column search heuristics section, the algorithm is designed to produce a limited number of columns during column search procedures. Table 5.1 outlines the maximum number of columns generated by each column search procedure. Empirical observations on large instances lead us to establish these figures. The objective is to uphold a suitably vast number of generated columns, broadening the search space and evading local minima while ensuring that solution times remain reasonable.

Table 5.1: *Number of Columns Generated by the CGNS and CGNSNN Algorithms.*

Algorithm	$iter^{max}$	$column^{max}$
Alg. 1 Generate last solution columns	L×T	L×T×11
Alg. 2 Generate carryover columns	L×T	L×T×11
Alg. 3 Generate backlogged part columns	1,000,000	2,000
Alg. 4 Generate random improving columns	1,000,000	500
Alg. 5 Generate backlogged part columns by injection	1,000,000	2,500
Alg. 6 Generate best position columns	I×L×T	I^{max}
Alg. 7 NN-based column generation	I×L×T	N^{max}

N^{max} is set to 2,000 for generated problems and 4,000 for real-world instances.

Tests are conducted on L30-I100 data sets to assess the additional value of different search heuristic approaches within the overall CGNS algorithm. Table 5.2 presents the validation and comparative results of distinct column search heuristics within the CGNS algorithm across ten instances of data sets, each consisting of 30 machines and 100 parts. The instances are solved using an exact method (Gurobi) and the CGNS algorithm, employing four distinct versions: CGNS algorithms (3), (4), (5), and (6). Each version of the CGNS algorithm removes the associated search method from the neighbourhood search phase. It is essential to highlight that search methods, algorithms (1) and (2), are considered trivial and are not tested for their contribution to the overall solution. The table illustrates that while excluding specific search heuristics from the algorithm might reduce the objective gap for particular instances, the CGNS algorithm that encompasses all search heuristics consistently outperforms both the Exact method and alternative versions of the CGNS algorithm. Similarly, the CGNSNN algorithm uses algorithms (1) and (2), which are trivial, and uses a parameter of N^{max} for the maximum number of columns that can be added to the column pool at each iteration.

Table 5.2: *Validation and Comparison of Column Search Heuristics.*

		Gap					
Data (L-I)	Instance	Gurobi	CGNS	CGNS$^{(3)}$	CGNS$^{(4)}$	CGNS$^{(5)}$	CGNS$^{(6)}$
30-100	1	23,6%	35,9%	35,5%	**29,3%**	34,0%	37,4%
	2	37,8%	28,5%	34,4%	30,7%	27,9%	**24,2%**
	3	30,7%	**22,2%**	24,8%	24,4%	23,6%	25,1%
	4	33,7%	32,5%	30,3%	33,7%	**30,2%**	33,8%
	5	67,3%	**33,5%**	45,0%	38,0%	46,3%	33,8%
	6	11,1%	9,3%	9,7%	9,7%	**8,8%**	9,0%
	7	47,8%	39,2%	**29,6%**	43,5%	36,6%	42,2%
	8	45,3%	39,0%	41,3%	37,1%	41,1%	**36,6%**
	9	48,7%	34,1%	34,4%	33,9%	34,7%	**33,7%**
	10	36,8%	35,4%	31,9%	31,9%	**31,5%**	34,5%
Average		38,28%	**30,95%**	31,7%	31,2%	31,5%	31,03%

Furthermore, we conducted a series of tests to thoroughly assess the reliability and performance of the proposed CGNS algorithm. For the L30-I100 datasets, the time limit was set to 3600 seconds, while for the other datasets, it was limited to 1800 seconds. During the Rolling Horizon phase of the HDFO and SODBS algorithms, four periods were utilised per iteration $(\alpha, \mu) = (4, 0)$ over 400 seconds, as outlined in [13]. Additionally, we incorporated the Rolling Horizon (RH) algorithm with two different parameter settings: $(\alpha, \mu) = (3, 1)$, referred to as RH1(3-1), and $(5, 1)$, referred to as RH2(5-1). This approach allowed us to examine the sensitivity of the benchmark algorithms to varying initialisation phase configurations, as discussed in [10]. As illustrated in Table 5.3, when RH1 and RH2 phases were run for the entire dataset time limits, they either failed to generate initial solutions or delivered unsatisfactory results compared to the CGNS algorithm.

Building on this analysis, we further investigated the performance of the benchmark algorithms when the initialisation phase time limits were extended. For the 20-100 and 20-120 datasets (20 machines - 100 parts and 20 machines - 120 parts), the limits were increased to 1000 seconds, while for the 30-100 dataset (30 machines - 100 parts), the limit was set to 2000 seconds, as presented in Table 5.5. The parameters used for the rolling horizon phase of the algorithms in this study remained $(\alpha, \mu) = (4, 0)$. The results indicate that the rolling horizon algorithm struggled to generate initial solutions even with these extended time limits for the initialisation phase. For the 30-100 dataset, despite successful initialisation, the benchmark algorithms still lacked sufficient time to improve solution quality, resulting in an average gap of around 20% compared to the best results obtained in the numerical studies in Table 5.3.

The validation study reveals that within the given time limits, the Gurobi, HDFO, and SODBS algorithms demonstrate superior performance for smaller problem sizes. However, as the problem size increases in terms of machines and parts, the CGNS algorithm outperforms both the benchmark HDFO and SODBS algorithms, as well as Gurobi. As the problem grows, the HDFO and SODBS algorithms struggle to produce solutions within the allotted time.

Notably, the HDFO and SODBS algorithms begin to show diminished performance at problem sizes of L20-I80 and fail to generate solutions starting from the L20-I100 datasets. In contrast, the CGNS and CGNSNN algorithms surpass Gurobi from the L20-I100 dataset onwards and remain competitive with benchmark algorithms on smaller problems.

Table 5.3: *Overall Comparison of the Proposed Algorithms and Benchmarks.*

			Gap						
Data (L-I)	Instance	Time limit	CGNS	Gurobi	HDFO	RH1 (3-1)	RH2 (5-1)	SODBS	CGNSNN
10-50	1	1800	1.70%	**0.63%**	0.64%	0.88%	3.52%	1.21%	1.53%
	2		3.25%	**1.19%**	1.48%	19,09%	26,32%	13,53%	2.72%
	3		0.95%	**0.49%**	0.59%	2,79%	1,25%	0,80%	0.58%
	4		3.10%	**1.22%**	1.49%	2,88%	4,51%	1,74%	2.86%
	5		1.61%	0.50%	**0.49%**	2,40%	13,02%	1,51%	0.71%
	6		2.43%	1.92%	1.99%	2,23%	2,32%	**0,87%**	1.95%
	7		4.60%	4.19%	4.21%	10.82%	4.15%	**1.86%**	4.98%
	8		18.51%	**14.02%**	14.70%	14.35%	16.12%	14.46%	14.36%
	9		1.25%	1.10%	1.49%	2.15%	2.13%	**0.44%**	1.16%
	10		1.91%	1.39%	1.48%	3.41%	1.77%	**0.36%**	1.83%
Average			3.93%	**2.66%**	2.86%	6.10%	7.51%	3.68%	3.27%
10-80	1	1800	5.76%	**5.05%**	5.33%	6,56%	7,78%	6,38%	5.57%
	2		9.51%	7.98%	8.74%	15,28%	5,57%	**2,63%**	8.87%
	3		16.11%	13.61%	**11.36%**	32,09%	24,32%	23,57%	14.15%
	4		2.10%	2.34%	2.35%	4,66%	6,72%	2,26%	**1.56%**
	5		**9.05%**	9.65%	12.09%	16,34%	16,61%	14,26%	10.61%
	6		3.82%	3.72%	**3.65%**	3,38%	6,18%	4,35%	3.68%
	7		3.75%	2.76%	5.01%	6,78%	4,06%	**1,56%**	3.68%
	8		3.10%	2.51%	4.36%	5,18%	3,73%	**2,17%**	2.72%
	9		3.66%	3.64%	5.37%	5,34%	5,47%	**0,60%**	3.96%
	10		6.72%	6.49%	**6.11%**	6,86%	10,03%	7,09%	6.91%
Average			6.36%	**5.71%**	6.44%	10.25%	9.05%	6.49%	6.17%
20-80	1	1800	15.37%	20.83%	50.34%	38,92%	28,63%	5,65%	**16.66%**
	2		6.51%	**4.89%**	15.03%	19,24%	16,57%	13,99%	5.34%
	3		21.82%	16.00%	22.78%	48,70%	18,05%	**14,12%**	18.65%
	4		16.20%	**11.52%**	22.76%	19,21%	21,24%	20,75%	14.17%
	5		**10.67%**	13.69%	16.88%	22,19%	32,85%	31,59%	11.08%
	6		25.70%	**21.56%**	59.03%	67,07%	43,68%	41,27%	28.81%
	7		2.27%	2.36%	NS	NS	8,00%	NS	**2.26%**
	8		7.09%	7.12%	6.92%	9,19%	9,00%	7,15%	**6.08%**
	9		6.30%	**3.67%**	8.18%	9,59%	8,91%	5,63%	6,00%
	10		4.29%	7.42%	14.47%	11,88%	7,51%	4,61%	**4,11%**
Average			11.62%	**10.91%**	24.04%*	27.33%*	19.45%	18.31%*	11.32%
20-100	1	1800	6.42%	10.01%	NS	NS	21,09%	NS	**6.06%**
	2		17.74%	20.97%	32.98%	51,50%	35,84%	35,46%	**14.22%**
	3		10.37%	10.74%	24.34%	NS	23,29%	20,78%	**9.43%**
	4		13.41%	19.79%	NS	NS	45,01%	NS	**11.76%**
	5		10.52%	12.53%	NS	NS	28,61%	NS	**7.66%**
	6		24.16%	35.07%	NS	NS	43,59%	NS	**22.19%**
	7		6.11%	8.23%	NS	NS	14,95%	NS	**5.40%**
	8		9.88%	17.82%	NS	NS	38,52%	NS	**8.29%**
	9		3.60%	3.76%	NS	NS	11,19%	NS	**2.85%**
	10		12.60%	**9.80%**	NS	NS	23,91%	NS	10.84%
Average			11.48%	14.87%	28.66%*	51.50%*	28.60%	28.12%*	**9.87%**
20-120	1	1800	**1,99%**	10,91%	NS	NS	14,78%	NS	2,22%
	2		13,95%	46,89%	NS	NS	37,78%	NS	**12,08%**
	3		**7,86%**	23,51%	NS	NS	35,66%	NS	8,47%
	4		7,24%	26,27%	NS	NS	NS	NS	**6,57%**
	5		9,16%	29,06%	NS	NS	NS	NS	**7,18%**
	6		**29,55%**	62,04%	NS	NS	NS	NS	30,04%
	7		**5,29%**	19,20%	NS	NS	31,85%	NS	5,39%
	8		29,19%	58,08%	NS	NS	NS	NS	**23,03%**
	9		7,87%	21,72%	NS	NS	21,41%	NS	**4,35%**
	10		18,44%	31,27%	NS	NS	51,37%	NS	**17,59%**
Average			13.05%	32.90%	NS	NS	32.14%*	NS	**11,69%**
30-100	1	3600	35,91%	**23,63%**	NS	NS	49,14%	NS	32,99%
	2		28,54%	37,77%	NS	NS	70,32%	NS	**28,43%**
	3		22,16%	30,70%	NS	NS	45,41%	NS	**20,60%**
	4		32,49%	33,66%	NS	NS	51,00%	NS	**27,62%**
	5		40,92%	67,26%	NS	NS	74,60%	NS	**35,02%**
	6		9,31%	11,12%	NS	NS	26,70%	NS	**8,93%**
	7		39,17%	47,82%	NS	NS	57,21%	NS	**21,54%**
	8		39,02%	45,32%	NS	NS	66,41%	NS	**34,68%**
	9		34,05%	48,74%	NS	NS	57,18%	NS	**25,90%**
	10		35,38%	36,80%	NS	NS	59,44%	NS	**28,50%**
Average			31.69%	38.28%	NS	NS	55.74%	NS	**26,42%**

*Average values are calculated only for instances where the algorithm successfully provided a solution.
NS: Instances where the algorithm did not yield a solution.

Table 5.4: *Paired t-test Results Comparing Proposed Algorithms and Gurobi.*

Comparison	**Mean$_{\text{CGNS}}$**	**Mean$_{\text{Other}}$**	**Mean Diff.**	**t-stat**	**p-value**	**Signif.**
CGNS vs Gurobi	13.02%	17.57%	-4.54%	-3.77	0.00038	***
CGNS vs CGNSNN	13.02%	11.46%	+1.57%	4.11	0.00012	***

Note: Significance levels: * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

As summarised in Table 5.4, the paired t-test analyses demonstrate that the proposed CGNS algorithm significantly outperforms the commercial solver Gurobi, with a p-value well below the 0.01 threshold. Moreover, further improvement is achieved by the learning-augmented variant CGNSNN, which also shows a statistically significant advantage over the base CGNS method.

These results confirm the robustness and effectiveness of the proposed column generation framework and provide strong evidence that incorporating machine learning techniques (as in CGNSNN) enhances the ability to identify high-quality production patterns within computationally efficient timeframes.

The following section thoroughly analyses the algorithm's performance in a real-world industrial context, benchmarking its results against the original production plans crafted by expert planners. The comparison emphasises key performance indicators such as total cost, production cost, and backlog cost. By assessing the algorithm under realistic operational constraints and data, this section seeks to illustrate its practical effectiveness and potential for integration into decision-support systems. The findings underscore how the proposed approach enhances planning consistency, minimises setup-related inefficiencies, and is a scalable alternative to traditional manual scheduling methods.

Table 5.5: *Comparison of Benchmark Algorithms on Large Datasets with Extended Initialization Time Limits.*

			Gap (%)			
Data (L-I)	Instance	Time limit	Best	HDFO	SODBS	Best Algorithm
20-100	1	1800	6.06	19.73	19.75	CGNSNN
	2		14.22	36.57	34.43	CGNSNN
	3		9.43	NS	NS	CGNSNN
	4		11.76	NS	NS	CGNSNN
	5		7.66	NS	NS	CGNSNN
	6		22.19	NS	NS	CGNSNN
	7		5.40	15.28	12.49	CGNSNN
	8		8.29	NS	NS	CGNSNN
	9		2.85	10.61	10.00	CGNSNN
	10		9.8	NS	NS	CGNS
20-120	1	1800	1.99	NS	NS	CGNS
	2		12.08	NS	NS	CGNSNN
	3		7.86	NS	NS	CGNS
	4		6.57	NS	NS	CGNSNN
	5		7.18	NS	NS	CGNSNN
	6		29.55	NS	NS	CGNS
	7		5.29	NS	NS	CGNS
	8		23.93	NS	NS	CGNSNN
	9		4.35	NS	NS	CGNSNN
	10		17.59	NS	NS	CGNSNN
30-100	1	3600	23.63	62.65	47.71	Gurobi
	2		28.43	71.33	69.87	CGNSNN
	3		20.60	41.02	44.56	CGNSNN
	4		27.62	55.30	49.50	CGNSNN
	5		35.02	70.70	74.22	CGNSNN
	6		8.93	19.48	25.88	CGNSNN
	7		21.54	NS	NS	CGNSNN
	8		34.68	76.02	65.57	CGNSNN
	9		25.90	64.39	56.04	CGNSNN
	10		28.50	NS	NS	CGNSNN

NS: No solution found within the time limit.
Best Algorithm: Indicates the algorithm with the lowest gap for each instance.

5.2 Numerical results II: A real-life case

This section compares the results of the CGNS algorithm with the company's production plans. We rigorously assess the algorithm's efficacy using the company's authentic plastic injection plant data. This data is sourced from ten distinct system snapshots within the online SAP repository, ensuring real-life applicability and relevance. The company's prevailing planning approach relies on the expertise of planners and a user interface tailored for visual monitoring of demand and production orders.

Conversely, the CGNS algorithm retrieves updated demand data from the server at midnight and uses it to craft a production plan by morning. This approach integrates instant demand, inventory metrics, and live production orders from the database. The computational time limit for the RMP is 1000 seconds. Moreover, the algorithm's overall runtime extends to around four hours to ensure the attainment of conclusive solutions. This time limit is applicable and suitable for the problem since the results are ready at the beginning of the morning shift.

We structure the evaluation of the CGNS algorithm around four dimensions: total production costs, setup durations, inventory levels, and backlogs. The computational environment utilised for this analysis is a Mac computer equipped with a 2.4 GHz i9 processor and 64 GB of RAM. Visual Studio 2019 Community Edition (.Net) is utilised for implementation purposes, and to address the RLP and RMP problems, the Gurobi 9.5 solver is employed.

The comparison between the original plans and those generated by the algorithm is related to their respective cost implications. Figure 5.1 represents a comprehensive visualisation of inventory comparisons. This representation displays inventory profiles based on the machine workload essential for inventory buildup. The findings underscore the algorithm's tendency to maintain moderately lower initial inventory levels, gradually transitioning to slightly elevated levels as the planning horizon progresses. Overall, the

algorithm's inventory trends align closely with those observed in the original plan.

Turning our attention to the domain of backlog management, Figure 5.2 exhibits the algorithm's propensity to outperform the original plans devised by the planners. Notably, the algorithm achieves this by substantially reducing backlog levels, particularly in the immediate periods of the planning horizon. The CGNS algorithm achieves this by increasing the setup times at the beginning of the planning horizon.

Addressing the relationship between backlog levels and setup times, Figure 5.3 offers valuable insights. The illustration shows that the original plan is inclined to initiate more setups during the second and third periods of the planning horizon compared to the plan devised by the algorithm. However, it's worth noting that the algorithm-integrated plan records an overall escalation of around 12% in setup costs, primarily attributed to the amplified setups during the first period and the setups made in the second half of the planning horizon.

Figure 5.1: *Inventory Comparison.*

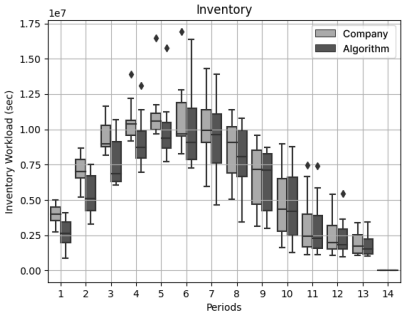


Figure 5.2: *Backlog Comparison.*

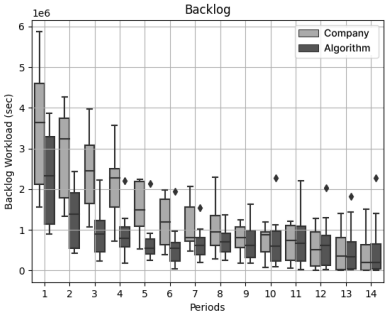


Figure 5.3: *Setup Comparison.*

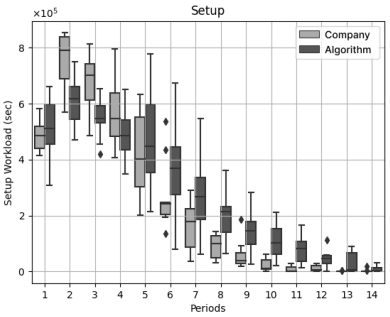


Table 5.6 encapsulates a holistic summary of the algorithm's performance against the actual plans. The empirical evidence derived from our test instances convincingly substantiates the algorithm's supremacy over the company's existing plans. The quantification of production cost, as described in Table 5.6, involves the multiplication of total tonnage with hour-consumption parameters directly influenced by the algorithm's interventions. These results reflect a 2.8% enhancement in the average tonnage consumed by the company, indicating improved production costs. In a broader context, the algorithm achieves a noteworthy average improvement of 44.5% across ten data sets compared to the existing plans.

Diving into finer details, Table 5.7 shows a detailed cost comparison encompassing setup, inventory, backlog, and production across diverse data sets. The algorithm consistently exhibits superior cost performance, outperforming the original plans across all test instances. Notably, the algorithm consistently reduces inventory, backlog, and production costs while incurring a marginal increase in setup costs.

Table 5.6: *Overall Comparison of the Proposed Algorithm and the Company's Original Plan.*

Instance	Total Cost Original (x1.000.000)	Total Cost CGNS (x1.000.000)	Imp.	Average Tonnage Original	Average Tonnage CGNS	Tonnage Imp.
1	18,6	11,0	41,0%	965	941	2,5%
2	16,8	9,4	44,3%	959	937	2,3%
3	16,5	9,0	45,5%	968	953	1,5%
4	14,0	7,2	48,7%	976	960	1,6%
5	14,7	9,7	34,1%	1014	988	2,6%
6	15,5	8,9	42,6%	1016	989	2,7%
7	11,6	6,0	48,4%	1025	984	4,0%
8	4,9	2,8	42,5%	1043	1002	3,9%
9	6,1	2,6	57,8%	1052	1013	3,7%
10	7,0	3,3	52,3%	1058	1024	3,2%
Avg.	**12,6**	**7,0**	**44,5%**	**1008**	**979**	**2,8%**

Table 5.7: *Cost Comparisons of the Proposed Algorithm and the Company's Original Plans Regarding Setup, Inventory, Backlog, and Production Costs.*

	Original Plan				CGNS Algorithm			
Data Set	Setup Cost (x1.000)	Inv. Cost (x1.000)	Back. Cost (x1.000)	Prod. Cost (x1.000)	Setup Cost (x1.000)	Inv. Cost (x1.000)	Back. Cost (x1.000)	Prod. Cost (x1.000)
1	341,6	10,1	17737,2	524,16	410,8	8,9	10046,6	512,4
2	341,6	9,8	15946,6	497,4	416,2	8,5	8440,5	488,5
3	376,2	9,3	15611,9	518,3	470,9	8,0	8015,4	512,3
4	387,4	9,3	13144,5	491,2	466,6	8,2	6233,5	484,8
5	439,6	10,0	13692,0	582,5	535,0	9,1	6586,7	567,0
6	357,8	7,0	14642,1	480,7	370,4	5,6	8050,5	466,3
7	305,3	7,3	10866,0	443,5	344,9	6,3	5215,6	425,3
8	293,4	8,0	4224,5	384,3	252,4	7,6	2193,9	368,9
9	323,3	8,6	5363,4	386,0	324,0	8,0	1863,4	371,8
10	362,9	11,6	6185,9	440,2	360,7	10,9	2544,5	426,1
Avg.	**352,9**	**9,1**	**11741,4**	**474,8**	**395,2**	**8,1**	**6119,1**	**462,3**

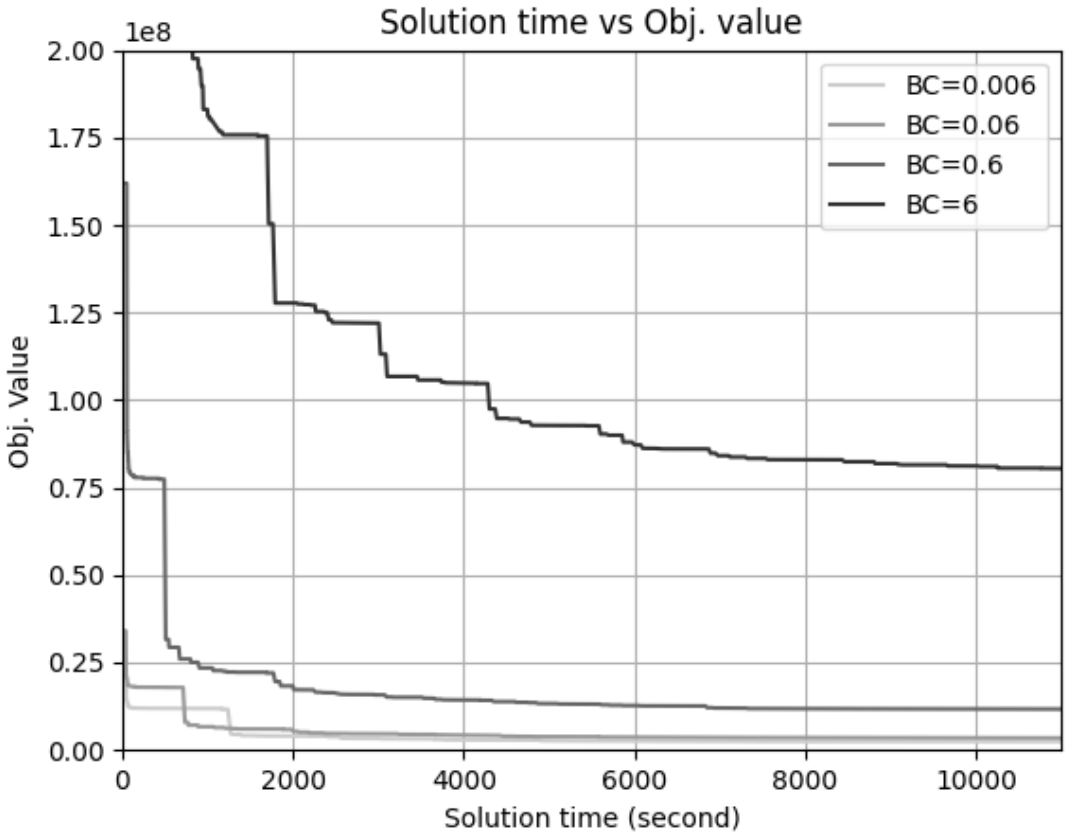
The cost figures in the production planning problem (i.e. inventory and backlog costs) may change over time. To test the performance of the proposed algorithm in varying cost scenarios, we apply a comprehensive and detailed sensitivity analysis on the backlogging cost. Data set 5 is used for this investigation, generating solutions under different backlogging cost scenarios: 0.006, 0.076, 0.6, and 6. The results unequivocally validate the algorithm's superiority over the original plan, demonstrating a clear correlation between the magnitude of improvement and the chosen backlogging costs. Notably, increased backlogging costs lead to higher setup costs, counterbalanced by reduced inventory costs. Table 5.8 presents the results for a more in-depth understanding of the sensitivity analysis.

Finally, Figure 5.4 unravels the temporal evolution of the problem's objective value with solution time, depicted in, encompassing a range of backlogging cost scenarios. The outcomes distinctly illustrate that, for each backlogging cost parameter, the refinement of the optimisation gap remains consistent, ultimately converging towards the final solution within the time frame of approximately 6000 to 8000 seconds. This aspect underscores the algorithm's efficacy in generating solutions within the imposed four-hour time limit.

Table 5.8: *Variation of the Costs Due to the Change in Unit Backlogging Cost Comparison of the Company's Original Plan and the Proposed Algorithm.*

	Original Plan				CGNS Algorithm				
Data Set	Setup Cost (x1.000)	Inv. Cost (x1.000)	Back. Cost (x1.000)	Prod. Cost (x1.000)	Setup Cost (x1.000)	Inv. Cost (x1.000)	Back. Cost (x1.000)	Prod. Cost (x1.000)	Imp.
$C^{back} = 6$	439,6	10,0	136920,0	582,5	608,8	8,8	77550,4	567,2	42,9%
$C^{back} = 0.6$	439,6	10,0	13692,0	582,5	535,0	9,1	6586,7	567,0	34,1%
$C^{back} = 0.06$	439,6	10,0	1369,2	582,5	425,5	9,3	977,2	565,6	17,6%
$C^{back} = 0.006$	439,6	10,0	136,9	582,5	263,2	9,8	123,7	562,6	17,9%

Figure 5.4: *Solution Time, Objective Value Evaluation. Sensitivity Analysis Over Backlogging Costs.*



Furthermore, we evaluate the performance of the CGNSNN algorithm in comparison to the baseline CGNS approach by analysing the total cost outcomes across multiple real-world datasets. A statistical paired t-test is conducted to assess whether the differences in total cost between the two algorithms are significant. As shown in Table 5.9, the CGNSNN algorithm yields slightly higher average cost values across the test instances. However, the calculated p-value of 0.4086 indicates that the observed differences are not statistically significant at the 5% significance level. This suggests that while CGNSNN offers comparable results to CGNS, the improvement is not consistent enough to reject

the null hypothesis of no difference between the algorithms on very large datasets.

Table 5.9: *Paired t-test Comparison of CGNS and CGNSNN Total Costs on Real World Instances.*

Dataset	Total Cost CGNS	Total Cost CGNSNN	t-stat	p-value	Signif.
1	10978234,51	10754541,46			
2	9353615,68	9273787,11			
3	9006574,47	8992273,06			
4	7193023,81	7271024,82			
5	9697698,82	9655599,19	-0,8667	0,4086	$> 5\%$
6	8892836,30	8849275,75			
7	5992175,56	6351071,04			
8	2822717,55	2857553,87			
9	2567152,36	2698678,16			
10	3342228,82	3612380,79			
Mean	6984625,79	7031618,53			

6. CONCLUSIONS

This thesis addressed the computational complexity and scalability challenges inherent in simultaneous lot-sizing and scheduling problems, particularly in industrial settings characterised by sequence-dependent setups, secondary resource constraints, and setup carryovers. Traditional optimisation methods often fail to scale or adapt efficiently to these large and realistic problem instances. To overcome this, the study proposed a hybrid solution framework that integrates column generation with neighbourhood search and machine learning driven decision support.

The first major contribution of this research is the development of the Column Generation with Neighbourhood Search (CGNS) algorithm. CGNS introduces a novel decomposition strategy that combines machine-based and time-based subproblem generation with a controlled column pool exploration mechanism. Experimental results across a comprehensive benchmark set demonstrated that CGNS consistently outperformed benchmark approaches such as HDFO, SODBS, and commercial solvers.

The second and more advanced contribution is the CGNSNN algorithm, which extends CGNS by incorporating neural network-based predictions into the column generation process. A deep learning model is trained to predict the likelihood of improvement for specific item-machine-period combinations, thereby guiding the neighbourhood search more intelligently.

The integration of machine learning into mathematical optimisation, demonstrated through CGNSNN, represents a significant contribution to the field. This hybrid framework not only accelerates the solution process but also enhances the quality of the results by focusing computational effort on promising regions of the solution space. The resulting approach is both scalable and robust, capable of solving real-world problems previously deemed too large or complex for exact methods.

Beyond its academic contributions, this research provides industrially viable methods for automated and intelligent production planning in complex manufacturing environments. The algorithms developed here are applicable to a wide range of scheduling problems beyond plastic injection moulding, including pharmaceuticals, automotive, and electronics manufacturing.

Future research may explore extending this hybrid framework to multi-objective settings, integrating reinforcement learning for adaptive online decision-making, and applying transfer learning techniques to generalise across different production environments. Moreover, the modular design of CGNSNN offers potential for deployment in distributed and real-time optimisation settings, enabling smart factory systems under Industry 4.0 paradigms.

REFERENCES

[1] K. Copil, M. Wörbelauer, H. Meyr, and H. Tempelmeier, "Simultaneous lotsizing and scheduling problems: a classification and review of models," *OR Spectrum*, vol. 39, no. 1, pp. 1–64, 2017.

[2] E. Albey, Ü. Bilge, and R. Uzsoy, "An exploratory study of disaggregated clearing functions for production systems with multiple products," *International Journal of Production Research*, vol. 52, pp. 5301–5322, Apr. 2014.

[3] Y. Kang, E. Albey, S. Hwang, and R. Uzsoy, "The impact of lot-sizing in multiple product environments with congestion," *Journal of Manufacturing Systems*, vol. 33, no. 3, pp. 436–444, 2014.

[4] K. Copil, M. Wörbelauer, H. Meyr, and H. Tempelmeier, "Simultaneous lotsizing and scheduling problems: a classification and review of models," *OR Spectrum*, vol. 39, no. 1, pp. 1–64, 2017.

[5] A. Drexl and A. Kimms, "Lot sizing and scheduling — survey and extensions," *European Journal of Operational Research*, vol. 99, no. 2, pp. 221–235, 1997.

[6] D. Gupta and T. Magnusson, "The capacitated lot-sizing and scheduling problem with sequence-dependent setup costs and setup times," *Computers & Operations Research*, vol. 32, no. 4, pp. 727–747, 2005.

[7] S. Ghosh Dastidar and R. Nagi, "Scheduling injection molding operations with multiple resource constraints and sequence dependent setup times and costs," *Computers & Operations Research*, vol. 32, no. 11, pp. 2987–3005, 2005.

[8] J. Kang, "Capacitated lot-sizing problem with sequence-dependent setup, setup carryover and setup crossover," *Processes*, vol. 8, no. 7, 2020.

[9] C. U. Şafak, "Simultaneous lot sizing, scheduling, workforce, overtime and shift planning: Mip model including setup times and backlogging decisions," master's thesis, Özyeğin University, Istanbul, Turkey, Dec. 2017.

[10] C. Zhang, D. Zhang, and T. Wu, "Data-driven branching and selection for lot-sizing and scheduling problems with sequence-dependent setups and setup carryover," *Computers & Operations Research*, vol. 132, p. 105289, 2021.

[11] S. Ghosh Dastidar and R. Nagi, "Scheduling injection molding operations with multiple resource constraints and sequence dependent setup times and costs," *Computers & Operations Research*, vol. 32, no. 11, pp. 2987–3005, 2005.

[12] I.-S. Kwak and I.-J. Jeong, "A hierarchical approach for the capacitated lot-sizing and scheduling problem with a special structure of sequence-dependent setups," *International Journal of Production Research*, vol. 49, no. 24, pp. 7425–7439, 2011.

[13] L. Z. Jing Xiao, Canrong Zhang and J. N. D. Gupta, "Mip-based fix-and-optimise algorithms for the parallel machine capacitated lot-sizing and scheduling problem," *International Journal of Production Research*, vol. 51, no. 16, pp. 5011–5028, 2013.

[14] Y. Kang, E. Albey, and R. Uzsoy, "Rounding heuristics for multiple product dynamic lot-sizing in the presence of queueing behavior," *Computers & Operations Research*, vol. 100, pp. 54–65, 2018.

[15] M. Alimian, V. Ghezavati, R. Tavakkoli-Moghaddam, and R. Ramezanian, "Solving a parallel-line capacitated lot-sizing and scheduling problem with sequence-dependent setup time/cost and preventive maintenance by a rolling horizon method," *Computers & Industrial Engineering*, vol. 168, p. 108041, 2022.

[16] K. Cervantes-Sanmiguel, M. Vargas-Flores, and O. Ibarra-Rojas, "A two-stage sequential approach for scheduling with lot-sizing decisions in the context of plastic injection systems," *Computers & Industrial Engineering*, vol. 151, p. 106969, 2021.

[17] O. J. Ibarra-Rojas, R. Z. Ríos-Mercado, Y. A. Rios-Solis, and M. A. Saucedo-Espinosa, "A decomposition approach for the piece–mold–machine manufacturing problem," *International Journal of Production Economics*, vol. 134, no. 1, pp. 255–261, 2011. Enterprise risk management in operations.

[18] C. Almeder and B. Almada-Lobo, "Synchronisation of scarce resources for a parallel machine lotsizing problem," *International Journal of Production Research*, vol. 49, no. 24, pp. 7315–7335, 2011.

[19] K. P. Martínez, R. Morabito, and E. A. V. Toso, "A coupled process configuration, lot-sizing and scheduling model for production planning in the molded pulp industry," *International Journal of Production Economics*, vol. 204, pp. 227–243, 2018.

[20] H. Tempelmeier and K. Copil, "Capacitated lot sizing with parallel machines, sequence-dependent setups, and a common setup operator," *OR Spectrum*, vol. 38, no. 4, pp. 819–847, 2016.

[21] W. Kaczmarczyk, "Proportional lot-sizing and scheduling problem with identical parallel machines," *International Journal of Production Research*, vol. 49, no. 9, pp. 2605–2623, 2011.

[22] H. Tempelmeier and L. Buschkühl, "Dynamic multi-machine lotsizing and sequencing with simultaneous scheduling of a common setup resource," *International Journal of Production Economics*, vol. 113, no. 1, pp. 401–412, 2008. Research and Applications in E-Commerce and Third-Party Logistics Management Special Section on Meta-standards in Operations Management: Cross-disciplinary perspectives.

[23] Z. Hu and G. Hu, "A two-stage stochastic programming model for lot-sizing and scheduling under uncertainty," *International Journal of Production Economics*, vol. 180, pp. 198–207, 2016.

[24] B. Almada-Lobo, D. Klabjan, M. A. Carravilla, and J. Oliveira, "Multiple machine continuous setup lotsizing with sequence-dependent setups," *Computational Optimization and Applications*, vol. 47, no. 3, pp. 529–552, 2010.

[25] T. Wu, F. Xiao, C. Zhang, Y. He, and Z. Liang, "The green capacitated multi-item lot sizing problem with parallel machines," *Computers & Operations Research*, vol. 98, pp. 149–164, 2018.

[26] D. M. Carvalho and M. C. Nascimento, "Lagrangian heuristics for the capacitated multi-plant lot sizing problem with multiple periods and items," *Computers & Operations Research*, vol. 71, pp. 137–148, 2016.

[27] J. Xiao, H. Yang, C. Zhang, L. Zheng, and J. N. Gupta, "A hybrid lagrangian-simulated annealing-based heuristic for the parallel-machine capacitated lot-sizing and scheduling problem with sequence-dependent setup times," *Computers & Operations Research*, vol. 63, pp. 72–82, 2015.

[28] F. Sahling, L. Buschkühl, H. Tempelmeier, and S. Helber, "Solving a multi-level capacitated lot sizing problem with multi-period setup carry-over via a fix-and-optimize heuristic," *Computers & Operations Research*, vol. 36, no. 9, pp. 2546–2553, 2009.

[29] B. Almada-Lobo and R. J. James, "Neighbourhood search meta-heuristics for capacitated lot-sizing with sequence-dependent setups," *International Journal of Production Research*, vol. 48, no. 3, pp. 861–878, 2010.

[30] I.-S. Shim, H.-C. Kim, H.-H. Doh, and D.-H. Lee, "A two-stage heuristic for single machine capacitated lot-sizing and scheduling with sequence-dependent setup costs," *Computers & Industrial Engineering*, vol. 61, no. 4, pp. 920–929, 2011.

[31] F. M. T. Márcio AF Belo-Filho and B. Almada-Lobo, "Models for capacitated lot-sizing problem with backlogging, setup carryover and crossover," *Journal of the Operational Research Society*, vol. 65, no. 11, pp. 1735–1747, 2014.

[32] M. G. Wichmann, C. Johannes, and T. S. Spengler, "Energy-oriented lot-sizing and scheduling considering energy storages," *International Journal of Production Economics*, vol. 216, pp. 204–214, 2019.

[33] F. Hein, C. Almeder, G. Figueira, and B. Almada-Lobo, "Designing new heuristics for the capacitated lot sizing problem by genetic programming," *Computers & Operations Research*, vol. 96, pp. 1–14, 2018.

[34] Z. Degraeve and R. Jans, "A new dantzig-wolfe reformulation and branch-and-price algorithm for the capacitated lot-sizing problem with setup times," *Operations Research*, vol. 55, pp. 909–920, 2023/11/19/ 2007.

[35] D. M. Carvalho and M. C. Nascimento, "A kernel search to the multi-plant capacitated lot sizing problem with setup carry-over," *Computers & Operations Research*, vol. 100, pp. 43–53, 2018.

[36] C. Almeder, D. Klabjan, R. Traxler, and B. Almada-Lobo, "Lead time considerations for the multi-level capacitated lot-sizing problem," *European Journal of Operational Research*, vol. 241, no. 3, pp. 727–738, 2015.

[37] W. Kaczmarczyk, "Valid inequalities for proportional lot-sizing and scheduling problem with fictitious microperiods," *International Journal of Production Economics*, vol. 219, pp. 236–247, 2020.

[38] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers Operations Research*, vol. 134, p. 105400, 2021.

[39] Q. Wang and C. Tang, "Deep reinforcement learning for transportation network combinatorial optimization: A survey," *Knowledge-Based Systems*, vol. 233, p. 107526, 2021.

[40] T. D. Barrett, W. R. Clements, J. N. Foerster, and A. I. Lvovsky, "Exploratory combinatorial optimization with reinforcement learning," 2020.

[41] Q. Cappart, C. Andersen, D. Bergman, , and L.-M. Rousseau, "Combining reinforcement learning and constraint programming for combinatorial optimization," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, pp. 3677–3687, 2021.

[42] Y. Tang, S. Agrawal, and Y. Faenza, "Reinforcement learning for integer programming: Learning to cut," in *Proceedings of the 37th International Conference on Machine Learning* (H. D. III and A. Singh, eds.), vol. 119 of *Proceedings of Machine Learning Research*, pp. 9367–9376, PMLR, 13–18 Jul 2020.

[43] M. Goerigk and J. Kurtz, "Data-driven robust optimization using deep neural networks," *Computers Operations Research*, vol. 151, p. 106087, 2023.

[44] R. N. Monemi, S. Gelareh, and N. Maculan, "Solution algorithms for dock scheduling and truck sequencing in cross-docks: A neural branch-and-price and a metaheuristic," *Computers Operations Research*, vol. 167, p. 106604, 2024.

[45] J. Gerbaux, G. Desaulniers, and Q. Cappart, "A machine-learning-based column generation heuristic for electric bus scheduling," *Computers Operations Research*, vol. 173, p. 106848, 2025.

[46] M. Morabit, G. Desaulniers, and A. Lodi, "Machine-learning-based column selection for column generation," *Transportation Science*, vol. 55, no. 4, pp. 815–831, 2021.

[47] S.-N. Johnn, V.-A. Darvariu, J. Handl, and J. Kalcsics, "A graph reinforcement learning framework for neural adaptive large neighbourhood search," *Computers Operations Research*, vol. 172, p. 106791, 2024.

[48] Y. Liu, X. Wang, T. Meng, W. Ai, and K. Li, "Lg-gnn: Local and global information-aware graph neural network for default detection," *Computers Operations Research*, vol. 169, p. 106738, 2024.

[49] Q. Cappart, D. ChÃ©telat, E. B. Khalil, A. Lodi, C. Morris, and P. VeliÄkoviÄ‡, "Combinatorial optimization and reasoning with graph neural networks," *Journal of Machine Learning Research*, vol. 24, no. 130, pp. 1–61, 2023.

[50] M. Gasse, D. Chetelat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[51] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. de Freitas, "Learning to learn without gradient descent by gradient descent," 2017.

[52] V. Nair, S. Bartunov, F. Gimeno, I. von Glehn, P. Lichocki, I. Lobov, B. O'Donoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang, R. Addanki, T. Hapuarachchi, T. Keck, J. Keeling, P. Kohli, I. Ktena, Y. Li, O. Vinyals, and Y. Zwols, "Solving mixed integer programs using neural networks," 2021.

[53] E. B. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina, "Learning to branch in mixed integer programming," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, 2016.

[54] G. Zarpellon, J. Jo, A. Lodi, and Y. Bengio, "Parameterizing branch-and-bound search trees to learn branching policies," 2021.

[55] P. Gupta, M. Gasse, E. B. Khalil, M. P. Kumar, A. Lodi, and Y. Bengio, "Hybrid models for learning to branch," 2020.

APPENDICES

APPENDIX A. MIQP FORMULATION

The Mixed Integer Quadratic Programming (MIQP) formulation represents an earlier variant of the model introduced in CGNS. It was initially designed to eliminate the setup carryover variables, denoted as β, by modifying the structure of the objective function.

This formulation relaxes the Mixed Integer Linear Programming (MILP) model by removing the setup crossover variables, t^+ and t^-. Instead, it penalises setup crossover losses within the objective function, thereby approximating the cost impact associated with crossover setups. The penalty is expressed through the following terms:

$$\sum_{p\in P}\sum_{p'\in P}\sum_{l\in L}\sum_{t\in T}\alpha_{plt}^{T}Q_{pp'}\alpha_{p'lt+1}.$$

In this context, $Q_{pp'}$ represents the estimated reduced cost of transitioning from pattern p to pattern p', as defined in Equation-4.31. This penalty term is derived from reduced cost estimates obtained in previous iterations, allowing the model to account for crossover effects without introducing explicit binary variables.

While the MIQP formulation exhibits slightly improved computational performance relative to the proposed MILP model, it also presents challenges in maintaining feasibility. Given that the penalisation relies on estimations, additional steps are necessary to ensure solution validity. Specifically, a manual feasibility check must be performed after solving the MIQP to identify and eliminate infeasible columns from the column pool before advancing to the Relaxed LP (RLP) phase.

Ultimately, the MILP formulation is favoured in this study due to its structure, which facilitates better integration with machine learning models. In contrast to the MIQP approach, the MILP formulation leverages actual capacity constraint satisfaction and solution structure, enabling more reliable feature extraction and learning in data-driven optimisation contexts.

The mathematical formulation is presented below:

$$\begin{aligned}\min \sum_{i\in I}\sum_{t\in T}C_i^{inv}s_{it}+\sum_{i\in I}\sum_{t\in T}C_i^{back}b_{it}+\sum_{i\in I}\sum_{l\in L}\sum_{t\in T}C_l^{prod}T_i^{cyc}\Pi_i q_{ilt}\\ +\sum_{p\in P}\sum_{l\in L}\sum_{t\in T}C^{stp}T_p^{stp}\alpha_{plt}+\sum_{l\in L}\sum_{t\in T}C^{stp}T^{cap}w_{lt}^{f}\\ +\sum_{p\in P}\sum_{p'\in P}\sum_{l\in L}\sum_{t\in T}\alpha_{plt}^{T}Q_{pp'}\alpha_{p'lt+1}\end{aligned} \tag{A.1}$$

$$s_{it-1}+b_{it}+\sum_{l\in L}\Pi_i q_{ilt}-s_{it}-b_{it-1}=D_{it}\quad(\forall i\in I;\forall t\in T) \tag{A.2}$$

$$b_{i0}=B_i^0\qquad(\forall i\in I) \tag{A.3}$$

$$s_{i0} = S_i^0 \qquad (\forall i \in I) \tag{A.4}$$

$$\sum_{p \in P_i} \alpha_{plt} - q_{ilt} \geq 0 \qquad (\forall i \in I; \forall l \in L; \forall t \in T) \tag{A.5}$$

$$T^{cap} - \sum_{i \in I} T_i^{cyc} \Pi_i q_{ilt} - \sum_{p \in P} T_p^{st} \alpha_{plt} \geq 0 \quad (\forall l \in L; \forall t \in T) \tag{A.6}$$

$$\sum_{p \in P} \alpha_{plt} + w_{lt}^f = 1 \qquad (\forall l \in L; \forall t \in T) \tag{A.7}$$

$$\sum_{l \in L} w_{ml} \leq 1 \qquad (\forall m \in M) \tag{A.8}$$

$$\sum_{m \in M | \Gamma_{im} = 1} w_{ml} - \alpha_{plt} \geq 0 \qquad (p \in P; i \in I_p; l \in L; t \in T) \tag{A.9}$$

$$w_{ml} \leq \Lambda_{ml} \qquad (\forall m \in M; \forall l \in L) \tag{A.10}$$

APPENDIX B. HDFO ALGORITHM SUMMARY

The Hybrid Decomposition Fix-and-Optimise (HDFO) algorithm is a metaheuristic approach that integrates Relax-and-Fix, Machine-Based Local Search, and Time-Based Neighbourhood Search techniques for solving large-scale lot-sizing and scheduling problems. This appendix provides detailed explanations and mathematical formulations used in the HDFO algorithm. For further information, please refer to [13]

B.1 Randomised Least Flexible Machine (RLFM) Rule

Each machine's flexibility is measured as:

$$FI_m = \frac{|\{j \in J : a_{mj} = 1\}|}{|J|} \quad \forall m \in M \tag{B.1}$$

Here, a_{mj} is a binary parameter indicating whether machine m can process item j. Machines with lower FI_m are considered less flexible. The RLFM rule constructs a cumulative probability distribution p_{mj} over machines for each item j, where:

$$p_{mj} \propto \frac{1}{FI_m + \epsilon}, \quad \text{with small } \epsilon > 0 \text{ for numerical stability} \tag{B.2}$$

A machine m is then randomly selected for item j according to this distribution to form the machine-item assignment matrix $\bar{M}_{mj}$.

B.2 Alternating Decomposition Strategies

Machine-Based Decomposition Phase: The algorithm fixes binary variables related to all machines except for one selected machine m. All associated variables (production q_{ilt}, setup y_{ilt}, sequence z_{ijlt}) for this machine are optimised across the full planning horizon T:

$$\min \sum_{t=1}^{T} \left(\sum_{i \in I} (HC_i \cdot s_{it} + BC_i \cdot b_{it} + PC_l \cdot q_{ilt}) + \sum_{i,j \in I} SC \cdot ST_{ij} \cdot z_{ijlt} \right) \tag{B.3}$$

subject to: inventory balance, capacity, setup time/cost, and assignment constraints (B.4)

Time-Based Decomposition Phase: If no improvement is found in the machine-based phase, the algorithm selects a time subset $P \subset T$ and re-optimises all related variables in those periods:

$$\min \sum_{t \in P} (\cdots) \tag{B.5}$$

$$\text{subject to: all constraints active in periods } t \in P \tag{B.6}$$

The selection of P is biased using adaptive probabilities w_t based on recency and frequency of updates:

$$w_t = \frac{1}{\text{freq}_t + \lambda \cdot \text{recency}_t + \epsilon} \tag{B.7}$$

where λ controls the trade-off and ϵ avoids division by zero.

B.3 Tabu Lists and Selection History

During the iterative search, tabu lists TL_M and TL_T are used to store recently visited machines and time periods. These lists are updated after each iteration to:

- Prevent immediate reselection of recently optimised components.
- Encourage exploration of new neighbourhoods.
- Guide adaptive selection using frequency/recency-based weights.

B.4 Termination Conditions

The algorithm terminates when any of the following conditions are met:

1. Global time limit T_L is exceeded.
2. Maximum number of iterations $I_{\max}$ is reached.
3. No improvement in the best-known solution for δ consecutive iterations.

These stopping criteria ensure practical applicability for large-scale instances.

B.5 Algorithm Summary

Algorithm 10 Hybrid Decomposition Fix-and-Optimise (HDFO) Algorithm

Initialisation: Generate an initial solution using the Relax-and-Fix procedure:
  Partition the horizon into rolling time windows of length α with overlap μ
  Fix binary variables for $t < t_s$, optimise for $t_s \leq t \leq t_e$, relax otherwise
while $t_e < T$ **do**
  Solve subproblem SubMIP($\bar{Y}, \bar{Z}, w_m, d_t, \bar{M}_{mj}$)
  Update fixed binaries $\bar{Y} \leftarrow Y$, $\bar{Z} \leftarrow Z$
  Slide the window: $t_s \leftarrow t_e - c$, $t_e \leftarrow t_e + \alpha - \mu$
end while
if $t_e \geq T$ **then**
  Solve final subproblem with $t_e \leftarrow T$
end if
Improvement Phase (Two-Phase Search):
repeat
  Local Search (Machine-Based Decomposition):
  Select a machine m not in the tabu list
  Re-optimise all variables related to m across all periods
  Fix binary variables of all other machines
  Update $\bar{M}_{mj}$ using Randomised Least Flexible Machine (RLFM) rule
  Update tabu list
  if solution improves **then**
    Accept and continue
  else
    Neighbourhood Modification (Time-Based Decomposition):
    Select a subset of periods P (biased towards less frequently selected)
    Re-optimise all variables related to P
    Update tabu list
  end if
until termination condition met (e.g., time limit T_L)

- The RLFM rule assigns machines to items based on their flexibility, inversely, diversifying the search.
- The algorithm alternates between machine-based and time-based decompositions to avoid local minima.
- Tabu lists track recent selections to guide exploration.
- Termination occurs when a global time limit or maximum number of iterations is reached.

APPENDIX C. SODBS ALGORITHM SUMMARY

The Sequence-Oriented Decomposition with Block Search (SODBS) algorithm is a metaheuristic approach that integrates decomposition, block-wise sequence exploration, and selective re-optimisation to solve the capacitated lot-sizing and scheduling problem with sequence-dependent setups. This appendix presents the detailed structure and logic behind the SODBS algorithm. For additional technical details, refer to [10].

C.1 Sequence-Based Block Search Mechanism

A key mechanism in SODBS is to revise production sequences in promising blocks. For each machine l and selected time block $B \subseteq T$, the algorithm extracts current item sequences and defines alternative permutations of these blocks.

Block Candidate Construction: For each machine l and period block B:

- Let $S_{l,B}$ be the current sequence of items.
- Construct a set of neighbourhood sequences $\mathcal{N}(S_{l,B})$ by permuting or modifying subsets of $S_{l,B}$.
- Evaluate each candidate in $\mathcal{N}$ by fixing other variables and solving for updated decisions (y, z, q).

  This targets sequence quality and reduces computational overhead.

C.2 Adaptive Decomposition and Local Search

To enhance diversification:

- Machine-time pairs (l, B) are selected using adaptive frequency and performance scores.
- Previously visited sequences are stored to avoid cycles (Tabu memory).
- If no improvement is observed in η iterations, new block sizes or alternative machine subsets are tested.

C.3 Termination Conditions

SODBS terminates when any of the following are satisfied:

1. A global time budget T_L is exceeded.
2. The number of non-improving iterations exceeds δ.
3. All blocks have been exhaustively searched and no better configuration is found.

C.4 Algorithm Summary

Algorithm 11 Sequence-Oriented Decomposition with Block Search (SODBS).

Initialisation: Generate an initial solution using the Relax-and-Fix procedure:
  Partition the horizon into rolling time windows of length α with overlap μ
  Fix binary variables for $t < t_s$, optimise for $t_s \leq t \leq t_e$, relax otherwise
while $t_e < T$ **do**
  Solve subproblem SubMIP($\bar{Y}, \bar{Z}, w_m, d_t, \bar{M}_{mj}$)
  Update fixed binaries $\bar{Y} \leftarrow Y$, $\bar{Z} \leftarrow Z$
  Slide the window: $t_s \leftarrow t_e - c$, $t_e \leftarrow t_e + \alpha - \mu$
end while
if $t_e \geq T$ **then**
  Solve final subproblem with $t_e \leftarrow T$
end if
Iterative Block Search:
repeat
  Select machine l and time block B (adaptive frequency)
  Extract current sequence $S_{l,B}$
  Generate neighbourhood $\mathcal{N}(S_{l,B})$
  for all sequences $\sigma \in \mathcal{N}(S_{l,B})$ **do**
    Evaluate σ by fixing other machine/period variables
    Update solution if objective improves
  end for
  Update memory, block scores, and tabu list
until termination criteria met (e.g., time limit TL)

- Blocks allow local sequence changes without disrupting global feasibility.
- Adaptive selection ensures diverse machine and time combinations are explored.
- Optional neural prediction accelerates search by filtering unpromising permutations.

APPENDIX D. MACHINE LEARNING APPROACHES

In this section, we present the machine learning approaches studied during the PhD. studies. The approach we studied is to attack different parts of the proposed CGNS formulation and solution method.

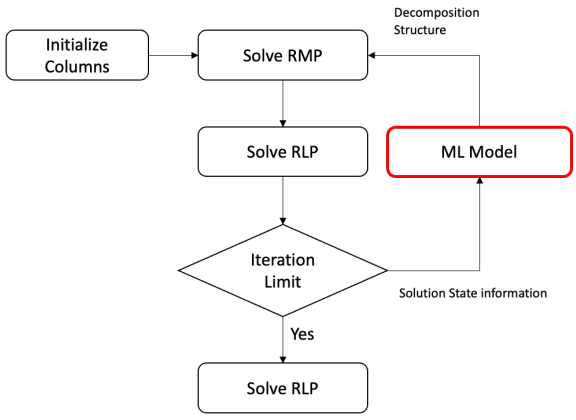
Table D.1: *Comparison of Neural Network-Based Learning Approaches.*

Approach #	NN Structure	Learning Mechanism	Improve RMP Solution Time	Improve Column Generation Mechanism
Approach 1	Deep NN	Supervised	✓	
Approach 2	Deep NN	Reinforcement		✓
Approach 3	Deep NN	Supervised + Reinforcement	✓	✓
Approach 4	RNN	Reinforcement	✓	✓
Approach 5	Q Learning	Reinforcement	✓	✓

D.1 Generate columns by machine learning (CGNSNN)

Our first approach is the selected one in the dissertation, which is to use NN in the column generation section of the CGNS algorithm. Figure-D.1 shows the overall structure of the CGNS algorithm, but we replace the mechanical column search heuristics with machine learning algorithms searching for different products i within the current solution.

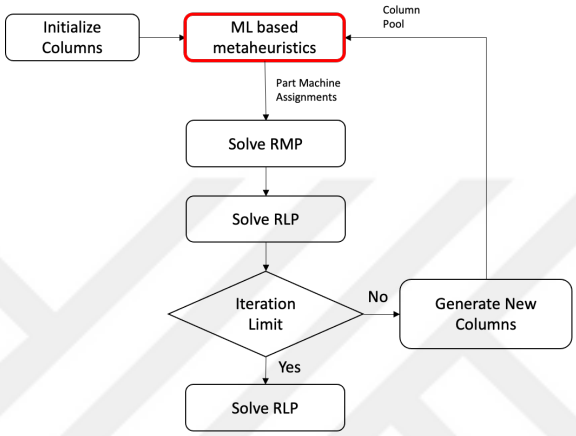
Figure D.1: *Approach 1: Overview.*



D.2 Use machine learning to predict the objective value of the current column pool

This approach uses a machine learning algorithm to predict the objective (outcome) of a generated column pool. This information will be used to eliminate iterations of the original CGNS algorithm by batch processing the generated column pools and selecting the most promising column pool to be used in the RMP.

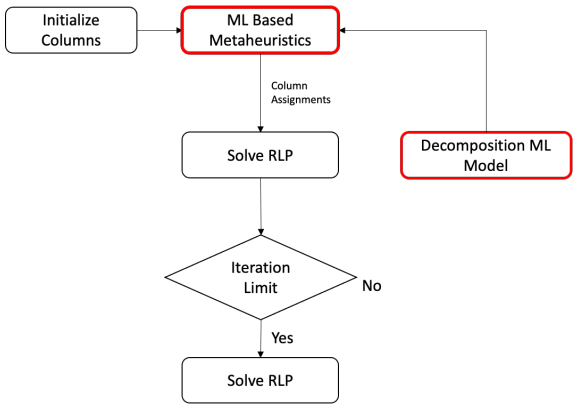
Figure D.2: *Approach 2: Overview.*



D.3 Use machine learning to predict and generate column pool (Approach 1 + Approach 2)

This approach is a combination of approaches 1 and 2, where the column generation process is handled by the machine learning algorithm, and further, it is also preprocessed to predict the possible outcome of the generated column pool to eliminate a number of iterations done within the RMP process.

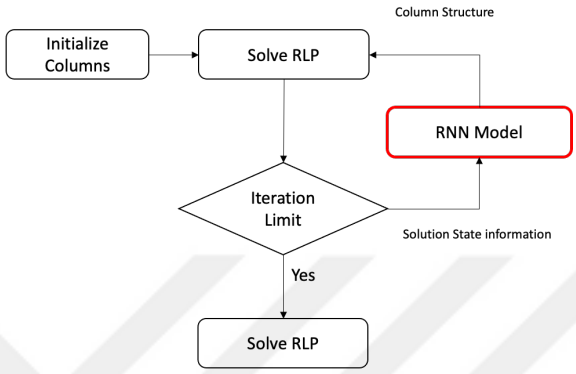
Figure D.3: *Approach 3: Overview.*



D.4 Use RNN based structure to generate sequences and eliminate RMP

This approach uses RNN to generate sequences of products (columns) at each iteration and evaluate the solution by the RLP iteratively to improve the objective.

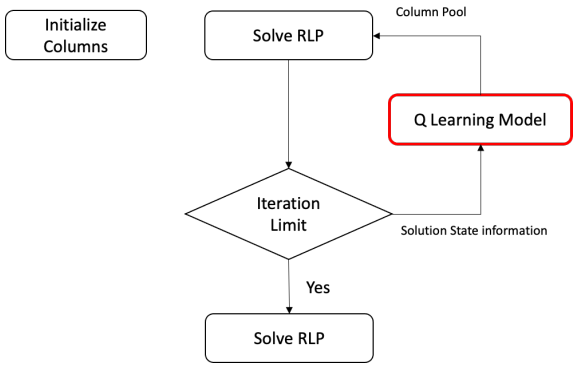
Figure D.4: *Approach 4: Overview.*



D.5 Use reinforcement learning (Q Learning) to modify incumbent solution

This approach uses Q learning to modify the current solution by operations like insert, delete and replace product i within the current column and iteratively improve the objective value.

Figure D.5: *Approach 5: Overview.*



VITA

Cevdet Utku ŞAFAK earned his Bachelor of Science in Mechanical Engineering from Middle East Technical University in 2010. He later completed his Master of Science in Industrial Engineering at Özyeğin University in 2018.

He commenced his professional career as an R&D Mechanical Design Engineer and subsequently took on the role of Program Manager at Vestel Electronics. In this position, he was responsible for designing and developing consumer electronics products while coordinating large-scale engineering programs until 2024. He has since transitioned into a consulting role at OzuBEX, concentrating on the digital transformation of industrial operations, optimising large-scale real-world challenges and using machine learning to solve industry problems.

He is pursuing his PhD in the Industrial Engineering Program at Özyeğin University, focusing his research on integrating machine learning techniques into operations research. His work investigates the development of intelligent and scalable optimisation algorithms for complex production planning, lot sizing, and scheduling issues. By merging mathematical modelling with data-driven and machine-learning approaches, his research aims to effectively and efficiently tackle the challenges of solving large-scale, complex industrial problems.